SEED Platform Documentation

Release 1.3.2

June 11, 2016

Contents

1 Getting Started **3**
1.1 Development Setup . . . 3

2 Deployment Guide **13**
2.1 AWS Setup . . . 13
2.2 General Linux Setup . . . 16
2.3 Monitoring . . . 21

3 API **23**
3.1 Authentication . . . 23
3.2 Payloads . . . 23
3.3 Responses . . . 24
3.4 API-related Endpoints . . . 24
3.5 Account Management Endpoints . . . 24
3.6 File Upload Endpoints . . . 32
3.7 SEED (Building and Project) Endpoints . . . 33

4 Data Model **53**
4.1 parents and children . . . 53
4.2 manual-matching vs auto-matching . . . 54
4.3 what really happens to the BuildingSnapshot table on import (and when) . . . 55
4.4 what really happens to the CanonicalBuilding table on import (and when) . . . 57
4.5 organization . . . 57
4.6 *_source_id fields . . . 57
4.7 extra_data . . . 58
4.8 saving and possible data loss . . . 58

5 Mapping **61**
5.1 Import . . . 61
5.2 Mapping . . . 61

6 Modules **63**
6.1 Audit Logs Package . . . 63
6.2 Cleansing Package . . . 67
6.3 Common Package . . . 70
6.4 Configuration . . . 72
6.5 Data Package . . . 73
6.6 Data Importer Package . . . 73
6.7 Features Package . . . 86

6.8 Green Button Package . . . 86
6.9 Landing Package . . . 89
6.10 Library Packages . . . 94
6.11 Mapping Package . . . 95
6.12 Managers Package . . . 97
6.13 Public Package . . . 97
6.14 SEED Package . . . 98
6.15 Serializers Package . . . 140
6.16 URLs Package . . . 141
6.17 Utilities Package . . . 141
6.18 Views Package . . . 145

7 Developer Resources **179**
7.1 General Notes . . . 179
7.2 Django Notes . . . 179
7.3 AngularJS Integration Notes . . . 180
7.4 Logging . . . 181
7.5 Testing . . . 181

8 License **183**

9 Help **185**
9.1 For SEED-Platform Users . . . 185
9.2 For SEED-Platform Developers . . . 185

10 Updating this documentation **187**

11 Indices and tables **189**

Python Module Index **191**

The Standard Energy Efficiency Data (SEED) Platform™ is a web-based application that helps organizations easily manage data on the energy performance of large groups of buildings. Users can combine data from multiple sources, clean and validate it, and share the information with others. The software application provides an easy, flexible, and cost-effective method to improve the quality and availability of data to help demonstrate the economic and environmental benefits of energy efficiency, to implement programs, and to target investment activity.

The SEED application is written in Python/Django, with AngularJS, Bootstrap, and other JavaScript libraries used for the front-end. The back-end database is required to be PostgreSQL.

The SEED web application provides both a browser-based interface for users to upload and manage their building data, as well as a full set of APIs that app developers can use to access these same data management functions.

Work on SEED Platform is managed by Lawrence Berkeley National Laboratory, with funding from the U.S. Department of Energy. The software was originally developed by Building Energy.

CHAPTER 1

Getting Started

1.1 Development Setup

1.1.1 Installation on OSX

These instructions are for installing and running SEED on Mac OSX in development mode.

Quick Installation Instructions

This section is intended for developers who may already have their machine ready for general development. If this is not the case, skip to Prerequisites.

- install Postgres 9.4 and redis for cache and message broker
- use a virtualenv (if desired)
- *git clone git@github.com:seed-platform/seed.git*
- create a *local_untracked.py* in the *config/settings* folder and add CACHE and DB config (example *local_untracked.py.dist*)
- *export DJANGO_SETTINGS_MODULE=config.settings.dev*
- *pip install -r requirements/local.txt*
- *./manage.py syncdb*
- *./manage.py migrate*
- *./manage.py create_default_user*
- *./manage.py runserver*
- *celery -A seed worker -l info -c 4 –maxtasksperchild 1000 –events*
- navigate to *http://127.0.0.1:8000/app/#/profile/admin* in your browser to add users to organizations
- main app runs at *127.0.0.1:8000/app*

The *python manage.py create_default_user* will setup a default *superuser* which must be used to access the system the first time. The management command can also create other superusers.

```
./manage.py create_default_user --username=demo@seed.lbl.gov --organization=lbl --password=demo123
```

Prerequisites

These instructions assume you have MacPorts or Homebrew. Your system should have the following dependencies already installed:

- git (*port install git* or *brew install git*)
- Mercurial (*port install hg* or *brew install mercurial*)
- graphviz (*brew install graphviz*)
- virtualenv and virtualenvwrapper (Recommended)

> **Note:** Although you *could* install Python packages globally, this is the easiest way to install Python packages. Setting these up first will help avoid polluting your base Python installation and make it much easier to switch between different versions of the code.

```
pip install virtualenv
pip install virtualenvwrapper
```

- Follow instructions on virtualenvwrapper to setup your environment.
- Once you have these installed, creating and entering a new virtualenv called "`seed`" for SEED development is by calling:

```
mkvirtualenv --python=python2.7 seed
```

PostgreSQL 9.4

MacPorts:

```
sudo su - root
port install postgresql94-server postgresql94 postgresql94-doc
# init db
mkdir -p /opt/local/var/db/postgresql94/defaultdb
chown postgres:postgres /opt/local/var/db/postgresql94/defaultdb
su postgres -c '/opt/local/lib/postgresql94/bin/initdb -D /opt/local/var/db/postgresql94/defaultdb'

# At this point, you may want to add start/stop scripts or aliases to
# ~/.bashrc or your virtualenv ``postactivate`` script
# (in ``~/.virtualenvs/{env-name}/bin/postactivate``).

alias pg_start='sudo su postgres -c "/opt/local/lib/postgresql94/bin/pg_ctl \
    -D /opt/local/var/db/postgresql94/defaultdb \
    -l /opt/local/var/db/postgresql94/defaultdb/postgresql.log start"'
alias pg_stop='sudo su postgres -c "/opt/local/lib/postgresql94/bin/pg_ctl \
    -D /opt/local/var/db/postgresql94/defaultdb stop"'

pg_start

sudo su - postgres
PATH=$PATH:/opt/local/lib/postgresql94/bin/
```

Homebrew:

```
brew install postgres
# follow the post install instructions to add to launchagents or call
```

```
# manually with `postgres -D /usr/local/var/postgres`
# Skip the remaining Postgres instructions!
```

Configure PostgreSQL. Replace 'seeddb', 'seeduser' with desired db/user. By default use password *seedpass* when prompted

```
createuser -P seeduser
createdb `whoami`
psql -c 'CREATE DATABASE "seeddb" WITH OWNER = "seeduser";'
psql -c 'GRANT ALL PRIVILEGES ON DATABASE "seeddb" TO seeduser;'
psql -c 'ALTER USER seeduser CREATEDB;'
psql -c 'ALTER USER seeduser CREATEROLE;'
```

Now exit any root environments, becoming just yourself (even though it's not that easy being green), for the remainder of these instructions.

Python Packages

Run these commands as your normal user id.

Change to a virtualenv (using virtualenvwrapper) or do the following as a superuser. A virtualenv is usually better for development. Set the virtualenv to seed.

```
workon seed
```

Make sure PostgreSQL command line scripts are in your PATH (if using port)

```
export PATH=$PATH:/opt/local/lib/postgresql94/bin
```

Some packages (uWSGI) may need to find your C compiler. Make sure you have 'gcc' on your system, and then also export this to the *CC* environment variable:

```
export CC=gcc
```

Install requirements with *pip*

```
pip install -r requirements/local.txt
```

Install library with *setup.py*

```
python setup.py install
```

NodeJS/npm

Install npm. You can do this by installing from nodejs.org, MacPorts, or Homebrew:

MacPorts:

```
sudo port install npm
```

Homebrew:

```
brew install npm
```

Configure Django and Databases

In the *config/settings* directory, there must be a file called *local_untracked.py* that sets up databases and a number of other things. To create and edit this file, start by copying over the template

```
cd config/settings
cp local_untracked.py.dist local_untracked.py
```

- Edit *local_untracked.py*. Open the file you created in your favorite

editor. The PostgreSQL config section will look something like this:

```
# postgres DB config
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'seeddb',
        'USER': 'seeduser',
        'PASSWORD': 'seedpass',
        'HOST': 'localhost',
        'PORT': '5432',
    }
}
```

You may want to comment out the AWS settings.

For Redis, edit the *CACHES* and *BROKER_URL* values to look like this:

```
CACHES = {
    'default': {
        'BACKEND': 'redis_cache.cache.RedisCache',
        'LOCATION': "127.0.0.1:6379",
        'OPTIONS': {'DB': 1},
        'TIMEOUT': 300
    }
}
BROKER_URL = 'redis://127.0.0.1:6379/1'
```

Run Django Migrations

Change back to the root of the repository. Now run the migration script to set up the database tables

```
export DJANGO_SETTINGS_MODULE=config.settings.dev
./manage.py migrate
```

Django Admin User

You need a Django admin (super) user.

```
./manage.py create_default_user --username=admin@my.org --organization=lbnl --password=badpass
```

Of course, you need to save this user/password somewhere, since this is what you will use to login to the SEED website.

If you want to do any API testing (and of course you do!), you will need to add an API KEY for this user. You can do this in postgresql directly:

```
psql seeddb seeduser
seeddb=> update landing_seeduser set api_key='DEADBEEF' where id=1;
```

The 'secret' key DEADBEEF is hard-coded into the test scripts.

Install Redis

You need to manually install Redis for Celery to work.

MacPorts:

```
sudo port install redis
```

Homebrew:

```
brew install redis
# follow the post install instructions to add to launchagents or
# call manually with `redis-server`
```

Install JavaScript Dependencies

The JS dependencies are installed using node.js package management (npm), with a helper package called *bower*.

```
./bin/install_javascript_dependencies.sh
```

Start the Server

You should put the following statement in ~/.bashrc or add it to the virtualenv post-activation script (e.g., in *~/.virtualenvs/seed/bin/postactivate*).

```
export DJANGO_SETTINGS_MODULE=config.settings.dev
```

The combination of Redis, Celery, and Django have been encapsulated in a single shell script, which examines existing processes and doesn't start duplicate instances:

```
./bin/start-seed.sh
```

When this script is done, the Django stand-alone server will be running in the foreground.

Login

Open your browser and navigate to http://127.0.0.1:8000

Login with the user/password you created before, e.g., *admin@my.org* and *badpass*.

Note: these steps have been combined into a script called *start-seed.sh*. The script will also try to not start Celery or Redis if they already seem to be running.

1.1.2 Installation on Docker

1.1.3 Installing on Windows and OS X

Install Docker Toolbox (Windows/OS X)

Install Docker-Toolbox, which installs several applications including Docker, Docker Machine, and Docker Compose. It is possible to install these individually as well by installing Docker, Docker-Machine, and Docker-Compose.

Create Docker-Machine Image

The command below will create a 100GB volume for development. This is a very large volume and can be adjusted. Make sure to create a volume greater than 30GB.

```
docker-machine create --virtualbox-disk-size 100000 -d virtualbox dev
```

Start Docker-Machine Image

```
docker-machine start dev  # if not already running
eval $(docker-machine env dev)
```

Run Docker Compose

```
docker-compose build
```

Be Patient ... If the containers build successfully, then start the containers

```
docker-compose up
```

Note that you may need to build the containers a couple times for everything to converge

Create User

```
docker-compose run web ./manage.py create_default_user
```

Login

Get the Docker IP address (*docker-machine ip dev*) and point your browser at http://*ip-address*:8000 and log in with the account:

```
username: demo@seed.lbl.gov
password: demo
```

1.1.4 Installing Docker on Ubuntu

Docker will run directly on Ubuntu, so, in contrast to Windows/OS X there is no need to run in in Virtual Box, unless desired. Running without Virtual Box means you can set it up so that local changes are reflected in the containers and you can edit files etc with your normal setup.

First follow the instructions here: https://docs.docker.com/engine/installation/linux/ubuntulinux/

If you set up a docker group and add yourself to it you can run docker commands directly. Otherwise you will need to precede docker commands with sudo. You will need to log out entirely for the changes to take place. You can test this by running

```
docker run hello-world
```

if you still have issues, try rebooting.

If you ran the hello-word docker you can use the following to clean up. First check to see what existing containers there are, and what there status is:

```
docker ps -a
```

You should see something similar to this if there are no containers.

:: CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

If you ran the hello world container you should see it listed (the image is hello-word).Check its status to make sure it exited. Then you can go ahead and remove it. A quick way to remove all old containers is this.

```
docker rm $(docker ps -a -q)
```

Otherwise specify the numeric id to remove individual containers.

Next you can list images in a similar way.

```
docker images
```

Images not connected to a container are known as dangling images. You can get rid of them using this command:

```
docker rmi -f $(docker images -q -a -f dangling=true)
```

Otherwise they can be removed using *docker rmi image* using the image name or id shown by docker images.

Install Docker Compose

```
sudo apt-get install python-pip
sudo pip install docker-compose
```

Optionally install Virtual Box and Docker-Machine

This is only necessary if you want to run inside Virtual Box.

```
sudo apt-get install virtual-box
wget https://github.com/docker/machine/releases/download/v0.7.0/docker-machine-`uname -s`-`uname -m`
sudo mv docker-$(uname -s)-$(uname -m) /usr/local/bin/docker-machine
sudo chmod +x /usr/local/bin/docker-machine
```

If you do this proceed by following the instructions for Windows/OS X starting from Create Docker-Machine Image.

Setting up without a Virtual Machine

If we don't use Virtual Box we can run Docker directly. This assumes you are in the same directory as the Git repo. You should also set up a virtualenv for it. Setting it up this way means it will use your local ip, so you will be able to access the SEED website via localhost. As we are using containers we don't have to worry about setting up the database and Redis directly, Docker will do this for us.

In this part we are going to set up the project so that the seed directory in web container's root file system points to the copy on your local file system (i.e. the directory with the repo in it). This is an advantage of running docker directly: changes on your local file system show up in the container so you can edit with your local tools etc. without having to have them running in the container.

Before you start ensure you have set up a virtualenv for the project. Then at a minimum you will need to install the tos module manually.

```
pip install -e  'git+https://github.com/revsys/django-tos.git@aca823ccd12fdb897b2827832458b3c34e91dee
```

Note the quotes.

If you notice complaints about this not being present try: *pip install ip install -r requirements/base.txt*, you might also need to install test and local

Edit *docker-compose.yml* in the repo base.

Look for the section web:, then underneath it the volumes: section. Add two lines like this:

- $HOME/projects/seed:/seed - $HOME/.virtualenvs/seed/src/django-tos-master:/seed/src/django-tos-master

You will to change the part before the colon to match your local setup. On my system the repo is a directory called seed under the projects folder in my home directory for the first line. In the second line my virtualenvs live under .virtualenvs in my home directory as I use virtualenv wrapper. You will need to adjust this to match your local setup.

Then you will need to open the ports for Redis and PostgreSQL. In the section *db-postgres:* add

::

ports:

- "5432:5432"

in db-redis add

::

ports:

- "6379:6379"

You should be careful not to add the changes to this file to your git commits as it is local only. You can do this with the following command.

```
git update-index  --skip-worktree docker-compose.yml
```

Doing this ensures git preserves your local changes and will warn you of any conflicts caused by upstream changes. Occasionally it might be necessary to temporarily unset the flag using *–no-skip-worktree* (you can reset it afterwards). You can find more on how acts, and how to fix conflicts here: http://fallengamer.livejournal.com/93321.html

Next do the following to create a local settings file

```
cp config/settings/local_untracked.py.dist config/settings/local_untracked.py
```

Then you will need to edit the databases section. Here is a sample

```
# postgres DB config
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'seed',
        'USER': 'seed',
        'PASSWORD': 'seed',
        'HOST': '172.17.0.1',
        'PORT': '5432',
```

```
    }
}
```

The tricky part is the HOST line. The web server can't connect to the database on localhost with this setup. Use *ifconfig* to find out the ip addresses on your local machine. In this example 172.17.0.1 was listed for docker0 and that worked.

Run Docker Compose

```
docker-compose build
```

Note this process will spit out a warning that some containers are being ignored. Don't worry they will be set up later.

Be Patient ... If the containers build successfully, then start the containers

```
docker-compose up
```

Note that you may need to build the containers a couple times for everything to converge. You will likely need to do this. Run *docker-compose up* hit Ctrl-C, then run both the commands again to get everything working correctly.

Note for whatever reason things like collectstatic aren't run automatically if you aren't using Virtual Box. You can fix it with the following. Use this to connect to a shell in the container.

```
docker exec -it "seed_web_1" bash
```

Then run the following when you are there.

```
bin/postcompile
```

You might see some errors, don't worry, these mostly occur because its trying to use Amazon S3., which is not relevant here.

This should only need to be done once (unless things change, e.g. adding more static files) as long as the docker image is around.

Create User

```
docker-compose run web ./manage.py create_default_user
```

Login

Point your browser at http://127.0.0.1:8000 and log in with the account:

```
username: demo@seed.lbl.gov
password: demo
```

CHAPTER 2

Deployment Guide

SEED is intended to be installed on Linux instances in the cloud (e.g. AWS), and on local hardware. SEED Platform does not officially support Windows for production deployment. If this is desired, see the Django notes.

2.1 AWS Setup

Amazon Web Services (AWS) provides the preferred hosting for SEED.

seed is a Django Project and Django's documentation is an excellent place for general understanding of this project's layout.

2.1.1 Prerequisites

Ubuntu server 13.10 or newer, with the following list of *aptitude packages* installed.

Copy the *prerequisites.txt* files to the server and install the dependencies:

```
$ sudo dpkg --set-selections < ./prerequisites.txt
$ sudo apt-get dselect-upgrade
```

or with a single command as `su`

```
# aptitude install $(cat ./prerequisites.txt | awk '{print $1}')
```

Note: PostgresSQL server is not included above, and it is assumed that the system will use the AWS RDS PostgresSQL service

Note: postgresql `>=9.3` is required to support JSON Type

A smaller list of packages to get going:

```
$ sudo apt-get install python-pip python-dev libatlas-base-dev gfortran \
python-dev build-essential g++ npm libxml2-dev libxslt1-dev \
postgresql-devel postgresql-9.3 postgresql-server-dev-9.3 libpq-dev \
libmemcached-dev openjdk-7-jre-headless
```

Amazon Web Services (AWS) Dependencies

The following AWS services are used for **seed**:

- RDS (PostgreSQL >=9.3)
- ElastiCache (redis)
- SES
- S3

2.1.2 Python Dependencies

Clone the **seed** repository from **github**

```
$ git clone git@github.com:SEED-platform/seed.git
```

enter the repo and install the python dependencies from requirements

```
$ cd seed
$ sudo pip install -r requirements/local.txt
```

2.1.3 JavaScript Dependencies

`npm` is required to install the JS dependencies. The `bin/install_javascript_dependencies.sh` script will download all JavaScript dependencies and build them. `bower` and `gulp` should be installed globally for convenience.

```
$ sudo apt-get install build-essential
$ sudo apt-get install libssl-dev
$ sudo apt-get install curl
```

```
$ sudo npm install -g bower gulp
$ bin/install_javascript_dependencies.sh
```

2.1.4 Database Configuration

Copy the `local_untracked.py.dist` file in the `config/settings` directory to `config/settings/local_untracked.py`, and add a `DATABASES` configuration with your database username, password, host, and port. Your database configuration can point to an AWS RDS instance or a PostgreSQL 9.4 database instance you have manually installed within your infrastructure.

```
# Database
DATABASES = {
    'default': {
        'ENGINE':'django.db.backends.postgresql_psycopg2',
        'NAME': 'seed',
        'USER': '',
        'PASSWORD': '',
        'HOST': '',
        'PORT': '',
    }
}
```

Note: other databases could be used such as MySQL, but are not supported due to the postgres-specific JSON Type

In in the above database configuration, `seed` is the database name, this is arbitrary and any valid name can be used as long as the database exists.

create the database within the postgres `psql` shell:

```
postgres-user=# CREATE DATABASE seed;
```

or from the command line:

```
$ createdb seed
```

create the database tables and migrations:

```
$ python manage.py syncdb
$ python manage.py migrate
```

Note: running migrations can be shortened into a one-liner `./manage.py syncdb --migrate`

create a superuser to access the system

```
$ python manage.py create_default_user --username=demo@example.com --organization=example --password=
```

Note: Every user must be tied to an organization, visit `/app/#/profile/admin` as the superuser to create parent organizations and add users to them.

2.1.5 cache and message broker

The SEED project relies on redis for both cache and message brokering, and is available as an AWS ElastiCache service. `local_untracked.py` should be updated with the `CACHES` and `BROKER_URL` settings.

```
CACHES = {
    'default': {
        'BACKEND': 'redis_cache.cache.RedisCache',
        'LOCATION': "seed-core-cache.ntmprk.0001.usw2.cache.amazonaws.com:6379",
        'OPTIONS': { 'DB': 1 },
        'TIMEOUT': 300
    }
}
BROKER_URL = 'redis://seed-core-cache.ntmprk.0001.usw2.cache.amazonaws.com:6379/1'
```

Note: The popular `memcached` can also be used as a cache back-end, but is not supported and redis has a different cache key format, which could cause breakage and isn't tested. Likewise, `rabbitmq` or AWS `SQS` are alternative message brokers, which could cause breakage and is not tested.

2.1.6 running celery the background task worker

Celery is used for background tasks (saving data, matching, creating projects, etc) and must be connected to the message broker queue. From the project directory, `celery` can be started:

```
$ python manage.py celery worker -B -c 2 --loglevel=INFO -E --maxtasksperchild=1000
```

2.1.7 running the development web server

The Django dev server (not for production use) can be a quick and easy way to get an instance up and running. The dev server runs by default on port 8000 and can be run on any port. See Django's runserver documentation for more options.

```
$ python manage.py runserver
```

2.1.8 running a production web server

Our recommended web server is uwsgi sitting behind nginx. The `bin/start_uwsgi.sh` script can been created to start `uwsgi` assuming your Ubuntu user is named `ubuntu`.

Also, static assets will need to be moved to S3 for production use. The `bin/post_compile` script contains a list of commands to move assets to S3.

```
$ bin/post_compile
```

```
$ bin/start_uwsgi
```

The following environment variables can be set within the `~/.bashrc` file to override default Django settings.

```
export SENTRY_DSN=https://xyz@app.getsentry.com/123
export DEBUG=False
export ONLY_HTTPS=True
```

2.2 General Linux Setup

While Amazon Web Services (AWS) provides the preferred hosting for SEED, running on a bare-bones Linux server follows a similar setup, replacing the AWS services with their Linux package counterparts, namely: PostgreSQL and Redis.

seed is a Django project and Django's documentation is an excellent place to general understanding of this project's layout.

2.2.1 Prerequisites

Ubuntu server 14.04 or newer

We need to install the base packages needed to run the app:

```
$ sudo apt-get update
$ sudo apt-get upgrade
$ sudo apt-get install libpq-dev python-dev python-pip libatlas-base-dev \
gfortran build-essential g++ npm libxml2-dev libxslt1-dev git mercurial \
libssl-dev curl uwsgi-core uwsgi-plugin-python
$ sudo apt-get install redis-server
$ sudo apt-get install postgresql postgresql-contrib
```

Note: postgresql >=9.3 is required to support JSON Type

2.2.2 Configure PostgreSQL

```
$ sudo su - postgres
$ createdb "seed-deploy"
$ createuser -P DBUsername
$ psql
postgres=# GRANT ALL PRIVILEGES ON DATABASE "seed-deploy" TO DBUsername;
postgres=# \q;
$ exit
```

Note: Any database name and username can be used here in place of "seed-deploy" and DBUsername

2.2.3 Python Dependencies

clone the **seed** repository from **github**

```
$ git clone git@github.com:SEED-platform/seed.git
```

enter the repo and install the python dependencies from requirements

```
$ cd seed
$ sudo pip install -r requirements/local.txt
```

2.2.4 JavaScript Dependencies

`npm` is required to install the JS dependencies. The `bin/install_javascript_dependencies.sh` script will download all JavaScript dependencies and build them. `bower` and `gulp` should be installed globally for convenience.

```
$ curl -sL https://deb.nodesource.com/setup_5.x | sudo -E bash -
$ sudo apt-get install -y nodejs
$ sudo npm install -g bower gulp
```

```
$ bin/install_javascript_dependencies.sh
```

2.2.5 Django Database Configuration

Copy the `local_untracked.py.dist` file in the `config/settings` directory to `config/settings/local_untracked.py`, and add a `DATABASES` configuration with your database username, password, host, and port. Your database configuration can point to an AWS RDS instance or a PostgreSQL 9.4 database instance you have manually installed within your infrastructure.

```
# Database
DATABASES = {
    'default': {
        'ENGINE':'django.db.backends.postgresql_psycopg2',
        'NAME': 'seed-deploy',
```

```
        'USER': 'DBUsername',
        'PASSWORD': '',
        'HOST': 'localhost',
        'PORT': '5432',
    }
}
```

Note: Other databases could be used such as MySQL, but are not supported due to the postgres-specific JSON Type

In in the above database configuration, `seed` is the database name, this is arbitrary and any valid name can be used as long as the database exists. Enter the database name, user, password you set above.

The database settings can be tested using the Django management command, `./manage.py dbshell` to connect to the configured database.

create the database tables and migrations:

```
$ python manage.py syncdb
$ python manage.py migrate
```

Note: running migrations can be shortened into a one-liner `./manage.py syncdb --migrate`

2.2.6 Cache and Message Broker

The SEED project relies on redis for both cache and message brokering, and is available as an AWS ElastiCache service or with the `redis-server` Linux package. (`sudo apt-get install redis-server`)

`local_untracked.py` should be updated with the `CACHES` and `BROKER_URL` settings.

```
CACHES = {
    'default': {
        'BACKEND': 'redis_cache.cache.RedisCache',
        'LOCATION': "127.0.0.1:6379",
        'OPTIONS': {'DB': 1},
        'TIMEOUT': 300
    }
}
BROKER_URL = 'redis://127.0.0.1:6379/1'
```

Note: The popular `memcached` can also be used as a cache back-end, but is not supported and redis has a different cache key format, which could cause breakage and isn't tested. Likewise, `rabbitmq` or AWS `SQS` are alternative message brokers, which could cause breakage and is not tested.

2.2.7 Creating the initial user

create a superuser to access the system

```
$ python manage.py create_default_user --username=demo@example.com --organization=example --password=
```

Note: Every user must be tied to an organization, visit `/app/#/profile/admin` as the superuser to create parent organizations and add users to them.

2.2.8 Running celery the background task worker

Celery is used for background tasks (saving data, matching, creating projects, etc) and must be connected to the message broker queue. From the project directory, `celery` can be started:

```
$ python manage.py celery worker -B -c 2 --loglevel=INFO -E --maxtasksperchild=1000
```

2.2.9 Running the development web server

The Django dev server (not for production use) can be a quick and easy way to get an instance up and running. The dev server runs by default on port 8000 and can be run on any port. See Django's runserver documentation for more options.

```
$ python manage.py runserver --settings=config.settings.dev
```

2.2.10 Running a production web server

Our recommended web server is uwsgi sitting behind nginx. The python package `uwsgi` is needed for this, and should install to `/usr/local/bin/uwsgi` Since AWS S3, is not being used here, we recommend using `dj-static` to load static files.

Note: The use of the `dev` settings file is production ready, and should be used for non-AWS installs with `DEBUG` set to `False` for production use.

```
$ sudo pip install uwsgi dj-static
```

Generate static files:

```
$ sudo ./manage.py collectstatic --settings=config.settings.dev
```

Update `config/settings/local_untracked.py`:

```
DEBUG = False
# static files
STATIC_ROOT = 'collected_static'
STATIC_URL = '/static/'
```

Start the web server:

```
$ sudo /usr/local/bin/uwsgi --http :80 --module standalone_uwsgi --max-requests 5000 --pidfile /tmp/u
```

Warning: Note that uwsgi has port set to `80`. In a production setting, a dedicated web server such as NGINX would be receiving requests on port 80 and passing requests to uwsgi running on a different port, e.g 8000.

2.2.11 environmental variables

The following environment variables can be set within the `~/.bashrc` file to override default Django settings.

```
export SENTRY_DSN=https://xyz@app.getsentry.com/123
export DEBUG=False
export ONLY_HTTPS=True
```

2.2.12 SMTP service

In the AWS setup, we use SES to provide an email service Django can use as an email backend and configured it in our config/settings/main.py:

```
EMAIL_BACKEND = 'django_ses.SESBackend'
```

Many options for setting up your own SMTP service/server or using other SMTP third party services are available and compatible including gmail.

Django can likewise send emails via python's smtplib with sendmail or postfix installed. See their docs for more info.

```
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
```

2.2.13 local_untracked.py

```
# PostgreSQL DB config
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'seed',
        'USER': 'your-username',
        'PASSWORD': 'your-password',
        'HOST': 'your-host',
        'PORT': 'your-port',
    }
}

# config for local storage backend
DEFAULT_FILE_STORAGE = 'django.core.files.storage.FileSystemStorage'
STATICFILES_STORAGE = DEFAULT_FILE_STORAGE
DOMAIN_URLCONFS = {}
DOMAIN_URLCONFS['default'] = 'urls.main'

CACHES = {
    'default': {
        'BACKEND': 'redis_cache.cache.RedisCache',
        'LOCATION': "127.0.0.1:6379",
        'OPTIONS': {'DB': 1},
        'TIMEOUT': 300
    }
}

# redis celery config
BROKER_URL = 'redis://127.0.0.1:6379/1'
CELERY_DEFAULT_QUEUE = 'seed-dev'
CELERY_QUEUES = (
    Queue(
```

```
        CELERY_DEFAULT_QUEUE,
        Exchange(CELERY_DEFAULT_QUEUE),
        routing_key=CELERY_DEFAULT_QUEUE
    ),
)

# SMTP config
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'

# static files
STATIC_ROOT = 'collected_static'
STATIC_URL = '/static/'
```

2.3 Monitoring

2.3.1 Flower

Flower is used to monitor background tasks. Assuming your redis broker is running on *localhost* and on port *6379*, DB *1*. Then go to localhost:5555 to check celery. If running on AWS, the *bin/start_flower.sh* will start flower on port *8080* and be available for Google credentialed accounts.

```
flower --port=5555 --broker=redis://localhost:6379/1`
```

2.3.2 Sentry

Todo

Fill this out

CHAPTER 3

API

3.1 Authentication

Authentication is handled via an authorization token set in an http header. To request an API token, go to `/app/#/profile/developer` and click 'Get a New API Key'.

Every request must include an 'Authorization' http header made up of your username (email) and your API key, separated with a ':'. For example, with curl:

```
curl -H Authorization:user@email_address.com:5edfd7f1f0696d4139118f8b95ab1f05d0dd418e https://seeddor
```

Or using the Python Requests library:

```
headers = {'authorization': 'user@email_address.com:5edfd7f1f0696d4139118f8b95ab1f05d0dd418e'}
result = requests.get('https://seeddomain.com/app/api/get_api_schema/', headers=headers)
print result.json()
```

If authentication fails, the response's status code will be 302, redirecting the user to `/app/login`.

3.2 Payloads

Many requests require a JSON-encoded payload and/or parameters in the query string of the url. A frequent requirement is including the organization_id of the org you belong to. E.g.:

```
curl -H Authorization:user@email_address.com:5edfd7f1f0696d4139118f8b95ab1f05d0dd418e \
  https://seeddomain.com/app/accounts/get_organization?organization_id={your org id here}
```

Or in a JSON payload:

```
curl -H Authorization:user@email_address.com:5edfd7f1f0696d4139118f8b95ab1f05d0dd418e \
  -d '{"organization_id":6, "user_id": 12, "role": "viewer"}' \
  https://seeddomain/app/accounts/update_role/
```

Using Python:

```
headers = {'authorization': 'user@email_address.com:5edfd7f1f0696d4139118f8b95ab1f05d0dd418e'}
params = json.dumps({'organization_id': 6, 'user_id': 12, 'role': 'viewer'})
result = requests.post('https://seeddomain.com/app/accounts/update_role/',
                       data=params,
                       headers=headers)
print result.json()
```

3.3 Responses

Responses from all requests will be JSON-encoded objects, as specified in each endpoint's documentation. In the case of an error, most endpoints will return this instead of the expected payload (or an HTTP status code):

```
{
    'status': 'error',
    'message': 'explanation of the error here'
}
```

3.4 API-related Endpoints

seed.views.api.**get_api_schema**()

URI: /app/api/get_api_schema/

Returns a hash of all API endpoints and their descriptions.

Returns:

```
    {
        '/example/url/here': {
            'name': endpoint name,
            'description': endpoint description
        }...
    }
```

Todo

Should this require authentication? Should it limit the return to endpoints the user has authorization for?

Todo

Format docstrings better.

3.5 Account Management Endpoints

seed.views.accounts.**add_org**()

URI: /app/accounts/add_org/

Creates a new organization.

Payload:

```
    {
        'organization_name': The name of the new org,
        'user_id': the user id of the owner of the new org,
    }
```

Returns:

```
{
    'status': 'success' or 'error',
    'message': 'message, if any',
    'organization_id': The ID of the new org, if created.
}
```

seed.views.accounts.**add_user**()

URI: /app/accounts/add_user/

Creates a new SEED user. One of 'organization_id' or 'org_name' is needed. Sends invitation email to the new user.

Payload:

```
{
    'organization_id': ID of an existing org to add the new user to,
    'org_name': Name of a new org to create with user as owner
    'first_name': First name of new user
    'last_name': Last name of new user
    'role': {
        'value': The permission level of new user within this org
            (one of member, viewer, owner)
    },
    'email': Email address of new user.
}
```

Returns:

```
{
    'status': 'success',
    'message': email address of new user,
    'org': name of the new org (or existing org),
    'org_created': True if new org created,
    'username': Username of new user
}
```

seed.views.accounts.**add_user_to_organization**()

URI: /app/accounts/add_user_to_organization/

Adds an existing user to an organization.

Payload:

```
{
    'organization_id': The ID of the organization,
    'user_id': the user id of the owner of the new org,
}
```

Returns:

```
{
    'status': 'success' or 'error',
    'message': 'message, if any',
}
```

seed.views.accounts.**create_sub_org**()

URI: /app/accounts/create_sub_org/

Creates a child org of a parent org.

Payload:

```
{
    'parent_org_id': ID of the parent org,
    'sub_org': {
        'name': Name of new sub org,
        'email': Email address of owner of sub org, which
                must already exist
    }
}
```

Returns:

```
{
    'status': 'success' or 'error',
    'message': Error message, if any,
    'organization_id': ID of newly-created org
}
```

seed.views.accounts.**get_cleansing_rules**()

URI: /app/accounts/get_cleansing_rules/

Returns the cleansing rules for an org.

Parameters **request** –

GET Expects organization_id in the query string.

Returns:

```
{
 'status': 'success',
 'in_range_checking': An array of in-range error rules,
 'missing_matching_field': An array of fields to verify existence,
 'missing_values': An array of fields to ignore missing values
}
```

seed.views.accounts.**get_organization**()

URI: /app/accounts/get_organization/

Retrieves a single organization by id.

GET Expects ?organization_id=(:org_id)

Returns:

```
{'status': 'success or error', 'message': 'error message, if any',
 'organization':
    {'name': org name,
     'org_id': org's identifier (used with Authorization header),
     'id': org's identifier,
     'number_of_users': count of members of org,
     'user_is_owner': True if the user is owner of this org,
     'user_role': The role of user in this org (owner, viewer, member),
     'owners': [
         {
          'first_name': the owner's first name,
          'last_name': the owner's last name,
          'email': the owner's email address,
          'id': the owner's identifier (int)
          }
         ]
      'sub_orgs': [ a list of orgs having this org as parent, in
                    the same format...],
```

```
          'is_parent': True if this org contains suborgs,
          'num_buildings': Count of buildings belonging to this org
        }
    }
```

seed.views.accounts.**get_organizations**()

URI: /app/accounts/get_organizations/

Retrieves all orgs the user has access to.

Returns:

```
    {'organizations': [
        {'name': org name,
         'org_id': org's identifier (used with Authorization header),
         'id': org's identifier,
         'number_of_users': count of members of org,
         'user_is_owner': True if the user is owner of this org,
         'user_role': The role of user in this org (owner, viewer, member),
         'owners': [
                    {
                     'first_name': the owner's first name,
                     'last_name': the owner's last name,
                     'email': the owner's email address,
                     'id': the owner's identifier (int)
                    }
                  ]
         'sub_orgs': [ a list of orgs having this org as parent, in
                     the same format...],
         'is_parent': True if this org contains suborgs,
         'num_buildings': Count of buildings belonging to this org
        }...
       ]
    }
```

seed.views.accounts.**get_organizations_users**()

URI: /app/accounts/get_organizations_users/

Retrieve all users belonging to an org.

Payload:

```
    {'organization_id': org_id}
```

Returns:

```
    {'status': 'success',
     'users': [
        {
         'first_name': the user's first name,
         'last_name': the user's last name,
         'email': the user's email address,
         'id': the user's identifier (int),
         'role': the user's role ('owner', 'member', 'viewer')
        }
      ]
    }
```

Todo

check permissions that request.user is owner or admin and get more info about the users.

seed.views.accounts.**get_query_threshold**()

URI: /app/accounts/get_query_threshold/

Returns the "query_threshold" for an org. Searches from members of sibling orgs must return at least this many buildings from orgs they do not belong to, or else buildings from orgs they don't belong to will be removed from the results.

GET Expects organization_id in the query string.

Returns:

```
{
 'status': 'success',
 'query_threshold': The minimum number of buildings that must be
     returned from a search to avoid squelching non-member-org results.
}
```

seed.views.accounts.**get_shared_fields**()

URI: /app/accounts/get_shared_fields/

Retrieves all fields marked as shared for this org tree.

GET Expects organization_id in the query string.

Returns:

```
{
 'status': 'success',
 'shared_fields': [
     {
      "title": Display name of field,
      "sort_column": database/search name of field,
      "class": css used for field,
      "title_class": css used for title,
      "type": data type of field,
          (One of: 'date', 'floor_area', 'link', 'string', 'number')
      "field_type": classification of field.  One of:
          'contact_information', 'building_information',
          'assessor', 'pm',
      "sortable": True if buildings can be sorted on this field,
     }
     ...
   ],
   'public_fields': [
       {
          "title": Display name of field,
          "sort_column": database/search name of field,
          "class": css used for field,
          "title_class": css used for title,
          "type": data type of field,
            (One of: 'date', 'floor_area', 'link', 'string', 'number')
          "field_type": classification of field.  One of:
              'contact_information', 'building_information',
              'assessor', 'pm',
          "sortable": True if buildings can be sorted on this field,
         }
         ...
```

```
        ]
    }
```

seed.views.accounts.**get_user_profile**()
URI: /app/accounts/get_user_profile/

Retrieves the request's user's first_name, last_name, email and api key if exists.

Returns:

```
    {
        'status': 'success',
        'user': {
            'first_name': user's first name,
            'last_name': user's last name,
            'email': user's email,
            'api_key': user's API key
        }
    }
```

seed.views.accounts.**remove_user_from_org**()
URI: /app/accounts/remove_user_from_org/

Removes a user from an organization.

Payload:

```
    {
        'organization_id': ID of the org,
        'user_id': ID of the user
    }
```

Returns:

```
    {
        'status': 'success' or 'error',
        'message': 'error message, if any'
    }
```

seed.views.accounts.**reset_cleansing_rules**()
URI: /app/accounts/reset_cleansing_rules/

Resets an organization's data cleansing rules

Parameters **request** –

GET Expects organization_id in the query string.

Returns:

```
    {
     'status': 'success',
     'in_range_checking': An array of in-range error rules,
     'missing_matching_field': An array of fields to verify existence,
     'missing_values': An array of fields to ignore missing values
    }
```

seed.views.accounts.**save_cleansing_rules**()
URI: /app/accounts/save_cleansing_rules/

Saves an organization's settings: name, query threshold, shared fields

Payload:

```
    {
        'organization_id: 2,
        'cleansing_rules': {
            'missing_matching_field': [
                {
                    'field': 'address_line_1',
                    'severity': 'error'
                }
            ],
            'missing_values': [
                {
                    'field': 'address_line_1',
                    'severity': 'error'
                }
            ],
            'in_range_checking': [
                {
                    'field': 'conditioned_floor_area',
                    'enabled': true,
                    'type': 'number',
                    'min': null,
                    'max': 7000000,
                    'severity': 'error',
                    'units': 'square feet'
                },
            ]
        }
    }
```

Returns:

```
    {
        'status': 'success',
    }
```

seed.views.accounts.**save_org_settings**()

URI: /app/accounts/save_org_settings/

Saves an organization's settings: name, query threshold, shared fields

Payload:

```
    {
        'organization_id: 2,
        'organization': {
            'query_threshold': 2,
            'name': 'demo org',
            'fields': [  # All internal sibling org shared fields
                {
                    'sort_column': database/search field name,
                        e.g. 'pm_property_id',
                }
            ],
            'public_fields': [  # All publicly shared fields
                {
                    'sort_column': database/search field name,
                        e.g. 'pm_property_id',
                }
            ],
```

```
        }
    }
```

Returns:

```
    {
        'status': 'success or error',
        'message': 'error message, if any'
    }
```

seed.views.accounts.**update_role**()

URI: /app/accounts/update_role/

Sets a user's role within an organization.

Payload:

```
    {
        'organization_id': organization's id,
        'user_id': user's id,
        'role': one of 'owner', 'member', 'viewer'
    }
```

Returns:

```
    {
        'status': 'success or error',
        'message': 'error message, if any'
    }
```

seed.views.accounts.**update_user**()

URI: /app/accounts/update_user/

Updates the request's user's first name, last name, and email

Payload:

```
    {
     'user': {
              'first_name': :first_name,
              'last_name': :last_name,
              'email': :email
             }
    }
```

Returns:

```
    {
        'status': 'success',
        'user': {
            'first_name': user's first name,
            'last_name': user's last name,
            'email': user's email,
            'api_key': user's API key
        }
    }
```

3.6 File Upload Endpoints

These endpoints behave drastically differently depending on whether the system is configured to upload files to S3 or to the local file system.

seed.data_importer.views.**handle_s3_upload_complete**()
URI: /data/s3_upload_complete/

Notify the system that an upload to S3 has been completed. This is a necessary step after uploading to S3 or the SEED instance will not be aware the file exists.

Valid source_type values are found in `seed.models.SEED_DATA_SOURCES`

GET Expects the following in the query string:

key: The full path to the file, within the S3 bucket. E.g. data_importer/buildings.csv

source_type: The source of the file. E.g. 'Assessed Raw' or 'Portfolio Raw'

source_program: Optional value from common.mapper.Programs source_version: e.g. "4.1"

import_record: The ID of the ImportRecord this file belongs to.

Returns:

```
{
    'success': True,
    'import_file_id': The ID of the newly-created ImportFile object.
}
```

seed.data_importer.views.**local_uploader**()
URI: /data/upload/

Endpoint to upload data files to, if uploading to local file storage. Valid source_type values are found in `seed.models.SEED_DATA_SOURCES`

GET The following parameters are expected to be in the query string:

import_record: the ID of the ImportRecord to associate this file with.

qqfile: The name of the file

source_type: A valid source type (e.g. 'Portfolio Raw' or 'Assessed Raw')

Payload:

```
The content of the file as a data stream.  Do not use multipart encoding.
```

Returns:

```
{
    'success': True,
    'import_file_id': The ID of the newly-uploaded ImportFile
}
```

seed.data_importer.views.**get_upload_details**()
URI: /data/get_upload_details/

Retrieves details about how to upload files to this instance.

Returns:

```
    If S3 mode:

    {
        'upload_mode': 'S3',
        'upload_complete': A url to notify that upload is complete,
        'signature': The url to post file details to for auth to upload to S3.
    }

    If local file system mode:

    {
        'upload_mode': 'filesystem',
        'upload_path': The url to POST files to (see local_uploader)
    }
```

seed.data_importer.views.**sign_policy_document**()

URI: /data/sign_policy_document/

Sign and return the policy document for a simple upload. http://aws.amazon.com/articles/1434/#signyours3postform

Payload:

```
    {
     "expiration": ISO-encoded timestamp for when signature should expire,
                   e.g. "2014-07-16T00:20:56.277Z",
     "conditions":
         [
             {"acl":"private"},
             {"bucket": The name of the bucket from get_upload_details},
             {"Content-Type":"text/csv"},
             {"success_action_status":"200"},
             {"key": filename of upload, prefixed with 'data_imports/',
                     suffixed with a unique timestamp.
                     e.g. 'data_imports/my_buildings.csv.1405469756'},
             {"x-amz-meta-category":"data_imports"},
             {"x-amz-meta-qqfilename": original filename}
         ]
    }
```

Returns:

```
    {
        "policy": A hash of the policy document. Using during upload to S3.
        "signature": A signature of the policy document.  Also used during upload to S3.
    }
```

3.7 SEED (Building and Project) Endpoints

seed.views.main.**create_dataset**()

URI: /app/create_dataset/

Creates a new empty dataset (ImportRecord).

Payload:

```
    {
        "name": Name of new dataset, e.g. "2013 city compliance dataset"
```

```
    "organization_id": ID of the org this dataset belongs to
}
```

Returns:

```
{
    'status': 'success',
    'id': The ID of the newly-created ImportRecord,
    'name': The name of the newly-created ImportRecord
}
```

seed.views.main.**create_pm_mapping**()

URI: /app/create_pm_mapping/

Create a mapping for PortfolioManager input columns.

Payload:

```
{
    columns: [ "name1", "name2", ... , "nameN"],
}
```

Returns:

```
{
    success: true,
    mapping: [
        [
            "name1",
            "mapped1", {
                bedes: true|false,
                numeric: true|false
            }
        ],
        [
            "name2",
            "mapped2", {
                bedes: true|false,
                numeric: true|false
            }
        ],
        ...
        [
            "nameN",
            "mappedN", {
                bedes: true|false,
                numeric: true|false
            }
        ]
    ]
}
-- OR --
{
    success: false,
    reason: "message goes here"
}
```

seed.views.main.**delete_buildings**()

URI: /app/delete_buildings/

Deletes all BuildingSnapshots the user has selected.

Does not delete selected_buildings where the user is not a member or owner of the organization the selected building belongs. Since search shows buildings across all the orgs a user belongs, it's possible for a building to belong to an org outside of *org_id*.

DELETE Expects 'org_id' for the organization, and the search payload similar to add_buildings/create_project

Payload:

```
{
    'organization_id': 2,
    'search_payload': {
        'selected_buildings': [2, 3, 4],
        'select_all_checkbox': False,
        'filter_params': ... // see search_buildings
    }
}
```

Returns:

```
{
    'status': 'success' or 'error'
}
```

seed.views.main.**delete_dataset**()

URI: /app/delete_dataset/

Deletes all files from a dataset and the dataset itself.

DELETE Expects organization id and dataset id.

Payload:

```
{
    "dataset_id": 1,
    "organization_id": 1
}
```

Returns:

```
{
    'status': 'success' or 'error',
    'message': 'error message, if any'
}
```

seed.views.main.**delete_duplicates_from_import_file**()

URI: /app/delete_duplicates_from_import_file/

Retrieves the number of matched and unmatched BuildingSnapshots for a given ImportFile record.

GET Expects import_file_id corresponding to the ImportFile in question.

Returns:

```
{
    "status": "success",
    "deleted": "Number of duplicates deleted"
}
```

seed.views.main.**delete_file**()

URI: /app/delete_file/

Deletes an ImportFile from a dataset.

Payload:

```
{
    "file_id": "ImportFile id",
    "organization_id": "current user organization id as integer"
}
```

Returns:

```
{
    'status': 'success' or 'error',
    'message': 'error message, if any'
}
```

seed.views.main.**delete_organization**()
URI: /app/delete_organization/

Starts a background task to delete an organization and all related data.

GET Expects 'org_id' for the organization.

Returns:

```
{
    'status': 'success' or 'error',
    'progress_key': ID of background job, for retrieving job progress
}
```

seed.views.main.**delete_organization_buildings**()
URI: /app/delete_organization_buildings/

Starts a background task to delete all BuildingSnapshots in an org.

GET Expects 'org_id' for the organization.

Returns:

```
{
    'status': 'success' or 'error',
    'progress_key': ID of background job, for retrieving job progress
}
```

seed.views.main.**export_buildings**()
URI: /app/export_buildings/

Begins a building export process.

Payload:

```
{
    "export_name": "My Export",
    "export_type": "csv",
    "selected_buildings": [1234,], (optional list of building ids)
    "selected_fields": optional list of fields to export
    "select_all_checkbox": True // optional, defaults to False
}
```

Returns:

```
{
    "success": True,
    "status": "success",
    "export_id": export_id; see export_buildings_download,
```

```
        "total_buildings": count of buildings,
    }
```

seed.views.main.**export_buildings_download**()
URI: /app/export_buildings/download/

Provides the url to a building export file.

Payload:

```
    {
        "export_id": export_id from export_buildings
    }
```

Returns:

```
    {
        'success': True or False,
        'status': 'success or error',
        'message': 'error message, if any',
        'url': The url to the exported file.
    }
```

seed.views.main.**export_buildings_progress**()
URI: /app/export_buildings/progress/

Returns current progress on building export process.

Payload:

```
    {
        "export_id": export_id from export_buildings
    }
```

Returns:

```
    {
        'success': True,
        'status': 'success or error',
        'message': 'error message, if any',
        'buildings_processed': number of buildings exported
    }
```

seed.views.main.**get_PM_filter_by_counts**()
URI: /app/get_PM_filter_by_counts/

Retrieves the number of matched and unmatched BuildingSnapshots for a given ImportFile record.

GET Expects import_file_id corresponding to the ImportFile in question.

Returns:

```
    {
        'status': 'success',
        'matched': Number of BuildingSnapshot objects that have matches,
        'unmatched': Number of BuildingSnapshot objects with no matches.
    }
```

seed.views.main.**get_aggregated_building_report_data**()
URI: /app/get_aggregated_building_report_data/

This method returns a set of aggregated building data for graphing. It expects as parameters

GET

Parameters

- **start_date** – The starting date for the data series with the format *YYYY-MM-DDThh:mm:ss+hhmm*
- **end_date** – The starting date for the data series with the format *YYYY-MM-DDThh:mm:ss+hhmm*
- **x_var** – The variable name to be assigned to the "x" value in the returned data series
- **y_var** – The variable name to be assigned to the "y" value in the returned data series
- **organization_id** – The organization to be used when querying data.

The x_var values should be from the following set of variable names:

•site_eui

•source_eui

•site_eui_weather_normalized

•source_eui_weather_normalized

•energy_score

The y_var values should be from the following set of variable names:

•gross_floor_area

•use_description

•year_built

This method includes building record count information as part of the result JSON in a property called "building_counts."

This property provides data on the total number of buildings available in each 'year ending' group, as well as the subset of those buildings that have actual data to graph. By sending these values in the result we allow the client to easily build a message like "200 of 250 buildings in this group have data."

Returns:

```
{
    "status": "success",
    "chart_data": [
        {
            "yr_e": x - group by year ending
            "x": x, - median value in group
            "y": y - average value thing
        },
        {
            "yr_e": x
            "x": x,
            "y": y
        }
        ...
    ],
    "building_counts": [
        {
            "yr_e": string for year ending - group by
            "num_buildings": number of buildings in query results
            "num_buildings_w_data": number of buildings with valid data in this group, BOTH x an
```

```
            },
            ...
        ]
        "num_buildings": total number of buildings in query results,
        "num_buildings_w_data": total number of buildings with valid data in query results
    }
```

—

parameters:

- name: x_var description: Name of column in building snapshot database to be used for "x" axis required: true type: string paramType: query
- name: y_var description: Name of column in building snapshot database to be used for "y" axis required: true type: string paramType: query
- start_date: description: The start date for the entire dataset. required: true type: string paramType: query
- end_date: description: The end date for the entire dataset. required: true type: string paramType: query
- name: organization_id description: User's organization which should be used to filter building query results required: true type: string paramType: query

type:

status: required: true type: string

chart_data: required: true type: array

building_counts: required: true type: array

num_buildings: required: true type: string

num_buildings_w_data: required: true type: string

status code:

- code: 400 message: Bad request, only GET method is available
- code: 401 message: Not authenticated
- code: 403 message: Insufficient rights to call this procedure

seed.views.main.**get_building**()

URI: /app/get_building/

Retrieves a building. If user doesn't belong to the building's org, fields will be masked to only those shared within the parent org's structure.

GET Expects building_id and organization_id in query string. building_id should be the *canonical_building* ID for the building, not the BuildingSnapshot id.

Returns:

```
    {
         'status': 'success or error',
         'message': 'error message, if any',
         'building': {'id': the building's id,
                      'canonical_building': the canonical building ID,
                      other fields this user has access to...
         },
         'imported_buildings': [ A list of buildings imported to create
```

```
                                this building's record, in the same
                                format as 'building'
                            ],
        'projects': [
           // A list of the building's projects
           {
               "building": {
                   "approved_date":07/30/2014,
                   "compliant": null,
                   "approver": "demo@seed.lbl.gov"
                   "approved_date": "07/30/2014"
                   "compliant": null
                   "label": {
                       "color": "red",
                       "name": "non compliant",
                       id: 1
                   }
               }
               "description": null
               "id": 3
               "is_compliance": false
               "last_modified_by_id": 1
               "name": "project 1"
               "owner_id": 1
               "slug": "project-1"
               "status": 1
               "super_organization_id": 1
           },
           . . .
       ],
        'user_role': role of user in this org,
        'user_org_id': the org id this user belongs to
    }
```

seed.views.main.**get_building_report_data**()

URI: /app/get_building_report_data/

This method returns a set of x,y building data for graphing. It expects as parameters

GET

Parameters

- **start_date** – The starting date for the data series with the format *YYYY-MM-DD*
- **end_date** – The starting date for the data series with the format *YYYY-MM-DD*
- **x_var** – The variable name to be assigned to the "x" value in the returned data series # NOQA
- **y_var** – The variable name to be assigned to the "y" value in the returned data series # NOQA
- **organization_id** – The organization to be used when querying data.

The x_var values should be from the following set of variable names:

•site_eui

•source_eui

•site_eui_weather_normalized

•source_eui_weather_normalized

•energy_score

The y_var values should be from the following set of variable names:

•gross_floor_area

•use_description

•year_built

This method includes building record count information as part of the result JSON in a property called "building_counts."

This property provides data on the total number of buildings available in each 'year ending' group, as well as the subset of those buildings that have actual data to graph. By sending these values in the result we allow the client to easily build a message like "200 of 250 buildings in this group have data."

Returns:

```
{
    "status": "success",
    "chart_data": [
        {
            "id" the id of the building,
            "yr_e": the year ending value for this data point
            "x": value for x var,
            "y": value for y var,
        },
        ...
    ],
    "building_counts": [
        {
            "yr_e": string for year ending
            "num_buildings": number of buildings in query results
            "num_buildings_w_data": number of buildings with valid data in query results
        },
        ...
    ]
    "num_buildings": total number of buildings in query results,
    "num_buildings_w_data": total number of buildings with valid data in the query results  # NC
}
```

—

parameters:

- name: x_var description: Name of column in building snapshot database to be used for "x" axis required: true type: string paramType: query
- name: y_var description: Name of column in building snapshot database to be used for "y" axis required: true type: string paramType: query
- start_date: description: The start date for the entire dataset. required: true type: string paramType: query
- end_date: description: The end date for the entire dataset. required: true type: string paramType: query
- name: organization_id description: User's organization which should be used to filter building query results required: true type: string paramType: query

- name: aggregate description: Aggregates data based on internal rules (given x and y var) required: true type: string paramType: query

type:

status: required: true type: string

chart_data: required: true type: array

num_buildings: required: true type: string

num_buildings_w_data: required: true type: string

status codes:

- code: 400 message: Bad request, only GET method is available
- code: 401 message: Not authenticated
- code: 403 message: Insufficient rights to call this procedure

seed.views.main.**get_building_summary_report_data**()

URI: /app/get_building_summary_report_data/

This method returns basic, high-level data about a set of buildings, filtered by organization ID.

It expects as parameters

GET

Parameters

- **start_date** – The starting date for the data series with the format *YYYY-MM-DD*
- **end_date** – The starting date for the data series with the format *YYYY-MM-DD*

Returns:

```
{
    "status": "success",
    "summary_data":
    {
        "num_buildings": "number of buildings returned from query",
        "avg_eui": "average EUI for returned buildings",
        "avg_energy_score": "average energy score for returned buildings"
    }
}
```

Units for return values are as follows:

property	units
avg_eui	kBtu/ft2/yr

—

parameters:

- name: organization_id description: User's organization which should be used to filter building query results required: true type: string paramType: query
- start_date: description: The start date for the entire dataset. required: true type: string paramType: query
- end_date: description: The end date for the entire dataset. required: true type: string paramType: query

type:

status: required: true type: string

summary_data: required: true type: object

status codes:

- code: 400 message: Bad request, only GET method is available
- code: 401 message: Not authenticated
- code: 403 message: Insufficient rights to call this procedure

seed.views.main.**get_column_mapping_suggestions**()

URI: /app/get_column_mapping_suggestions/

Returns suggested mappings from an uploaded file's headers to known data fields.

Payload:

```
{
    'import_file_id': The ID of the ImportRecord to examine,
    'org_id': The ID of the user's organization
}
```

Returns:

```
{
    'status': 'success',
    'suggested_column_mappings': {
        column header from file: [ (destination_column, score) ...]
        ...
    },
    'building_columns': [ a list of all possible columns ],
    'building_column_types': [a list of column types corresponding to building_columns],
}
```

..todo: The response of this method may not be correct. verify.

seed.views.main.**get_coparents**()

URI: /app/get_coparents/

Returns the nodes in the BuildingSnapshot tree that can be unmatched.

GET Expects organization_id and building_id in the query string

Returns:

```
{
    'status': 'success',
    'coparents': [
        {
            "id": 333,
            "coparent": 223,
            "child": 443,
            "parents": [],
            "canonical_building_id": 1123
        },
        {
            "id": 223,
            "coparent": 333,
            "child": 443,
            "parents": [],
            "canonical_building_id": 1124
        },
```

```
            ...
        ]
    }
```

seed.views.main.**get_dataset**()

URI: /app/get_dataset/

Retrieves ImportFile objects for one ImportRecord.

GET Expects dataset_id for an ImportRecord in the query string.

Returns:

```
    {
        'status': 'success',
        'dataset': {
            'name': Name of ImportRecord,
            'number_of_buildings': Total number of buildings in all ImportFiles for this dataset,
            'id': ID of ImportRecord,
            'updated_at': Timestamp of when ImportRecord was last modified,
            'last_modified_by': Email address of user making last change,
            'importfiles': [
                {
                   'name': Name of associated ImportFile, e.g. 'buildings.csv',
                   'number_of_buildings': Count of buildings in this file,
                   'number_of_mappings': Number of mapped headers to fields,
                   'number_of_cleanings': Number of fields cleaned,
                   'source_type': Type of file (see source_types),
                   'id': ID of ImportFile (needed for most operations)
                }
             ],
             ...
        },
            ...
    }
```

seed.views.main.**get_datasets**()

URI: /app/get_datasets/

Retrieves all datasets for the user's organization.

GET Expects 'organization_id' of org to retrieve datasets from in query string.

Returns:

```
    {
        'status': 'success',
        'datasets':  [
            {
                'name': Name of ImportRecord,
                'number_of_buildings': Total number of buildings in all ImportFiles,
                'id': ID of ImportRecord,
                'updated_at': Timestamp of when ImportRecord was last modified,
                'last_modified_by': Email address of user making last change,
                'importfiles': [
                    {
                        'name': Name of associated ImportFile, e.g. 'buildings.csv',
                        'number_of_buildings': Count of buildings in this file,
                        'number_of_mappings': Number of mapped headers to fields,
                        'number_of_cleanings': Number of fields cleaned,
                        'source_type': Type of file (see source_types),
```

```
                        'id': ID of ImportFile (needed for most operations)
                    }
                ],
                ...
            },
            ...
        ]
    }
```

seed.views.main.**get_datasets_count**()
URI: /app/get_datasets_count/

Retrieves the number of datasets for an org.

GET Expects organization_id in the query string.

Returns:

```
    {
        'status': 'success',
        'datasets_count': Number of datasets belonging to this org.
    }
```

seed.views.main.**get_first_five_rows**()
URI: /app/get_first_five_rows/

Retrieves the first five rows of an ImportFile.

Payload:

```
    {
        'import_file_id': The ID of the ImportFile
    }
```

Returns:

```
    {
        'status': 'success',
        'first_five_rows': [
            [list of strings of header row],
            [list of strings of first data row],
            ...
            [list of strings of fifth data row]
        ]
    }
```

seed.views.main.**get_import_file**()
URI: /app/get_import_file/

Retrieves details about an ImportFile.

GET Expects import_file_id in the query string.

Returns:

```
    {
        'status': 'success',
        'import_file': {
            "name": Name of the uploaded file,
            "number_of_buildings": number of buildings in the file,
            "number_of_mappings": number of mapped columns,
            "number_of_cleanings": number of cleaned fields,
```

```
            "source_type": type of data in file, e.g. 'Assessed Raw'
            "number_of_matchings": Number of matched buildings in file,
            "id": ImportFile ID,
            'dataset': {
                'name': Name of ImportRecord file belongs to,
                'id': ID of ImportRecord file belongs to,
                'importfiles': [  # All ImportFiles in this ImportRecord, with
                    # requested ImportFile first:
                    {'name': Name of file,
                     'id': ID of ImportFile
                    }
                    ...
                ]
            }
        }
    }
```

seed.views.main.**get_match_tree**()

URI: /app/get_match_tree/

returns the BuildingSnapshot tree

GET Expects organization_id and building_id in the query string

Returns:

```
    {
        'status': 'success',
        'match_tree': [ // array of all the members of the tree
            {
                "id": 333,
                "coparent": 223,
                "child": 443,
                "parents": [],
                "canonical_building_id": 1123
            },
            {
                "id": 223,
                "coparent": 333,
                "child": 443,
                "parents": [],
                "canonical_building_id": 1124
            },
            {
                "id": 443,
                "coparent": null,
                "child": 9933,
                "parents": [333, 223],
                "canonical_building_id": 1123
            },
            {
                "id": 9933,
                "coparent": null,
                "child": null,
                "parents": [443],
                "canonical_building_id": 1123
            },
            ...
        ]
    }
```

```
```

seed.views.main.**get_raw_column_names**()
URI: /app/get_raw_column_names/

Retrieves a list of all column names from an ImportFile.

Payload:

```
{
    'import_file_id': The ID of the ImportFile
}
```

Returns:

```
{
    'status': 'success',
    'raw_columns': [
        list of strings of the header row of the ImportFile
    ]
}
```

seed.views.main.**progress**()
URI: /app/progress/

Get the progress (percent complete) for a task.

Payload:

```
{
    'progress_key': The progress key from starting a background task
}
```

Returns:

```
{
    'progress_key': The same progress key,
    'progress': Percent completion
}
```

seed.views.main.**remap_buildings**()
URI: /app/remap_buildings/

Re-run the background task to remap buildings as if it hadn't happened at all. Deletes mapped buildings for a given ImportRecord, resets status.

NB: will not work if buildings have been merged into CanonicalBuilings.

Payload:

```
{
    'file_id': The ID of the ImportFile to be remapped
}
```

Returns:

```
{
    'status': 'success' or 'error',
    'progress_key': ID of background job, for retrieving job progress
}
```

seed.views.main.**save_column_mappings**()
URI: /app/save_column_mappings/

Saves the mappings between the raw headers of an ImportFile and the destination fields in the BuildingSnapshot model.

Valid source_type values are found in `seed.models.SEED_DATA_SOURCES`

Payload:

```
{
    "import_file_id": ID of the ImportFile record,
    "mappings": [
        ["destination_field": "raw_field"],  #direct mapping
        ["destination_field2":
            ["raw_field1", "raw_field2"],  #concatenated mapping
        ...
    ]
}
```

Returns:

```
{'status': 'success'}
```

`seed.views.main.`**`save_match`**`()`

URI: /app/save_match/

Adds or removes a match between two BuildingSnapshots. Creating a match creates a new BuildingSnapshot with merged data.

Payload:

```
{
    'organization_id': current user organization id,
    'source_building_id': ID of first BuildingSnapshot,
    'target_building_id': ID of second BuildingSnapshot,
    'create_match': True to create match, False to remove it,
    'organization_id': ID of user's organization
}
```

Returns:

```
{
    'status': 'success',
    'child_id': The ID of the newly-created BuildingSnapshot
                containing merged data from the two parents.
}
```

`seed.views.main.`**`save_raw_data`**`()`

URI: /app/save_raw_data/

Starts a background task to import raw data from an ImportFile into BuildingSnapshot objects.

Payload:

```
{
    'file_id': The ID of the ImportFile to be saved
}
```

Returns:

```
{
    'status': 'success' or 'error',
    'progress_key': ID of background job, for retrieving job progress
}
```

seed.views.main.**search_building_snapshots**()

URI: /app/search_building_snapshots/

Retrieves a paginated list of BuildingSnapshots matching search params.

Payload:

```
{
    'q': a string to search on (optional),
    'order_by': which field to order by (e.g. pm_property_id),
    'import_file_id': ID of an import to limit search to,
    'filter_params': {
        a hash of Django-like filter parameters to limit query.  See seed.search.filter_other_pa
    }
    'page': Which page of results to retrieve (default: 1),
    'number_per_page': Number of buildings to retrieve per page (default: 10),
}
```

Returns:

```
{
    'status': 'success',
    'buildings': [
        {
            'pm_property_id': ID of building (from Portfolio Manager),
            'address_line_1': First line of building's address,
            'property_name': Building's name, if any
        }...
    ]
    'number_matching_search': Total number of buildings matching search,
    'number_returned': Number of buildings returned for this page
}
```

seed.views.main.**search_buildings**()

URI: /app/search_buildings/

Retrieves a paginated list of CanonicalBuildings matching search params.

Payload:

```
{
    'q': a string to search on (optional),
    'show_shared_buildings': True to include buildings from other orgs in this user's org tree,
    'order_by': which field to order by (e.g. pm_property_id),
    'import_file_id': ID of an import to limit search to,
    'filter_params': {
        a hash of Django-like filter parameters to limit query.  See seed.search.filter_other_pa
        If 'project__slug' is included and set to a project's slug, buildings will include assoc
        for that project.
    }
    'page': Which page of results to retrieve (default: 1),
    'number_per_page': Number of buildings to retrieve per page (default: 10),
}
```

Returns:

```
{
    'status': 'success',
    'buildings': [
        {
            all fields for buildings the request user has access to, e.g.:
```

```
                    'canonical_building': the CanonicalBuilding ID of the building,
                    'pm_property_id': ID of building (from Portfolio Manager),
                    'address_line_1': First line of building's address,
                    'property_name': Building's name, if any
                ...
            }...
        ]
        'number_matching_search': Total number of buildings matching search,
        'number_returned': Number of buildings returned for this page
    }
```

seed.views.main.**start_mapping**()

URI: /app/start_mapping/

Starts a background task to convert imported raw data into BuildingSnapshots, using user's column mappings.

Payload:

```
    {
        'file_id': The ID of the ImportFile to be mapped
    }
```

Returns:

```
    {
        'status': 'success' or 'error',
        'progress_key': ID of background job, for retrieving job progress
    }
```

seed.views.main.**start_system_matching**()

URI: /app/start_system_matching/

Starts a background task to attempt automatic matching between buildings in an ImportFile with other existing buildings within the same org.

Payload:

```
    {
        'file_id': The ID of the ImportFile to be matched
    }
```

Returns:

```
    {
        'status': 'success' or 'error',
        'progress_key': ID of background job, for retrieving job progress
    }
```

seed.views.main.**update_building**()

URI: /app/update_building/

Manually updates a building's record. Creates a new BuildingSnapshot for the resulting changes.

PUT

Payload:

```
    {
        "organization_id": "organization id as integer",
        "building":
            {
                "canonical_building": "canonical building ID as integer"
```

```
                "fieldname": "value",
                "...": "Remaining fields in the BuildingSnapshot; see get_columns() endpoint for com
            }
    }
```

Returns:

```
    {
        "status": "success",
        "child_id": "The ID of the newly-created BuildingSnapshot"
    }
```

seed.views.main.**update_dataset**()

URI: /app/update_dataset/

Updates the name of a dataset.

Payload:

```
    {
        'dataset': {
            'id': The ID of the Import Record,
            'name': The new name for the ImportRecord
        }
    }
```

Returns:

```
    {
        'status': 'success' or 'error',
        'message': 'error message, if any'
    }
```

seed.views.main.**version**()

URI: /app/version/

Returns the SEED version and current git sha

CHAPTER 4

Data Model

Our primary data model is based on a tree structure with BuildingSnapshot instances as nodes of the tree and the tip of the tree referenced by a CanonicalBuilding.

Take the following example: a user has loaded a CSV file containing information about one building and created the first BuildingSnapshot (BS0). At this point in time, BS0 is linked to the first CanonicalBuilding (CB0), and CB0 is also linked to BS0.

```
BS0 <-- CB0
BS0 --> CB0
```

These relations are represented in the database as foreign keys from the BuildingSnapshot table to the CanonicalBuilding table, and from the CanonicalBuilding table to the BuildingSnapshot table.

The tree structure comes to fruition when a building, BS0 in our case, is matched with a new building, say BS1, enters the system and is auto-matched.

Here BS1 entered the system and was matched with BS0. When a match occurs, a new BuildingSnapshot is created, BS2, with the fields from the existing BuildingSnapshot, BS0, and the new BuildingSnapshot, BS1, merged together. If both the existing and new BuildingSnapshot have data for a given field, the new record's fields are preferred and merged into the child, B3.

The fields from new snapshot are preferred because that is the newer of the two records from the perspective of the system. By preferring the most recent fields this allows for evolving building snapshots over time. For example, if an existing canonical record has a Site EUI value of 75 and some changes happen to a building that cause this to change to 80 the user can submit a new record with that change.

All BuildingSnapshot instances point to a CanonicalBuilding.

```
BS0  BS1
  \ /
  BS2 <-- CB0

BS0 --> CB0
BS1 --> CB0
BS2 --> CB0
```

4.1 parents and children

BuildingSnapshots also have linkage to other BuildingSnapshots in order to keep track of their *parents* and *children*. This is represented in the Django model as a many-to-many relation from BuildingSnapshot to BuildingSnapshot. It is represented in the PostgreSQL database as an additional seed_buildingsnapshot_children table.

In our case here, BS0 and BS1 would both have *children* BS2, and BS2 would have *parents* BS0 and BS1.

Note: throughout most of the application, the `search_buildings` endpoint is used to search or list active building. This is to say, buildings that are pointed to by an active CanonicalBuilding. The `search_building_snapshots` endpoint allows the search of buildings regardless of whether the BuildingSnapshot is pointed to by an active CanonicalBuilding or not and this search is needed during the mapping preview and matching sections of the application.

For illustration purposes let's suppose BS2 and a new building BS3 match to form a child BS4.

parent	child
BS0	BS2
BS1	BS2
BS2	BS4
BS3	BS4

And the corresponding tree would look like:

```
BS0  BS1
  \ /
  BS2  BS3
    \  /
     BS4 <-- CB0

BS0 --> CB0
BS1 --> CB0
BS2 --> CB0
BS3 --> CB0
BS4 --> CB0
```

4.1.1 matching

During the auto-matching process, if a *raw* BuildingSnapshot matches an existing BuildingSnapshot instance, then it will point to the existing BuildingSnapshot instance's CanonicalBuilding. In the case where there is no existing BuildingSnapshot to match, a new CanonicalBuilding will be created, as happened to B0 and C0 above.

field	BS0	BS1	BS2 (child)
id1	**11**	11	11
id2		**12**	12
id3	**13**		13
id4	14	**15**	15

4.2 manual-matching vs auto-matching

Since BuildingSnapshots can be manually matched, there is the possibility for two BuildingSnapshots each with an active CanonicalBuilding to match and the system has to choose to move only one CanonicalBuilding to the tip of the tree for the primary BuildingSnapshot and *deactivate* the secondary BuildingSnapshot's CanonicalBuilding.

Take for example:

```
BS0  BS1
  \ /
  BS2  BS3
    \  /
     BS4 <-- CB0 (active: True)          BS5 <-- CB1 (active: True)
```

If a user decides to manually match BS4 and BS5, the system will take the primary BuildingSnapshot's CanonicalBuilding and have it point to their child and deactivate CB1. The deactivation is handled by setting a field on the CanonicalBuilding instance, *active*, from `True` to `False`.

Here is what the tree would look like after the manual match of **BS4** and **BS5**:

```
BS0  BS1
  \ /
  BS2  BS3
    \  /
     BS4  BS5 <-- CB1 (active: False)
       \  /
        BS6 <-- CB0 (active: True)
```

Even though BS5 is pointed to by a CanonicalBuilding, CB1, BS5 will not be returned by the normal `search_buildings` endpoint because the CanonicalBuilding pointing to it has its field `active` set to `False`.

Note: anytime a match is **unmatched** the system will create a new CanonicalBuilding or set an existing CanonicalBuilding's active field to `True` for any leaf BuildingSnapshot trees.

4.3 what really happens to the BuildingSnapshot table on import (and when)

The above is conceptually what happens but sometimes the devil is in the details. Here is what happens to the BuildingSnapshot table in the database when records are imported.

Every time a record is added at least two BuildingSnapshot records are created.

Consider the following simple record:

Property Id	Year Ending	Property Floor Area	Address 1	Release Date
499045	2000	1234	1 fake st	12/12/2000

The first thing the user is upload the file. When the user sees the "Successful Upload!" dialog one record has been added to the BuildingSnapshot table.

This new record has an id (73700 in this case) and a created and modified timestamp. Then there are a lot of empty fields and a source_type of 0. Then there is the extra_data column which contains the contents of the record in key-value form:

Address 1 "1 fake st"

Property Id "499045"

Year Ending "2000"

Release Date "12/12/2000"

Property Floor Area "1234"

And a corresponding extra_data_sources that looks like

Address 1 73700

Property Id 73700

Year Ending 73700

Release Date 73700

Property Floor Area 73700

All of the fields that look like _source_id are also populated with 73700 E.G. owner_postal_code_source_id.

The other fields of interest are the super_organization field which is populated with the user's default organization and the import_file_id field which is populated with a reference to a data_importer_importfile record.

At this point the record has been created before the user hits the "Continue to data mapping" button.

The second record (id = 73701) is created by the time the user gets to the screen with the "Save Mappings" button. This second record has the following fields populated:

- id
- created
- modified
- pm_property_id
- year_ending
- gross_floor_area
- address_line_1
- release_date
- source_type (this is 2 instead of 0 as with the other record)
- import_file_id
- super_organization_id.

That is all. All other fields are empty. In this case that is all that happens.

Now consider the same user uploading a new file from the next year that looks like

Property Id	Year Ending	Property Floor Area	Address 1	Release Date
499045	2000	1234	1 fake st	12/12/2001

As before one new record is created on upload. This has id 73702 and follows the same pattern as 73700. And similarly 73703 is created like 73701 before the "Save Mappings" button appears.

However this time the system was able to make a match with an existing record. After the user clicks the "Confirm mappings & start matching" button a new record is created with ID 73704.

73704 is identical to 73703 (in terms of contents of the BuildingSnapshot table only) with the following exceptions:

- created and modified timestamps are different
- match type is populated and has a value of 1
- confidence is populated and has a value of .9
- source_type is 4 instead of 2
- canonical_building_id is populated with a value
- import_file_id is NULL
- last_modified_by_id is populated with value 2 (This is a key into the landing_seeduser table)
- address_line_1_source_id is 73701
- gross_floor_area_source_id is populated with value 73701
- pm_property_id_source_id is populated with 73701
- release_date_source_id is populated with 73701

- year_ending_source_id is populated with 73701

4.4 what really happens to the CanonicalBuilding table on import (and when)

In addition to the BuildingSnapshot table the CanonicalBuilding table is also updated during the import process. To summarize the above 5 records were created in the BuildingSnapshot table:

1. 73700 is created from the raw 2000 data
2. 73701 is the mapped 2000 data,
3. 73702 is created from the raw 2001 data
4. 73703 is the mapped 2001 data
5. 73704 is the result of merging the 2000 and 2001 data.

In this process CanonicalBuilding is updated twice. First when the 2000 record is imported the CanonicalBuilding gets populated with one new row at the end of the matching step. I.E. when the user sees the "Load More Data" screen. At this point there is a new row that looks like

id	active	canonical_building_id
20505	TRUE	73701

At this point there is one new canonical building and that is the BuildingSnapshot with id 73701. Next the user uploads the 2001 data. When the "Matching Results" screen appears the CanonicalBuilding table has been updated. Now it looks like

id	active	canonical_building_id
20505	TRUE	73704

There is still only one canonical building but now it is the BuildingSnapshot record that is the result of merging the 2000 and 2001 data: id = 73704.

4.5 organization

BuildingSnapshots belong to an Organization by virtue of a super_organization field that is a foreign key into the organization model (orgs_organization in Postgres). Many endpoints filter the buildings based on the organizations the requesting user belongs to. E.G. get_buildings changes which fields are returned based on the requesting user's membership in the BuildingSnapshot's organization.

4.6 *_source_id fields

Any field in the BuildingSnapshot table that is populated with data from a submitted record will have a corresponding _source_id field. E.G pm_property_id has pm_property_id_source_id, address_line_1 has address_line_1_source_id, etc...

These are foreign keys into the BuildingSnapshot that is the source of that value. To extend the above table

field	BS0	BS1	BS2 (child)	BS2 (child) _source_id
id1	**11**		11	BS0
id2		**12**	12	BS1

NOTE: The BuildingSnapshot records made from the raw input file have all the _source_id fields populated with that record's ID. The non-canonical BuildingSnapshot records created from the mapped data have none set. The canonical BuildingSnapshot records that are the result of merging two records have only the _source_id fields set where the record itself has data. E.G. in the above address_line_1 is set to "1 fake st." so there is a value in the canonical BuildingSnapshot's address_line_1_source_id field. However there is no block number so block_number_source_id is empty. This is unlike the two raw BuildingSnapshot records who also have no block_number but nevertheless have a block_number_source_id populated.

4.7 extra_data

The BuildingSnapshot model has many "named" fields. Fields like "address_line_1", "year_built", and "pm_property_id". However the users are allowed to submit files with arbitrary fields. Some of those arbitrary fields can be mapped to "named" fields. E.G. "Street Address" can usually be mapped to "Address Line 1". For all the fields that cannot be mapped like that there is the extra_data field.

extra_data is Django json field that serves as key-value storage for other user-submitted fields. As with the other "named" fields there is a corresponding extra_data_sources field that serves the same role as the other _source_id fields. E.G. If a BuildingSnapshot has an extra_data field that looks like

an_unknown_field 1

something_else 2

It should have an extra_data_sources field that looks like

an_unknown_field some_BuildingSnapshot_id

something_else another_BuildingSnapshot_id

4.8 saving and possible data loss

When saving a BuildingSnapshot file some fields that are truncated if too long. The following are truncated to 128 characters

- tax_lot_id
- pm_property_id
- custom_id_1
- lot_number
- block_number
- district
- owner
- owner_email
- owner_telephone
- owner_address
- owner_city_state
- owner_postal_code

And the following are truncated to 255:

- property_name

- address_line_1
- address_line_2
- city
- postal_code
- state_province
- building_certification

No truncation happens to any of the fields stored in extra_data.

CHAPTER 5

Mapping

This document describes the set of calls that occur from the web client or API down to the back-end for the process of mapping.

An overview of the process is:

1. Import - A file is uploaded and saved in the database
2. Mapping - Mapping occurs on that file

5.1 Import

From the web UI, the import process invokes *seed.views.main.save_raw_data* to save the data. When the data is done uploading, we need to know whether it is a Portfolio Manager file, so we can add metadata to the record in the database. The end of the upload happens in *seed.data_importer.views.DataImportBackend.upload_complete* or *seed.data_importer.views.handle_s3_upload_complete*, depending on whether it is using a local file system or Amazon S3-based backend. At this point, the request object has additional attributes for Portfolio Manager files. These are saved in the model *seed.data_importer.models.ImportFile*.

5.2 Mapping

After the data is saved, the UI invokes *seed.views.main.get_column_mapping_suggestions* to get the columns to display on the mapping screen. This loads back the model that was mentioned above as an *ImportFile* instance, and then the *from_portfolio_manager* property can be used to choose the branch of the code:

If it is a Portfolio Manager file the *seed.common.mapper.get_pm_mapping* method provides a high-level interface to the Portfolio Manager mapping (see comments in the containing file, *mapper.py*), and the result is used to populate the return value for this method, which goes back to the UI to display the mapping screen.

Otherwise the code does some auto-magical logic to try and infer the "correct" mapping.

CHAPTER 6

Modules

6.1 Audit Logs Package

6.1.1 Submodules

6.1.2 Models

class seed.audit_logs.models.**AuditLog**(**args*, ***kwargs*)

Bases: django_extensions.db.models.TimeStampedModel

An audit log of events and notes. Inherits created and modified from TimeStampedModel

exception DoesNotExist

Bases: django.core.exceptions.ObjectDoesNotExist

exception AuditLog.**MultipleObjectsReturned**

Bases: django.core.exceptions.MultipleObjectsReturned

AuditLog.**action_response**

A placeholder class that provides a way to set the attribute on the model.

AuditLog.**content_object**

Provide a generic many-to-one relation through the content_type and object_id fields.

This class also doubles as an accessor to the related object (similar to ForwardManyToOneDescriptor) by adding itself as a model attribute.

AuditLog.**content_type**

Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

child.parent is a ForwardManyToOneDescriptor instance.

AuditLog.**get_audit_type_display**(**moreargs*, ***morekwargs*)

AuditLog.**get_next_by_created**(**moreargs*, ***morekwargs*)

AuditLog.**get_next_by_modified**(**moreargs*, ***morekwargs*)

AuditLog.**get_previous_by_created**(**moreargs*, ***morekwargs*)

AuditLog.**get_previous_by_modified**(**moreargs*, ***morekwargs*)

AuditLog.**objects** = <seed.audit_logs.models.AuditLogManager object>

AuditLog.**organization**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

AuditLog.**save**(**args*, ***kwargs*)
Ensure that only notes are saved

AuditLog.**to_dict**()
serializes an audit_log

AuditLog.**user**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

class seed.audit_logs.models.**AuditLogManager**
Bases: django.db.models.manager.Manager

ExpressionManager with `update` preventing the update of non-notes

get_queryset()

log_action(*request*, *conent_object*, *organization_id*, *action_response=None*, *action_note=None*, *audit_type=0*)

use_for_related_fields = True

class seed.audit_logs.models.**AuditLogQuerySet**(*model=None*, *query=None*, *using=None*, *hints=None*)
Bases: django.db.models.query.QuerySet

update(**args*, ***kwargs*)
only notes should be updated, so filter out non-notes

6.1.3 Tests

class seed.audit_logs.tests.**AuditLogModelTests**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

setUp()

test_audit()
tests audit save

test_audit_save()
audit_log `LOG` should not be able to save/update

test_audit_update()
audit_log `LOG` should not be able to save/update

test_generic_relation()
test CanonicalBuilding.audit_logs

test_get_all_audit_logs_for_an_org()
gets all audit logs for an org

test_model___unicode__()
tests the AuditLog inst. str or unicode

test_note()
tests note save

test_note_save()
notes should be able to save/update

class seed.audit_logs.tests.**AuditLogViewTests**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

setUp()

test_create_note()
tests create_note

test_get_building_logs()
test the django view get_building_logs

test_update_note()
tests update_note

6.1.4 URLs

6.1.5 Views

seed.audit_logs.views.**create_note**(*request*, **args*, ***kwargs*)
Retrieves logs for a building.

POST Expects the CanonicalBuildings's id in the JSON payload as building_id. Expects an organization_id (to which project belongs) in the query string. Expects the action_note to be in the JSON payload as action_note

Returns:

```
'audit_log' : {
    'user': {
        'first_name': user's firstname,
        'last_name': user's last_name,
        'id': user's id,
        'email': user's email address
    },
    'id': audit log's id,
    'audit_type': 'Note',
    'created': DateTime,
    'modified': DateTime,
    'action': method triggering log entry,
    'action_response': response of action,
    'action_note': the note body
    'organization': {
        'name': name of org,
        'id': id of org
```

```
    },
    'status': 'success'
```

seed.audit_logs.views.**get_building_logs**(*request*, **args*, ***kwargs*)

Retrieves logs for a building.

GET Expects the CanonicalBuildings's id in the query string as building_id. Expects an organization_id (to which project belongs) in the query string.

Returns:

```
    'audit_logs' : [
        {
            'user': {
                'first_name': user's firstname,
                'last_name': user's last_name,
                'id': user's id,
                'email': user's email address
            },
            'id': audit log's id,
            'audit_type': 'Log' or 'Note',
            'created': DateTime,
            'modified': DateTime,
            'action': method triggering log entry,
            'action_response': response of action,
            'action_note': the note body if Note or further description
            'organization': {
                'name': name of org,
                'id': id of org
            }
        }, ...
    ],
    'status': 'success'
```

seed.audit_logs.views.**update_note**(*request*, **args*, ***kwargs*)

Retrieves logs for a building.

PUT Expects the CanonicalBuildings's id in the JSON payload as building_id. Expects an organization_id (to which project belongs) in the query string. Expects the action_note to be in the JSON payload as action_note Expects the audit_log_id to be in the JSON payload as audit_log_id

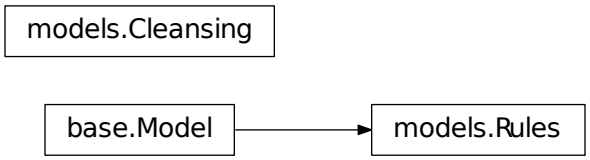
Returns:

```
    'audit_log' : {
        'user': {
            'first_name': user's firstname,
            'last_name': user's last_name,
            'id': user's id,
            'email': user's email address
        },
        'id': audit log's id,
        'audit_type': 'Note',
        'created': DateTime,
        'modified': DateTime,
        'action': method triggering log entry,
        'action_response': response of action,
        'action_note': the note body
        'organization': {
            'name': name of org,
            'id': id of org
```

```
    },
    'status': 'success'
```

6.2 Cleansing Package

6.2.1 Inheritance



6.2.2 Submodules

6.2.3 Models

class seed.cleansing.models.**Cleansing**(*organization*, **args*, ***kwargs*)

Bases: object

ASSESSOR_FIELDS = [{'sort_column': 'pm_property_id', 'title': 'PM Property ID'}, {'sort_column': 'tax_lot_id', 'title'

ASSESSOR_FIELDS_BY_COLUMN = {'lot_number': {'sort_column': 'lot_number', 'title': 'Lot Number'}, 'owner_addre

static cache_key(*file_pk*)

Static method to return the location of the cleansing results from redis.

Parameters **file_pk** – Import file primary key

Returns

cleanse(*data*)

Send in data as a queryset from the BuildingSnapshot ids.

Parameters **data** – rows of data to be cleansed

Returns

data_type_check(*datum*)

Check the data types of the fields. These should never be wrong as these are the data in the database.

This chunk of code is currently ignored.

Parameters **datum** – Database record containing the BS version of the fields populated

Returns None

in_range_checking(*datum*)

Check for errors in the min/max of the values.

Parameters **datum** – Database record containing the BS version of the fields populated

Returns None

static **initialize_cache**(*file_pk*)
Initialize the cache for storing the results. This is called before the celery tasks are chunked up.

Parameters **file_pk** – Import file primary key

Returns

missing_matching_field(*datum*)
Look for fields in the database that are not matched. Missing is defined as a None in the database

Parameters **datum** – Database record containing the BS version of the fields populated

Returns None

TODO: NL: Should we check the extra_data field for the data?

missing_values(*datum*)
Look for fields in the database that are empty. Need to know the list of fields that are part of the cleansing section.

The original intent of this method would be very intensive to run (looking at all fields except the ignored). This method was changed to check for required values.

Parameters **datum** – Database record containing the BS version of the fields populated

Returns None

TODO: Check the extra_data field for the data?

prune_data()
Prune the results will remove any entries that have zero cleansing_results

Returns None

reset_results()

save_to_cache(*file_pk*)
Save the results to the cache database. The data in the cache are stored as a list of dictionaries. The data in this class are stored as a dict of dict. This is important to remember because the data from the cache cannot be simply loaded into the above structure.

Parameters **file_pk** – Import file primary key

Returns None

class seed.cleansing.models.**Rules**(*id*, *org*, *field*, *enabled*, *category*, *type*, *min*, *max*, *severity*, *units*)
Bases: django.db.models.base.Model

exception **DoesNotExist**
Bases: django.core.exceptions.ObjectDoesNotExist

exception Rules.**MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

static Rules.**delete_rules**(*organization*)

Rules.**get_category_display**(**moreargs*, ***morekwargs*)

Rules.**get_severity_display**(**moreargs*, ***morekwargs*)

Rules.**get_type_display**(**moreargs*, ***morekwargs*)

static Rules.**initialize_rules**(*organization*)

Rules.**objects = <django.db.models.manager.Manager object>**

Rules.**org**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

static Rules.**restore_defaults**(*organization*)

6.2.4 Tasks

6.2.5 Tests

class seed.cleansing.tests.**CleansingDataSample**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

setUp()

test_cleanse()

class seed.cleansing.tests.**CleansingDataTestCoveredBuilding**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

setUp()

test_cleanse()

test_simple_login()

class seed.cleansing.tests.**CleansingDataTestPM**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

setUp()

test_cleanse()

class seed.cleansing.tests.**CleansingViewTests**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

setUp()

test_get_cleansing_results()

test_get_csv()

test_get_progress()

6.2.6 URLs

6.2.7 Views

seed.cleansing.views.**get_cleansing_results**(*request*, **args*, ***kwargs*)
Retrieve the details of the cleansing script.

seed.cleansing.views.**get_csv**(*request*, **args*, ***kwargs*)
Download a csv of the results.

seed.cleansing.views.**get_progress**(*request*, **args*, ***kwargs*)
Return the progress of the cleansing.

6.3 Common Package

6.3.1 Submodules

6.3.2 Mapper

class seed.common.mapper.**MapItem**(*key*, *item*)
Bases: object

Wrapper around a mapped item.

An object will be created with the following attributes:

•source => The source field from which we mapped

•field => The field to which we mapped

•is_bedes => flag, whether this field is BEDES-compliant

•is_numeric => whether the data is numeric (or string)

as_json()

class seed.common.mapper.**Mapping**(*fileobj*, *encoding=None*, *regex=False*, *spc_or_underscore=True*, *ignore_case=True*, *normalize_units=True*)
Bases: object

Mapping from one set of fields to another. The mapping can be many:1. The lookup may be by static string or regular expression.

META_BEDES = 'bedes'

META_NUMERIC = 'numeric'

META_TYPE = 'type'

apply(*keys*)
Get value for a list of keys.

Parameters **keys** – List of keys (strings)

Returns A pair of values. The first is a mapping {key: value} of the keys that matched. The second is a list [key, key, ..] of those that didn't.

get(*key*, *default=None*)
Wrapper around __getitem__ that will return the default instead of raising KeyError if the item is not found.

keys()
Get list of source keys.

Return: (list) Source keys

class seed.common.mapper.**MappingConfiguration**
Bases: object

Factory for creating Mapping objects from configurations.

pm(*version*)
Get Portfolio Manager mapping for given version.

Args: version (tuple): A list of integers/strings (major, minor, ..)

Raises: ValueError, if no mapping is found

class seed.common.mapper.**Programs**
Bases: object

Enumeration of program names.

PM = 'PortfolioManager'

seed.common.mapper.**get_pm_mapping**(*version*, *columns*, *include_none=False*)
Create and return Portfolio Manager (PM) mapping for a given version of PM and the given list of column names.

Args: version (str): Version in format 'x.y[.z]' columns (list): A list of [column_name, field, {metadata}] include_none (bool): If True, add {column:None} for unmatched columns.

Return: (dict) of {column:MapItem}, where *column* is one of the values in the input list. If *include_none* was True, then all columns should be in the output.

6.3.3 Util

seed.common.util.**apply_map**(*map_path*, *data_path*, *out_file*)
Apply a JSON mapping to data, and write the output.

Args: map_path (str): Path to mapping file data_path (str): Path to data file out_file (file): output stream

Return: None

seed.common.util.**create_map**(*path_in*, *path_out*)
Create a JSON mapping file, suitable for *map.Mapping()*, from a CSV input file in our own custom style.

Input columns: CurrentSEED,NewSEED,PM1,PM2,Type (ignore rest)

Parameters

- **path_in** –
- **path_out** –

Returns None

seed.common.util.**find_duplicates**(*map_path*, *data_path*, *out_file*)
Find duplicates created by a given mapping on a given input file.

Args: map_path (str): Path to mapping file data_path (str): Path to data file out_file (file): output stream

Return: None

6.3.4 Views

seed.common.views.**api_error**(*reason*)
JSON response for API error.

seed.common.views.**api_success**(***kwargs*)
JSON response for API success.

seed.common.views.**missing_request_keys**(*keys*, *body*)
Check for *keys* in *body*.

Args: keys (list): List of keys to check body (dict): body of request

Returns: None if all present, JSON error response (using *api_error()*) if one or more is missing.

`seed.common.views.`**`typeof_request_values`**(*types*, *body*)

Check for type of request values.

All keys in types are present in body.

Parameters

- **`types`** (*dict*) – key to type map, where *type* is a function used to test the type conversion. It should take one argument, the value, and raise a ValueError if it is invalid.
- **`body`** (*dict*) – body of request

Returns JSON

Example:

```
    None if all OK, JSON error response (using `api_error()`)
    if one or more is of the wrong type.
```

6.4 Configuration

6.4.1 Submodules

6.4.2 Storage

class `config.storage.`**`CachedS3BotoStorage`**(**args*, ***kwargs*)

Bases: `storages.backends.s3boto.S3BotoStorage`

S3 storage backend that saves the files locally, too.

`save`(*name*, *content*)

6.4.3 Template Context

`config.template_context.`**`sentry_js`**(*request*)

`config.template_context.`**`session_key`**(*request*)

6.4.4 Tests

6.4.5 Utils

`config.utils.`**`de_camel_case`**(*name*)

6.4.6 Views

`config.views.`**`robots_txt`**(*request*, *allow=False*)

6.4.7 WSGI

WSGI config for config project.

This module contains the WSGI application used by Django's development server and any production WSGI deployments. It should expose a module-level variable named `application`. Django's `runserver` and `runfcgi` commands discover this application via the `WSGI_APPLICATION` setting.

Usually you will have the standard Django WSGI application here, but it also might make sense to replace the whole Django WSGI application with a custom one that later delegates to the Django one. For example, you could introduce WSGI middleware here, or combine a Django application with an application of another framework.

6.5 Data Package

6.5.1 Submodules

6.5.2 BEDES

6.5.3 Module contents

6.6 Data Importer Package

6.6.1 Submodules

6.6.2 Managers

class seed.data_importer.managers.**NotDeletedManager**
Bases: django.db.models.manager.Manager

get_queryset(**args*, ***kwargs*)

use_for_related_fields = True

6.6.3 Models

class seed.data_importer.models.**BuildingImportRecord**(*id*, *import_record*, *building_model_content_type*, *building_pk*, *was_in_database*, *is_missing_from_import*)
Bases: django.db.models.base.Model

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception BuildingImportRecord.**MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

BuildingImportRecord.**building_model_content_type**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

child.parent is a ForwardManyToOneDescriptor instance.

BuildingImportRecord.**building_record**
Provide a generic many-to-one relation through the content_type and object_id fields.

This class also doubles as an accessor to the related object (similar to ForwardManyToOneDescriptor) by adding itself as a model attribute.

BuildingImportRecord.**import_record**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

child.parent is a ForwardManyToOneDescriptor instance.

BuildingImportRecord.**objects = <django.db.models.manager.Manager object>**

class seed.data_importer.models.**DataCoercionMapping**(*id*, *table_column_mapping*, *source_string*, *source_type*, *destination_value*, *destination_type*, *is_mapped*, *confidence*, *was_a_human_decision*, *valid_destination_value*, *active*)

Bases: django.db.models.base.Model

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception DataCoercionMapping.**MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

DataCoercionMapping.**objects = <django.db.models.manager.Manager object>**

DataCoercionMapping.**save**(**args*, ***kwargs*)

DataCoercionMapping.**source_string_sha**

DataCoercionMapping.**table_column_mapping**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

child.parent is a ForwardManyToOneDescriptor instance.

class seed.data_importer.models.**ImportFile**(*id*, *created*, *modified*, *deleted*, *import_record*, *file*, *export_file*, *file_size_in_bytes*, *cached_first_row*, *cached_second_to_fifth_row*, *num_columns*, *num_rows*, *num_mapping_warnings*, *num_mapping_errors*, *mapping_error_messages*, *num_validation_errors*, *num_tasks_total*, *num_tasks_complete*, *num_coercion_errors*, *num_coercions_total*, *has_header_row*, *raw_save_done*, *raw_save_completion*, *mapping_done*, *mapping_completion*, *matching_done*, *matching_completion*, *source_type*, *source_program*, *source_program_version*)

Bases: *seed.data_importer.models.NotDeletableModel*, django_extensions.db.models.TimeStampe

CLEANING_ACTIVE_CACHE_KEY

classmethod CLEANING_ACTIVE_CACHE_KEY_GENERATOR(*pk*)

CLEANING_PROGRESS_KEY

CLEANING_QUEUED_CACHE_KEY

classmethod CLEANING_QUEUED_CACHE_KEY_GENERATOR(*pk*)

exception DoesNotExist
Bases: `django.core.exceptions.ObjectDoesNotExist`

ImportFile.**EXPORT_PCT_COMPLETE_CACHE_KEY**

ImportFile.**EXPORT_QUEUED_CACHE_KEY**

ImportFile.**EXPORT_READY_CACHE_KEY**

exception ImportFile.**MultipleObjectsReturned**
Bases: `django.core.exceptions.MultipleObjectsReturned`

ImportFile.**QUEUED_TCM_DATA_KEY**

ImportFile.**QUEUED_TCM_SAVE_COUNTER_KEY**

ImportFile.**SAVE_COUNTER_CACHE_KEY**

ImportFile.**UPDATING_TCMS_KEY**

ImportFile.**buildingsnapshot_set**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

ImportFile.**cache_first_rows**()

ImportFile.**cleaned_data_rows**
Iterable of rows, made of iterable of column values of cleaned data

ImportFile.**cleaning_progress_pct**

ImportFile.**coercion_mapping_active**

ImportFile.**coercion_mapping_queued**

ImportFile.**data_rows**
Iterable of rows, made of iterable of column values of the raw data

ImportFile.**default_manager = <seed.data_importer.managers.NotDeletedManager object>**

ImportFile.**export_generation_pct_complete**

ImportFile.**export_ready**

ImportFile.**export_url**

ImportFile.**filename_only**

ImportFile.**first_row_columns**

ImportFile.**force_restart_cleaning_url**

ImportFile.**from_portfolio_manager**

ImportFile.**generate_url**

ImportFile.**get_next_by_created**(**moreargs*, ***morekwargs*)

ImportFile.**get_next_by_modified**(**moreargs*, ***morekwargs*)

ImportFile.**get_previous_by_created**(**moreargs*, ***morekwargs*)

ImportFile.**get_previous_by_modified**(**moreargs*, ***morekwargs*)

ImportFile.**import_record**

Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

ImportFile.**local_file**

ImportFile.**merge_progress_url**

ImportFile.**num_cells**

ImportFile.**num_cleaning_complete**

ImportFile.**num_cleaning_remaining**

ImportFile.**num_cleaning_total**

ImportFile.**num_failed_tablecolumnmappings**

ImportFile.**num_mapping_complete**

ImportFile.**num_mapping_remaining**

ImportFile.**num_mapping_total**

ImportFile.**objects = <seed.data_importer.managers.NotDeletedManager object>**

ImportFile.**premerge_progress_url**

ImportFile.**raw_objects = <django.db.models.manager.Manager object>**

ImportFile.**ready_to_import**

ImportFile.**save**(*in_validation=False*, **args*, ***kwargs*)

ImportFile.**second_to_fifth_rows**

ImportFile.**tablecolumnmapping_formset**(**args*, ***kwargs*)

ImportFile.**tablecolumnmapping_set**

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

parent.children is a ReverseManyToOneDescriptor instance.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

ImportFile.**tablecolumnmappings**

ImportFile.**tablecolumnmappings_failed**

ImportFile.**tcm_errors_json**

ImportFile.**tcm_fields_to_save**

ImportFile.**tcm_json**

ImportFile.**update_tcms_from_save**(*json_data*, *save_counter*)

class seed.data_importer.models.**ImportRecord**(*id*, *deleted*, *name*, *app*, *owner*, *start_time*, *finish_time*, *created_at*, *updated_at*, *last_modified_by*, *notes*, *merge_analysis_done*, *merge_analysis_active*, *merge_analysis_queued*, *pre-merge_analysis_done*, *pre-merge_analysis_active*, *pre-merge_analysis_queued*, *matching_active*, *matching_done*, *is_imported_live*, *keep_missing_buildings*, *status*, *import_completed_at*, *merge_completed_at*, *mcm_version*, *super_organization*)

Bases: *seed.data_importer.models.NotDeletableModel*

exception DoesNotExist

Bases: django.core.exceptions.ObjectDoesNotExist

ImportRecord.**MAPPING_ACTIVE_KEY**

ImportRecord.**MAPPING_QUEUED_KEY**

exception ImportRecord.**MultipleObjectsReturned**

Bases: django.core.exceptions.MultipleObjectsReturned

classmethod ImportRecord.**SUMMARY_ANALYSIS_ACTIVE_KEY**(*pk*)

classmethod ImportRecord.**SUMMARY_ANALYSIS_QUEUED_KEY**(*pk*)

ImportRecord.**add_files_url**

ImportRecord.**app_namespace**

ImportRecord.**buildingimportrecord_set**

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

parent.children is a ReverseManyToOneDescriptor instance.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

ImportRecord.**dashboard_url**

ImportRecord.**default_manager = <seed.data_importer.managers.NotDeletedManager object>**

ImportRecord.**delete**(**args*, ***kwargs*)

ImportRecord.**delete_url**

ImportRecord.**display_as_in_progress**

ImportRecord.**estimated_seconds_remaining**

ImportRecord.**files**

ImportRecord.**form**

ImportRecord.**get_status_display**(**moreargs*, ***morekwargs*)

ImportRecord.**importfile_set**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

ImportRecord.**is_mapping_or_cleaning**

ImportRecord.**is_not_in_progress**

ImportRecord.**last_modified_by**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

ImportRecord.**mark_merge_started**()
Marks the ImportRecord as having a merge in progress.

ImportRecord.**mark_merged**()
Marks the ImportRecord as having been processed (via merge_import_record())

ImportRecord.**match_progress_key**
Cache key used to track percentage completion for merge task.

ImportRecord.**matched_buildings**

ImportRecord.**merge_progress_key**
Cache key used to track percentage completion for merge task.

ImportRecord.**merge_progress_url**

ImportRecord.**merge_seconds_remaining_key**

ImportRecord.**merge_status**

ImportRecord.**merge_status_key**
Cache key used to set/get status messages for merge task.

ImportRecord.**merge_url**

ImportRecord.**missing_buildings**

ImportRecord.**new_buildings**

ImportRecord.**num_buildings_imported_total**

ImportRecord.**num_coercion_errors**

ImportRecord.**num_columns**

ImportRecord.**num_failed_tablecolumnmappings**

ImportRecord.**num_files**

ImportRecord.**num_files_cleaned**

ImportRecord.**num_files_mapped**

ImportRecord.**num_files_merged**

ImportRecord.**num_files_to_clean**

ImportRecord.**num_files_to_map**

ImportRecord.**num_files_to_merge**

ImportRecord.**num_matched_buildings**

ImportRecord.**num_missing_buildings**

ImportRecord.**num_new_buildings**

ImportRecord.**num_not_ready_for_import**

ImportRecord.**num_ready_for_import**

ImportRecord.**num_rows**

ImportRecord.**num_validation_errors**

ImportRecord.**objects = <seed.data_importer.managers.NotDeletedManager object>**

ImportRecord.**owner**

Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

ImportRecord.**pct_merge_complete**

ImportRecord.**pct_premerge_complete**

ImportRecord.**percent_files_cleaned**

ImportRecord.**percent_files_mapped**

ImportRecord.**percent_files_ready_to_merge**

ImportRecord.**percent_ready_for_import**

ImportRecord.**percent_ready_for_import_by_file_count**

ImportRecord.**pre_merge_url**

ImportRecord.**prefixed_pk**(*pk*, *max_len_before_prefix=32*)

This is a total hack to support prefixing until source_facility_id is turned into a proper pk. Prefixes a given pk with the import_record

ImportRecord.**premerge_estimated_seconds_remaining**

ImportRecord.**premerge_progress_key**

ImportRecord.**premerge_progress_url**

ImportRecord.**premerge_seconds_remaining_key**

ImportRecord.**raw_objects = <django.db.models.manager.Manager object>**

ImportRecord.**ready_for_import**

ImportRecord.**save_import_meta_url**

ImportRecord.**search_url**

ImportRecord.**start_merge_url**

ImportRecord.**status_denominator**

ImportRecord.**status_is_live**

ImportRecord.**status_numerator**

ImportRecord.**status_percent**

ImportRecord.**status_url**

ImportRecord.**summary_analysis_active**

ImportRecord.**summary_analysis_queued**

ImportRecord.**super_organization**

Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

ImportRecord.**to_json**

ImportRecord.**total_correct_mappings**

ImportRecord.**total_file_size**

ImportRecord.**worksheet_progress_json**

ImportRecord.**worksheet_url**

class seed.data_importer.models.**NotDeletableModel**(**args*, ***kwargs*)

Bases: django.db.models.base.Model

class Meta

abstract = False

NotDeletableModel.**delete**(**args*, ***kwargs*)

class seed.data_importer.models.**RangeValidationRule**(*id*, *table_column_mapping*, *passes*, *validationrule_ptr*, *max_value*, *min_value*, *limit_min*, *limit_max*)

Bases: *seed.data_importer.models.ValidationRule*

exception DoesNotExist

Bases: seed.data_importer.models.DoesNotExist

exception RangeValidationRule.**MultipleObjectsReturned**
Bases: seed.data_importer.models.MultipleObjectsReturned

RangeValidationRule.**objects = <django.db.models.manager.Manager object>**

RangeValidationRule.**validationrule_ptr**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

child.parent is a ForwardManyToOneDescriptor instance.

class seed.data_importer.models.**TableColumnMapping**(*id*, *app*, *source_string*, *import_file*, *destination_model*, *destination_field*, *order*, *confidence*, *ignored*, *was_a_human_decision*, *error_message_text*, *active*)
Bases: django.db.models.base.Model

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception TableColumnMapping.**MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

TableColumnMapping.**combined_model_and_field**

TableColumnMapping.**datacoercion_errors**

TableColumnMapping.**datacoercionmapping_set**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

parent.children is a ReverseManyToOneDescriptor instance.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

TableColumnMapping.**datacoercions**

TableColumnMapping.**destination_django_field**
commented out by AKL, not needed for SEED and removes dependency on libs.

TableColumnMapping.**destination_django_field_choices**

TableColumnMapping.**destination_django_field_has_choices**

TableColumnMapping.**fields_to_save = ['pk', 'destination_model', 'destination_field', 'ignored']**

TableColumnMapping.**first_five_rows**

TableColumnMapping.**first_row**

TableColumnMapping.**friendly_destination_field**

TableColumnMapping.**friendly_destination_model**

TableColumnMapping.**friendly_destination_model_and_field**

TableColumnMapping.**import_file**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

TableColumnMapping.**is_mapped**

TableColumnMapping.**objects = <django.db.models.manager.Manager object>**

TableColumnMapping.**save**(**args*, ***kwargs*)

TableColumnMapping.**source_string_sha**

TableColumnMapping.**validation_rules**

TableColumnMapping.**validationrule_set**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

class seed.data_importer.models.**ValidationOutlier**(*id*, *rule*, *value*)
Bases: django.db.models.base.Model

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception ValidationOutlier.**MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

ValidationOutlier.**objects = <django.db.models.manager.Manager object>**

ValidationOutlier.**rule**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

class seed.data_importer.models.**ValidationRule**(*id*, *table_column_mapping*, *passes*)
Bases: django.db.models.base.Model

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception ValidationRule.**MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

ValidationRule.**objects = <django.db.models.manager.Manager object>**

ValidationRule.**rangevalidationrule**
Accessor to the related object on the reverse side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

`place.restaurant` is a `ReverseOneToOneDescriptor` instance.

ValidationRule.**table_column_mapping**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

ValidationRule.**validationoutlier_set**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

seed.data_importer.models.**queue_update_status_for_import_record**(*pk*)
edited by AKL to trim down data_importer

seed.data_importer.models.**update_status_from_dcm**(*sender*, *instance*, ***kwargs*)

seed.data_importer.models.**update_status_from_import_file**(*sender*, *instance*, ***kwargs*)

seed.data_importer.models.**update_status_from_import_record**(*sender*, *instance*, ***kwargs*)

seed.data_importer.models.**update_status_from_tcm**(*sender*, *instance*, ***kwargs*)

6.6.4 URLs

6.6.5 Utils

class seed.data_importer.utils.**CoercionRobot**
Bases: object

lookup_hash(*uncoerced_value*, *destination_model*, *destination_field*)

make_key(*value*, *model*, *field*)

seed.data_importer.utils.**acquire_lock**(*name*, *expiration=None*)
Tries to acquire a lock from the cache. Also sets the lock's value to the current time, allowing us to see how long it has been held.

Returns False if lock already belongs by another process.

seed.data_importer.utils.**chunk_iterable**(*iter*, *chunk_size*)
Breaks an iterable (e.g. list) into smaller chunks, returning a generator of the chunk.

seed.data_importer.utils.**get_core_pk_column**(*table_column_mappings*, *primary_field*)

seed.data_importer.utils.**get_lock_time**(*name*)
Examines a lock to see when it was acquired.

seed.data_importer.utils.**release_lock**(*name*)
Frees a lock.

6.6.6 Views

class seed.data_importer.views.**DataImportBackend**(***kwargs*)
Bases: ajaxuploader.backends.local.LocalUploadBackend

Subclass of ajaxuploader's LocalUploadBackend, to handle creation of ImportFile objects related to the specified ImportRecord.

upload_complete(*request*, *filename*, **args*, ***kwargs*)
Called directly by fineuploader on upload completion.

seed.data_importer.views.**get_upload_details**(*request*, **args*, ***kwargs*)
Retrieves details about how to upload files to this instance.

Returns:

```
    If S3 mode:

    {
        'upload_mode': 'S3',
        'upload_complete': A url to notify that upload is complete,
        'signature': The url to post file details to for auth to upload to S3.
    }

    If local file system mode:

    {
        'upload_mode': 'filesystem',
        'upload_path': The url to POST files to (see local_uploader)
    }
```

seed.data_importer.views.**handle_s3_upload_complete**(*request*, **args*, ***kwargs*)
Notify the system that an upload to S3 has been completed. This is a necessary step after uploading to S3 or the SEED instance will not be aware the file exists.

Valid source_type values are found in seed.models.SEED_DATA_SOURCES

GET Expects the following in the query string:

key: The full path to the file, within the S3 bucket. E.g. data_importer/buildings.csv

source_type: The source of the file. E.g. 'Assessed Raw' or 'Portfolio Raw'

source_program: Optional value from common.mapper.Programs source_version: e.g. "4.1"

import_record: The ID of the ImportRecord this file belongs to.

Returns:

```
    {
        'success': True,
        'import_file_id': The ID of the newly-created ImportFile object.
    }
```

seed.data_importer.views.**sign_policy_document**(*request*, **args*, ***kwargs*)

Sign and return the policy document for a simple upload. http://aws.amazon.com/articles/1434/#signyours3postform

Payload:

```
    {
     "expiration": ISO-encoded timestamp for when signature should expire,
                   e.g. "2014-07-16T00:20:56.277Z",
     "conditions":
         [
             {"acl":"private"},
             {"bucket": The name of the bucket from get_upload_details},
             {"Content-Type":"text/csv"},
             {"success_action_status":"200"},
             {"key": filename of upload, prefixed with 'data_imports/',
                     suffixed with a unique timestamp.
                     e.g. 'data_imports/my_buildings.csv.1405469756'},
             {"x-amz-meta-category":"data_imports"},
             {"x-amz-meta-qqfilename": original filename}
         ]
    }
```

Returns:

```
    {
        "policy": A hash of the policy document. Using during upload to S3.
        "signature": A signature of the policy document.  Also used during upload to S3.
    }
```

6.6.7 Module contents

6.7 Features Package

6.7.1 Submodules

6.7.2 Steps

6.7.3 Module contents

6.8 Green Button Package

6.8.1 Subpackages

Green Button Tests Package

Submodules

XML Importer Tests

class seed.green_button.tests.test_xml_importer.**GreenButtonXMLImportTests**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

Tests of various ways of authenticating to the API.

Uses the get_building endpoint in all cases.

assert_models_created()
Tests that appropriate models for the sample xml file have been created.

setUp()

tearDown()

test_create_models()
Test of xml_importer.create_models.

test_import_xml()
Test of xml_importer.import_xml.

class seed.green_button.tests.test_xml_importer.**GreenButtonXMLParsingTests**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

Tests helper functions for pulling green button building data out of xml snippets.

assert_fn_mapping(*fn*, *mapping*)
Takes a function fn and a mapping from input values to expected output values. Asserts that fn returns the expected output for each of the input values.

setUp()

tearDown()

test_as_collection()
Test of xml_importer.as_collection.

test_building_data()
Test of xml_importer.building_data.

test_energy_type()
Test of xml_importer.energy_type.

test_energy_units()
Test of function that converts a green button 'uom' (unit of measurement?) integer to one of seed.models.ENERGY_UNITS.

test_interval_block_data()
Test of xml_importer.interval_block_data.

test_interval_data()
Test of xml_importer.interval_data.

test_meter_data()
Test of xml_importer.meter_data.

Module contents

6.8.2 Submodules

6.8.3 seed.green_button.xml_importer module

seed.green_button.xml_importer.**as_collection**(*val*)
Takes a value, returns that value if it is not a string and is an Iterable, and returns a list containing that value if it is not an Iterable or if it is a string. Returns None when val is None.

Parameters **val** – any value

Returns list containing val or val if it is Iterable and not a string.

seed.green_button.xml_importer.**building_data**(*xml_data*)
Extracts information about a building from a Green Button XML file.

Parameters **xml_data** – dictionary returned by xmltodict.parse when called on the contents of a Green Button XML file

Returns dictionary

•building information for a Green Button XML file

•information describing the meter used for collection

•list of time series meter reading data

seed.green_button.xml_importer.**create_models**(*data*, *import_file*)
Create a BuildingSnapshot, a CanonicalBuilding, and a Meter. Then, create TimeSeries models for each meter reading in data.

Parameters

- **data** – dictionary of building data from a Green Button XML file in the form returned by xml_importer.building_data
- **import_file** – ImportFile referencing the original xml file; needed for linking to BuildingSnapshot and for determining super_organization

Returns the created CanonicalBuilding

seed.green_button.xml_importer.**energy_type**(*service_category*)
Returns the seed model energy type corresponding to the green button service category.

Parameters **service_category** – int that is a green button service_category (string args will be converted to integers)

Returns int in seed.models.ENERGY_TYPES

seed.green_button.xml_importer.**energy_units**(*uom*)
Returns the seed model energy unit corresponding to the green button uom.

Parameters **uom** – int that is the green button uom number corresponding to the energy units supported by the green button schema (string args will be converted to integers)

Returns int in seed.models.ENERGY_UNITS

seed.green_button.xml_importer.**import_xml**(*import_file*)
Given an import_file referencing a raw Green Button XML file, extracts building and time series information from the file and constructs required database models.

Parameters **import_file** – a seed.models.ImportFile instance representing a Green Button XML file that has been previously uploaded

Returns the created CanonicalBuilding Inst.

seed.green_button.xml_importer.**interval_block_data**(*ib_xml_data*)
Takes a dictionary containing the contents of an IntervalBlock node from a Green Button XML file and returns a dictionary containing the start_time of the time series collection, the duration of the collection, and a list of readings containing the time series data from a meter.

Parameters **ib_xml_data** – dictionary of the contents of an IntervalBlock from a Green Button XML file

Returns dictionary containing meta data about an entire collection period and a list of the specific meter readings

seed.green_button.xml_importer.**interval_data**(*reading_xml_data*)
Takes a dictionary representing the contents of an IntervalReading XML node and pulls out data for a single time series reading. The dictionary will be a sub-dictionary of the dictionary returned by xmltodict.parse when called on a Green Button XML file. Returns a flat dictionary containing the interval data.

Parameters **reading_xml_data** – dictionary of IntervalReading XML node content in format specified by the xmltodict library.

Returns dictionary representing a time series reading with keys 'cost', 'value', 'start_time', and 'duration'.

seed.green_button.xml_importer.**meter_data**(*raw_meter_meta*)
Takes a dictionary representing the contents of the entry node in a Green Button XML file that specifies the meta data about the meter that was used to record time series data for that file. Returns a flat dictionary containing the meter meta data.

Parameters **raw_meter_meta** – dictionary of the contents of the meter specification entry node in a Green Button XML file

Returns dictionary containing information about a meter with keys 'currency', 'power_of_ten_multiplier', and 'uom'

6.8.4 Module contents

6.9 Landing Package

6.9.1 Subpackages

seed.landing.management package

Subpackages

Landing Management Package

Submodules

Update EULA

class seed.landing.management.commands.update_eula.**Command**(*stdout=None*, *stderr=None*, *no_color=False*)

Bases: django.core.management.base.BaseCommand

handle(**args*, ***options*)

help = 'Update the Terms of Service with a new HTML file'

Module contents

Module contents

6.9.2 Submodules

6.9.3 Forms

class seed.landing.forms.**LoginForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*)

Bases: django.forms.forms.Form

base_fields = OrderedDict([('email', <django.forms.fields.EmailField object at 0x7f18fb532910>), ('password', <djan

declared_fields = OrderedDict([('email', <django.forms.fields.EmailField object at 0x7f18fb532910>), ('password', <

media

6.9.4 Models

class seed.landing.models.**SEEDUser**(**args*, ***kwargs*)

Bases: django.contrib.auth.base_user.AbstractBaseUser, django.contrib.auth.models.PermissionsMixin

An abstract base class implementing a fully featured User model with admin-compliant permissions.

Username, password and email are required. Other fields are optional.

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception SEEDUser.**MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

SEEDUser.**REQUIRED_FIELDS = ['email']**

SEEDUser.**USERNAME_FIELD = 'username'**

SEEDUser.**auditlog_set**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

parent.children is a ReverseManyToOneDescriptor instance.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

SEEDUser.**buildingsnapshot_set**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

parent.children is a ReverseManyToOneDescriptor instance.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

SEEDUser.**columnmapping_set**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

parent.children is a ReverseManyToOneDescriptor instance.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

SEEDUser.**default_building_detail_custom_columns**
A placeholder class that provides a way to set the attribute on the model.

SEEDUser.**default_custom_columns**
A placeholder class that provides a way to set the attribute on the model.

SEEDUser.**default_organization**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

`SEEDUser.`**`email_user`**`(`*subject*, *message*, *from_email=None*`)`
Sends an email to this User.

`SEEDUser.`**`generate_key`**`()`
Creates and sets an API key for this user. Adapted from tastypie:

https://github.com/toastdriven/django-tastypie/blob/master/tastypie/models.py#L47 # noqa

`SEEDUser.`**`get_absolute_url`**`()`

`SEEDUser.`**`get_full_name`**`()`
Returns the first_name plus the last_name, with a space in between.

`SEEDUser.`**`get_next_by_date_joined`**`(`**moreargs*, ***morekwargs*`)`

`SEEDUser.`**`get_previous_by_date_joined`**`(`**moreargs*, ***morekwargs*`)`

`SEEDUser.`**`get_short_name`**`()`
Returns the short name for the user.

`SEEDUser.`**`groups`**
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`pizza.toppings` and `topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

`SEEDUser.`**`importrecord_set`**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

`SEEDUser.`**`last_modified_user`**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

`SEEDUser.`**`logentry_set`**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

`SEEDUser.`**`modified_import_records`**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

`SEEDUser.`**`objects = <django.contrib.auth.models.UserManager object>`**

`SEEDUser.`**`organizationuser_set`**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

`SEEDUser.`**`orgs`**
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`pizza.toppings` and `topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

`SEEDUser.`**`project_set`**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

SEEDUser.**projectbuilding_set**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

SEEDUser.**save**(**args*, ***kwargs*)
Ensure that email and username are synced.

SEEDUser.**user_agreement**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

SEEDUser.**user_permissions**
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`pizza.toppings` and `topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

6.9.5 Tests

class seed.landing.tests.**UserLoginTest**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

setUp()

test_simple_login()
Happy path login with no ToS.

test_tos_login()
Happy path login when there is a ToS.

6.9.6 URLs

6.9.7 Views

seed.landing.views.**landing_page**(*request*)

seed.landing.views.**login_view**(*request*)

Standard Django login, with additions: Lowercase the login email (username) Check user has accepted ToS, if any.

seed.landing.views.**password_reset**(*request*)

seed.landing.views.**password_reset_complete**(*request*)

seed.landing.views.**password_reset_confirm**(*request*, *uidb64=None*, *token=None*)

seed.landing.views.**password_reset_done**(*request*)

seed.landing.views.**password_set**(*request*, *uidb64=None*, *token=None*)

seed.landing.views.**signup**(*request*, *uidb64=None*, *token=None*)

6.9.8 Module contents

6.10 Library Packages

6.10.1 Submodules

6.10.2 Exporter Module

class seed.lib.exporter.**Exporter**(*export_id*, *export_name*, *export_type*)
Class to handle the exporting of buildings

export(*buildings*, *fields*, *row_cb*)
The main method of export. Uses the export type defined by the initializer

Parameters

- **buildings** – Array of building ids to export
- **fields** – Array of fields to export
- **row_cb** – ID for row cache

Returns

export_csv(*qs*, *fields=[]*, *cb=None*)

export_xls(*qs*, *fields=[]*, *cb=None*)

static fields_from_queryset(*qs*)
Creates a list of all accessible fields on a model based off of a queryset.

This method should not be here. It seems that is should be on the building snapshot model. Not moved yet because I am unsure if the qs argument is more than one data type (i.e. BuildingSnapshot and/or ?)

filename()
The expected file name based on the export_id, export_name, and export_type

Returns String of the expected filename

subdirectory()
Create and return the subdirectory

Returns String of the subdirectory

static subdirectory_from_export_id(*export_id*)
Return the subdirectory as constructed by the instance method.

Parameters **export_id** – The export ID

Returns String of the path to the exported file

tempfile = None

valid_export_type()

seed.lib.exporter.**batch_qs**(*qs*, *batch_size=1000*)
From: https://djangosnippets.org/snippets/1170/

Returns a (start, end, total, queryset) tuple for each batch in the given queryset.

Usage:

seed.lib.exporter.**construct_obj_row**(*obj*, *fields*)
Creates an exportable row of data from an object and a list of fields. Ignores nones and instances of the Django Manager object, replacing them with blank unicode strings.

seed.lib.exporter.**get_field_name_from_model**(*field*, *model*)
Takes a field name like "building_snapshot__state" and returns the verbose field name as set in django, to be used as the header in exported files.

Parameters

- **field** –
- **qs** –

Returns

seed.lib.exporter.**get_field_value_from_instance**(*field*, *obj*)
Does some deep diving to find the right value given a string like "building_snapshot__state"

seed.lib.exporter.**qs_to_rows**(*qs*, *fields*)

6.10.3 Module contents

6.11 Mapping Package

6.11.1 Submodules

6.11.2 seed.mappings.mapper module

seed.mappings.mapper.**get_attr_source**(*field_values*, *value*)
Return the first dictionary key that contains a value.

seed.mappings.mapper.**get_building_attrs**(*data_set_buildings*)
Returns a dictionary of attributes from each data_set_building.

Parameters **buildings** – list, group of BS instances to merge.

Returns BuildingSnapshot dict: possible attributes keyed on attr name.

seed.mappings.mapper.**get_source_id**(*source_inst*, *attr*)
Get the ID we save for our model source from `models` module.

seed.mappings.mapper.**merge_building**(*snapshot*, *b1*, *b2*, *can_attrs*, *conf*, *default=None*, *match_type=None*)
Set attributes on our Canonical model, saving differences.

Parameters

- **snapshot** – BuildingSnapshot model inst.
- **b1** – BuildingSnapshot model inst. Left parent.
- **b2** – BuildingSnapshot model inst. Right parent.
- **can_attrs** – dict of dicts, {'attr_name': {'dataset1': 'value'...}}.
- **default** – (optional), which dataset's value to default to.

Rtype default BuildingSnapshot

Returns inst(`snapshot`), updated.

`seed.mappings.mapper.`**`merge_extra_data`**(*b1*, *b2*, *default=None*)
Merge extra_data field between two BuildingSnapshots, return result.

Parameters

- **b1** – BuildingSnapshot inst.
- **b2** – BuildingSnapshot inst.
- **default** – BuildingSnapshot inst.

Returns tuple of dict

`seed.mappings.mapper.`**`save_variant`**(*snapshot*, *attr*, *attribute_values*)
Save different options from each dataset for a Canonical field value.

Parameters

- **snapshot** – BuildingSnapshot inst.
- **attr** – string, the disputed attribute on the can_building inst.

Attribute_values dict of obj:str. Keyed on datasource model instance.

6.11.3 seed.mappings.reconcile_mappings module

6.11.4 seed.mappings.seed_mappings module

How to map dataset attributes to CanonicalBuilding.

If the first element in the tuple is a callable, it will be passed a model instance for that type of mapping (AssessedBuilding for AssessedBuilding_to_CanonicalBuilding, etc.)

6.11.5 Module contents

6.12 Managers Package

6.12.1 Subpackages

Manager Tests Package

Submodules

Test JSON Manager

class seed.managers.tests.test_json_manager.**TestJsonManager**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

setUp()

test_order_by_returns_all_buildings()
Test that we're able to order by values of a json field.

Module contents

6.12.2 Submodules

6.12.3 JSON

class seed.managers.json.**JsonManager**
Bases: django.db.models.manager.Manager

get_queryset()

class seed.managers.json.**JsonQuerySet**(*model=None*, *query=None*, *using=None*, *hints=None*)
Bases: django.db.models.query.QuerySet

PRIMARY = 'extra_data'

TABLE = 'seed_buildingsnapshot'

json_order_by(*key*, *order_by*, *order_by_rev=False*, *unit=None*)

6.12.4 Module contents

6.13 Public Package

6.13.1 Submodules

6.13.2 Models

Because migrations are complicated, we're keeping our public fields here.

This deals with circular dependency issues between LANDINGUser and Organization

class seed.public.models.**SharedBuildingField**(**args*, ***kwargs*)
Bases: django_extensions.db.models.TimeStampedModel

BuildingSnapshot Exported Field, either public or internally shared.

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception SharedBuildingField.**MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

SharedBuildingField.**field**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

child.parent is a ForwardManyToOneDescriptor instance.

SharedBuildingField.**get_field_type_display**(**moreargs*, ***morekwargs*)

SharedBuildingField.**get_next_by_created**(**moreargs*, ***morekwargs*)

SharedBuildingField.**get_next_by_modified**(**moreargs*, ***morekwargs*)

SharedBuildingField.**get_previous_by_created**(**moreargs*, ***morekwargs*)

SharedBuildingField.**get_previous_by_modified**(**moreargs*, ***morekwargs*)

SharedBuildingField.**objects = <django.db.models.manager.Manager object>**

SharedBuildingField.**org**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

child.parent is a ForwardManyToOneDescriptor instance.

6.13.3 Module contents

6.14 SEED Package

6.14.1 Subpackages

Management Package

Subpackages

Management Packages

Submodules

S3

class seed.management.commands.set_s3_expires_headers_for_angularjs_partials.**Command**(*stdout=Nor*
stderr=Nor
no_color=F

Bases: django.core.management.base.BaseCommand

handle(**args*, ***options*)

help = "Sets S3 Expires headers for AngularJS partials to prevent browser caching old html partials. ./manage.py set_s3

option_list = (<Option at 0x7f18f73767e8: –prefix>,)

Module contents

Module contents

Templatetags Package

Submodules

Breadcrumbs

class seed.templatetags.breadcrumbs.**BreadcrumbNode**(*vars*, *render_func=<function create_crumb>*)

Bases: django.template.base.Node

render(*context*)

class seed.templatetags.breadcrumbs.**UrlBreadcrumbNode**(*title*, *url_node*, *render_func=<function create_crumb>*)

Bases: django.template.base.Node

render(*context*)

seed.templatetags.breadcrumbs.**breadcrumb**(*parser*, *token*)

Section author: Andriy Drozdyuk

Renders the breadcrumb.

Example:

```
{% breadcrumb "Title of breadcrumb" url_var %}
{% breadcrumb context_var  url_var %}
{% breadcrumb "Just the title" %}
{% breadcrumb just_context_var %}
```

Parameters:

```
First parameter is the title of the crumb
Second (optional) parameter is the url variable to link to, produced by url tag, i.e.:
    {% url "person_detail" object.id as person_url %}
    then:
    {% breadcrumb person.name person_url %}
```

seed.templatetags.breadcrumbs.**breadcrumb_root**(*parser*, *token*)

Section author: Andriy Drozdyuk

Renders the breadcrumb.

Examples:

```
{% breadcrumb "Title of breadcrumb" url_var %}
{% breadcrumb context_var  url_var %}
{% breadcrumb "Just the title" %}
{% breadcrumb just_context_var %}
```

Parameters:

```
First parameter is the title of the crumb,
Second (optional) parameter is the url variable to link to, produced by url tag, i.e.:
    {% url "person_detail/" object.id as person_url %}
    then:
    {% breadcrumb person.name person_url %}
```

seed.templatetags.breadcrumbs.**breadcrumb_url**(*parser*, *token*)
Same as breadcrumb but instead of url context variable takes in all the arguments URL tag takes.

```
{% breadcrumb "Title of breadcrumb" person_detail person.id %}
{% breadcrumb person.name person_detail person.id %}
```

seed.templatetags.breadcrumbs.**breadcrumb_url_root**(*parser*, *token*)
Same as breadcrumb but instead of url context variable takes in all the arguments URL tag takes.

```
{% breadcrumb "Title of breadcrumb" person_detail person.id %}
{% breadcrumb person.name person_detail person.id %}
```

seed.templatetags.breadcrumbs.**create_crumb**(*title*, *url=None*)
Helper function

seed.templatetags.breadcrumbs.**create_crumb_first**(*title*, *url=None*)
Helper function

Test Helpers Package

Subpackages

Test Helper Factor Package

Subpackages

Test Helper Factory Lib Package

Submodules

Chomsky

seed.test_helpers.factory.lib.chomsky.**generate_chomsky**(*times=5*, *line_length=72*)

Submodules

Helpers
class seed.test_helpers.factory.helpers.**DjangoFunctionalFactory**

classmethod invalid_test_cc_number()

classmethod rand_bool()

classmethod rand_city()

classmethod rand_city_suffix()

classmethod rand_currency(*start=0*, *end=100*)

classmethod rand_date(*start_year=1900*, *end_year=2011*)

classmethod rand_domain()

classmethod rand_email()

classmethod rand_float(*start=0*, *end=100*)

classmethod rand_int(*start=0*, *end=100*)

classmethod rand_name()

classmethod rand_phone()

classmethod rand_plant_name()

classmethod rand_str(*length=None*)

classmethod rand_street_address()

classmethod rand_street_suffix()

classmethod random_conversation(*paragraphs=3*)

classmethod test_cc_number(*valid=True*)

classmethod valid_test_cc_number()

Module contents

Tests Package

Submodules

Tests (Functional) Package

Submodules

Test API

Test Firefox

Admin Views

class seed.tests.test_admin_views.**AdminViewsTest**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

setUp()

test_add_org()
Happy path test for creating a new org.

test_add_org_dupe()
Trying to create an org with a dupe name fails.

test_add_user_existing_org()
Test creating a new user, adding them to an existing org in the process.

test_add_user_new_org()
Create a new user and a new org at the same time.

test_add_user_no_org()
Shouldn't be able to create a new user without either selecting or creating an org at the same time.

test_signup_process()
Simulates the entire new user signup process, from initial account creation by an admin to receiving the signup email to confirming the account and setting a password.

test_signup_process_force_lowercase_email()
Simulates the signup and login forcing login username to lowercase

Decorators

class seed.tests.test_decorators.**RequireOrganizationIDTests**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

setUp()

test_require_organization_id_fail_no_key()

test_require_organization_id_fail_not_numeric()

test_require_organization_id_success_integer()

test_require_organization_id_success_string()

class seed.tests.test_decorators.**TestDecorators**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

Tests for locking tasks and reporting progress.

locked = 1

pk = 34

setUp()

test_get_prog_key()
We format our cache key properly.

test_increment_cache()
Sum our progress by increments properly.

test_locking()
Make sure we indicate we're locked if and only if we're inside the function.

test_locking_w_exception()
Make sure we release our lock if we've had an exception.

test_progress()
When a task finishes, it increments the progress counter properly.

unlocked = 0

exception seed.tests.test_decorators.**TestException**
Bases: exceptions.Exception

Exporters

class seed.tests.test_exporters.**TestExporters**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

Tests for exporting data to various formats.

setUp()

tearDown()

test_csv_export()
Ensures exported CSV data matches source data

test_csv_export_extra_data()
Ensures exported CSV data matches source data

test_data_model_assumptions()
Some parts of export make certain assumptions about the data model, this test ensures that those assumptions are true.

test_xls_export()
Ensures exported XLS data matches source data

Models

class seed.tests.test_models.**TestBuildingSnapshot**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

Test the clean methods on BuildingSnapshotModel.

bs1_data = {'custom_id_1': 1243, 'city': 'Gotham City', 'pm_property_id': 1243, 'postal_code': 8999, 'tax_lot_id': '43

bs2_data = {'custom_id_1': 37376, 'city': 'Gotham City', 'pm_property_id': 9999, 'postal_code': 8999, 'tax_lot_id': '1

setUp()

test_clean()
Make sure we convert datestrings properly.

test_create_child()
Child BS has reference to parent.

test_get_column_mapping()
Honor organizational bounds, get mapping data.

test_get_column_mappings()
We produce appropriate data structure for mapping

test_get_tip()
BS tip should point to the end of the tree.

test_merge_extra_data()
extra_data dicts get merged proper-like.

test_merge_extra_data_does_not_override_with_blank_data()
Test that blank fields in extra data don't override real data

test_merge_extra_data_no_data()
Test edgecase where there is no extra_data to merge.

test_recurse_tree()
Make sure we get an accurate child tree.

test_remove_child()
Test behavior for removing a child.

test_save_snapshot_match_default_to_first_building()
Test good case for saving a snapshot match with the first building as default.

test_save_snapshot_match_default_to_second_building()
Test good case for saving a snapshot match with the second building as default.

test_source_attributions()
Test that we can point back to an attribute's source.

This is explicitly just testing the low-level data model, none of the convenience functions.

test_unmatch_snapshot_tree_last_match()
Tests the simplest case of unmatching a building where the child snapshot created from the original matching has not since been matched with another building (no children).

test_unmatch_snapshot_tree_prior_match()
Tests the more complicated case of unmatching a building after more buildings have been matched to the snapshot resulting from the original match.

test_unmatch_snapshot_tree_retains_canonical_snapshot()
TODO:

test_update_building()
Good case for updating a building.

test_update_building_with_dates()

class seed.tests.test_models.**TestCanonicalBuilding**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

Test the clean methods on CanonicalBuildingModel.

test_repr()

class seed.tests.test_models.**TestColumnMapping**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

Test ColumnMapping utility methods.

setUp()

test_is_concatenated()

test_is_direct()

Tasks

class seed.tests.test_tasks.**TestCleaner**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

Tests that our logic for constructing cleaners works.

setUp()

test_build_cleaner()

class seed.tests.test_tasks.**TestTasks**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

Tests for dealing with SEED related tasks.

setUp()

test_cached_first_row_order()
Tests to make sure the first row is saved in the correct order. It should be the order of the headers in the original file.

test_delete_organization()

test_delete_organization_buildings()
tests the delete buildings for an organization

test_delete_organization_doesnt_delete_user_if_multiple_memberships()
Deleting an org shouldn't delete the orgs users if the user belongs to many orgs.

test_get_ancestors()
Tests get_ancestors(building), returns all non-composite, non-raw BuildingSnapshot instances.

test_handle_id_matches_duplicate_data()
Test for handle_id_matches behavior when matching duplicate data

test_is_same_snapshot()
Test to check if two snapshots are duplicates

test_map_data()
Save mappings based on user specifications.

test_mapping_w_concat()
When we have a json encoded list as a column mapping, we concat.

test_match_buildings()
Good case for testing our matching system.

test_match_duplicate_buildings()
Test for behavior when trying to match duplicate building data

test_match_no_canonical_buildings()
If no canonicals exist, create, but no new BuildingSnapshots.

test_match_no_matches()
When a canonical exists, but doesn't match, we create a new one.

test_no_unmatched_buildings()
Make sure we shortcut out if there isn't unmatched data.

test_save_raw_data()
Save information in extra_data, set other attrs.

test_save_raw_data_batch_iterator()
Ensure split_csv completes

test_separates_system_and_possible_match_types()
We save possible matches separately.

class seed.tests.test_tasks.**TestTasksXLS**(*methodName='runTest'*)
Bases: *seed.tests.test_tasks.TestTasks*

Runs the TestTasks tests with an XLS file

setUp()

class seed.tests.test_tasks.**TestTasksXLSX**(*methodName='runTest'*)
Bases: *seed.tests.test_tasks.TestTasks*

Runs the TestsTasks tests with an XLSX file.

setUp()

Views

class seed.tests.test_views.**BuildingDetailViewTests**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

Tests of the SEED Building Detail page

setUp()

test_get_building()
tests the get_building view which returns building detail and source information from parent buildings.

test_get_building_imported_buildings_includes_green_button()

test_get_building_with_deleted_dataset()
tests the get_building view where the dataset has been deleted and the building should load without showing the sources from deleted import files.

test_get_building_with_project()
tests get_building projects payload

test_get_match_tree()
tests get_match_tree

test_get_match_tree_from_child()
tests get_match_tree from the child

test_save_match_audit_log()
tests that a building match logs an audit_log

test_save_match_invalid_org()
tests that a building match checks perm of org id

test_save_match_wrong_perms_different_building_orgs()
tests that a building match is valid for BS orgs

test_save_match_wrong_perms_org_id()
tests that a building match is valid for the org id

test_save_unmatch_audit_log()
tests that a building unmatch logs an audit_log

test_update_building_audit_log()
tests that a building update logs an audit_log

class seed.tests.test_views.**DataImporterViewTests**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

Tests of the data_importer views (and the objects they create).

setUp()

test_get_first_five_rows()
Make sure we get our first five rows back correctly.

test_get_first_five_rows_with_newlines()

test_get_raw_column_names()
Make sure we get column names back in a format we expect.

class seed.tests.test_views.**DefaultColumnsViewTests**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

Tests of the SEED default custom saved columns

setUp()

tearDown()

test_get_columns()

test_get_default_columns_initial_state()

test_get_default_columns_with_set_columns()

test_set_default_columns()

class seed.tests.test_views.**GetDatasetsViewsTests**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

setUp()

test_delete_dataset()

test_get_dataset()

test_get_datasets()

test_update_dataset()

class seed.tests.test_views.**ImportFileViewsTests**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

setUp()

test_delete_duplicates_from_import_file()

test_delete_file()

test_get_import_file()

test_get_pm_filter_by_counts()

class seed.tests.test_views.**MainViewTests**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

setUp()

test_create_pm_mapping()

test_export_buildings()

test_export_buildings_empty()

test_export_buildings_progress()

test_home()

class seed.tests.test_views.**MatchTreeTests**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

Currently only tests _parent_tree_coparents

setUp()

test_get_coparents()

test_parent_tree_coparents()
Tests that _parent_tree_coparents returns what we expect

class seed.tests.test_views.**ReportViewsTests**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

setUp()

test_get_aggregated_building_report_data()

test_get_building_report_data()

test_get_building_summary_report_data()

class seed.tests.test_views.**SearchBuildingSnapshotsViewTests**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

setUp()

test_search_building_snapshots()

class seed.tests.test_views.**SearchViewTests**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

Tests of the SEED search_buildings

setUp()

test_seach_active_canonicalbuildings()
tests the search_buildings method used throughout the app for only returning active CanonicalBuilding BuildingSnapshot instances.

test_search_case_insensitive_exact_match()
Tests search_buildings method when called with a case insensitive exact match.

test_search_empty_column()
Tests search_buildings method when called with an empty column query.

test_search_exact_exclude_filter()

test_search_exact_match()
Tests search_buildings method when called with an exact match.

test_search_exclude_filter()

test_search_extra_data()
tests the search_buildings method used throughout the app for only returning active CanonicalBuilding BuildingSnapshot instances.

test_search_extra_data_empty_column()
Empty column query on extra_data key should match key's value being empty in JsonField.

test_search_extra_data_exact_exclude_filter()

test_search_extra_data_exact_match()
Exact match on extra_data json keys

test_search_extra_data_exclude_filter()

test_search_extra_data_non_empty_column()
Not-empty column query on extra_data key.

test_search_extra_data_non_existent_column()
Empty column query on extra_data key should match key not existing in JsonField.

test_search_filter_date_range_ISO8601()

test_search_filter_range()
Tests search_buildings method when called with a range.

test_search_not_empty_column()
Tests search_buildings method when called with a not-empty column query.

test_search_sort()
tests the search_buildings method used throughout the app for only returning active CanonicalBuilding BuildingSnapshot instances.

test_sort_extra_data()
Tests that sorting on extra data takes the column type into account.

class seed.tests.test_views.**TestMCMViews**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

assert_expected_mappings(*actual*, *expected*)
For each k,v pair of form column_name: [dest_col, confidence] in actual, assert that expected contains the same column_name and dest_col mapping.

expected_mappings = {u'year built': [u'year_built', 50], u'building id': [u'Building air leakage', 64], u'name': [u'Na

raw_columns_expected = {u'status': u'success', u'raw_columns': [u'name', u'address', u'year built', u'building id']}

setUp()

test_create_dataset()
tests the create_dataset view, allows duplicate dataset names

test_get_column_mapping_suggestions()

test_get_column_mapping_suggestions_pm_file()

test_get_column_mapping_suggestions_with_columns()

test_get_raw_column_names()
Good case for get_raw_column_names.

test_progress()
Make sure we retrieve data from cache properly.

test_remap_buildings()
Test good case for resetting mapping.

test_reset_mapped_w_matching_done()
Make sure we don't delete buildings that have been merged.

test_reset_mapped_w_previous_matches()
Ensure we ignore mapped buildings with children BuildingSnapshots.

test_save_column_mappings()

test_save_column_mappings_idempotent()
We need to make successive calls to save_column_mappings.

test_save_column_mappings_w_concat()
Concatenated payloads come back as lists.

Tests

class seed.tests.tests.**ComplianceTestCase**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

test_basic_compliance_creation()

class seed.tests.tests.**ProjectTestCase**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

test_basic_project_creation()

class seed.tests.tests.**UtilsTests**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

setUp()

test_get_buildings_count_for_user()

Utils

class seed.tests.util.**FakeClient**
Bases: object

An extremely light-weight test client.

get(*view_func*, *data*, *headers=None*, ***kwargs*)

post(*view_func*, *data*, *headers=None*, ***kwargs*)

class seed.tests.util.**FakeRequest**(*data=None*, *headers=None*, *user=None*, *method='POST'*, ***kwargs*)
Bases: object

A simple request stub.

GET = {}

META = {'REMOTE_ADDR': '127.0.0.1'}

POST = {}

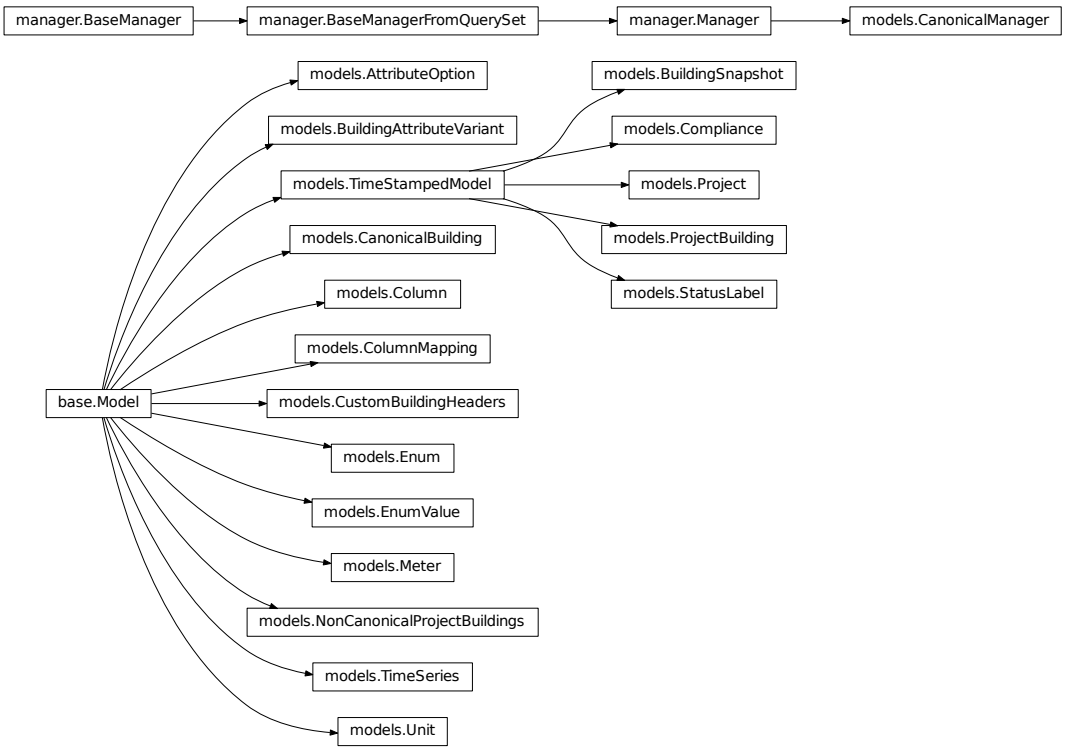
body = None

path = 'fake_login_path'

seed.tests.util.**make_fake_mappings**(*mappings*, *org*)
Takes a dict and saves a ColumnMapping object for each key

seed.tests.util.**make_fake_snapshot**(*import_file*, *init_data*, *bs_type*, *is_canon=False*, *org=None*)
For making fake mapped BuildingSnapshots to test matching against.

6.14.2 Inheritance



6.14.3 Submodules

6.14.4 Decorators

seed.decorators.**DecoratorMixin**(*decorator*)

Converts a decorator written for a function view into a mixin for a class-based view.

Example:

```
LoginRequiredMixin = DecoratorMixin(login_required)
class MyView(LoginRequiredMixin):
    pass

class SomeView(DecoratorMixin(some_decorator), DecoratorMixin(something_else)):
    pass
```

seed.decorators.**ajax_request**(*func*)

•Copied from django-annoying, with a small modification. Now we also check for 'status' or 'success' keys and return correct status codes

If view returned serializable dict, returns response in a format requested by HTTP_ACCEPT header. Defaults to JSON if none requested or match.

Currently supports JSON or YAML (if installed), but can easily be extended.

Example:

```
@ajax_request
def my_view(request):
    news = News.objects.all()
    news_titles = [entry.title for entry in news]
    return { 'news_titles': news_titles }
```

seed.decorators.**get_prog_key**(*func_name*, *import_file_pk*)
Return the progress key for the cache

seed.decorators.**lock_and_track**(*fn*, **args*, ***kwargs*)
Decorator to lock tasks to single executor and provide progress url.

seed.decorators.**require_organization_id**(*fn*)
Validate that organization_id is in the GET params and it's an int.

6.14.5 Factory

class seed.factory.**SEEDFactory**
Bases: *seed.test_helpers.factory.helpers.DjangoFunctionalFactory*

model factory for SEED

classmethod building_snapshot(*canonical_building=None*, **args*, ***kwargs*)
creates an BuildingSnapshot inst.

if canonical_building (CanonicalBuilding inst.) is None, then a CanonicalBuilding inst. is created and a BuildingSnapshot inst. is created and linked to the CanonicalBuilding inst.

6.14.6 Models

class seed.models.**AttributeOption**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Holds a single conflicting value for a BuildingSnapshot attribute.

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception AttributeOption.**MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

AttributeOption.**building_variant**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

child.parent is a ForwardManyToOneDescriptor instance.

AttributeOption.**get_value_source_display**(**moreargs*, ***morekwargs*)

AttributeOption.**objects = <django.db.models.manager.Manager object>**

class seed.models.**BuildingAttributeVariant**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Place to keep the options of BuildingSnapshot attribute variants.

When we want to select which source's values should sit in the Canonical Building's position, we need to draw from a set of options determined during the matching phase. We should only have one 'Variant' container per field_name, per snapshot.

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception BuildingAttributeVariant.**MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

BuildingAttributeVariant.**building_snapshot**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

child.parent is a ForwardManyToOneDescriptor instance.

BuildingAttributeVariant.**objects = <django.db.models.manager.Manager object>**

BuildingAttributeVariant.**options**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

parent.children is a ReverseManyToOneDescriptor instance.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

class seed.models.**BuildingSnapshot**(**args*, ***kwargs*)
Bases: django_extensions.db.models.TimeStampedModel

The periodical composite of a building from disparate data sources.

Represents the best data between all the data sources for a given building, potentially merged together with other BuildingSnapshot instances' attribute values.

Two BuildingSnapshots can create a child, forming a match between buildings. Thusly, a BuildingSnapshot's co-parent is the other parent of its child. The m2m field *children* with related name *parents* allow the traversal of the tree. A BuildingSnapshot can have one parent in the case where an edit to data was initiated by a user, and the original field is preserved (treating BuildingSnapshots as immutable objects) and a new BuildingSnapshot is created with the change.

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception BuildingSnapshot.**MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

BuildingSnapshot.**address_line_1_source**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

child.parent is a ForwardManyToOneDescriptor instance.

BuildingSnapshot.**address_line_2_source**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

BuildingSnapshot.**best_guess_canonical_building**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

BuildingSnapshot.**block_number_source**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

BuildingSnapshot.**building_certification_source**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

BuildingSnapshot.**building_count_source**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

BuildingSnapshot.**canonical_building**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

BuildingSnapshot.**canonical_for_ds**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

`BuildingSnapshot.`**`canonicalbuilding_set`**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

`BuildingSnapshot.`**`child_tree`**
Recurse to give us a merge-order list of children.

`BuildingSnapshot.`**`children`**
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`pizza.toppings` and `topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

`BuildingSnapshot.`**`city_source`**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

`BuildingSnapshot.`**`clean`**(**args*, ***kwargs*)

`BuildingSnapshot.`**`co_parent`**
returns the first co-parent as a BuildingSnapshot inst

`BuildingSnapshot.`**`co_parents`**
returns co-parents for a BuildingSnapshot as a queryset

`BuildingSnapshot.`**`conditioned_floor_area_source`**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

BuildingSnapshot.**custom_id_1_source**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

BuildingSnapshot.**district_source**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

BuildingSnapshot.**duplicate**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

BuildingSnapshot.**energy_alerts_source**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

BuildingSnapshot.**energy_score_source**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

BuildingSnapshot.**extra_data**
A placeholder class that provides a way to set the attribute on the model.

BuildingSnapshot.**extra_data_sources**
A placeholder class that provides a way to set the attribute on the model.

BuildingSnapshot.**generation_date_source**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

child.parent is a ForwardManyToOneDescriptor instance.

BuildingSnapshot.**get_match_type_display**(**moreargs*, ***morekwargs*)

BuildingSnapshot.**get_next_by_created**(**moreargs*, ***morekwargs*)

BuildingSnapshot.**get_next_by_modified**(**moreargs*, ***morekwargs*)

BuildingSnapshot.**get_previous_by_created**(**moreargs*, ***morekwargs*)

BuildingSnapshot.**get_previous_by_modified**(**moreargs*, ***morekwargs*)

BuildingSnapshot.**get_source_type_display**(**moreargs*, ***morekwargs*)

BuildingSnapshot.**gross_floor_area_source**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

child.parent is a ForwardManyToOneDescriptor instance.

BuildingSnapshot.**has_children**

BuildingSnapshot.**import_file**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

child.parent is a ForwardManyToOneDescriptor instance.

BuildingSnapshot.**last_modified_by**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

child.parent is a ForwardManyToOneDescriptor instance.

BuildingSnapshot.**lot_number_source**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

child.parent is a ForwardManyToOneDescriptor instance.

BuildingSnapshot.**meters**
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`pizza.toppings` and `topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

`BuildingSnapshot.`**`objects = <seed.managers.json.JsonManager object>`**

`BuildingSnapshot.`**`occupied_floor_area_source`**

Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

`BuildingSnapshot.`**`owner_address_source`**

Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

`BuildingSnapshot.`**`owner_city_state_source`**

Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

`BuildingSnapshot.`**`owner_email_source`**

Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

`BuildingSnapshot.`**`owner_postal_code_source`**

Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

`BuildingSnapshot.`**`owner_source`**

Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

`BuildingSnapshot.`**`owner_telephone_source`**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

`BuildingSnapshot.`**`parent_tree`**
Recurse to give us merge-order list of parents.

`BuildingSnapshot.`**`parents`**
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`pizza.toppings` and `topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

`BuildingSnapshot.`**`pm_property_id_source`**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

`BuildingSnapshot.`**`postal_code_source`**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

`BuildingSnapshot.`**`project_building_snapshots`**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

`BuildingSnapshot.`**`project_set`**
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`pizza.toppings` and `topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

BuildingSnapshot.**property_name_source**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

BuildingSnapshot.**property_notes_source**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

BuildingSnapshot.**recent_sale_date_source**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

BuildingSnapshot.**recurse_tree**(*attr*)
Recurse M2M relationship tree, extending list as we go.

Parameters **attr** – str, name of attribute we wish to traverse. .e.g. 'children', or 'parents'

BuildingSnapshot.**release_date_source**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

BuildingSnapshot.**save**(**args*, ***kwargs*)

BuildingSnapshot.**site_eui_source**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

BuildingSnapshot.**site_eui_weather_normalized_source**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

BuildingSnapshot.**source_eui_source**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

BuildingSnapshot.**source_eui_weather_normalized_source**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

BuildingSnapshot.**space_alerts_source**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

BuildingSnapshot.**state_province_source**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

BuildingSnapshot.**super_organization**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

BuildingSnapshot.**tax_lot_id_source**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

`BuildingSnapshot.`**`tip`**
returns the tip (leaf) of the BuildingSnapshot tree

`BuildingSnapshot.`**`to_dict`**(*fields=None*, *include_related_data=True*)
Returns a dict version of this building, either with all fields or masked to just those requested.

`BuildingSnapshot.`**`use_description_source`**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

`BuildingSnapshot.`**`variants`**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

`BuildingSnapshot.`**`year_built_source`**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

`BuildingSnapshot.`**`year_ending_source`**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

class `seed.models.`**`CanonicalBuilding`**(**args*, ***kwargs*)
Bases: `django.db.models.base.Model`

One Table to rule them all, One Table to find them, One Table to bring them all and in the database bind them.

exception `DoesNotExist`
Bases: `django.core.exceptions.ObjectDoesNotExist`

exception `CanonicalBuilding.`**`MultipleObjectsReturned`**
Bases: `django.core.exceptions.MultipleObjectsReturned`

CanonicalBuilding.**audit_logs**
Accessor to the related objects manager on the one-to-many relation created by GenericRelation.

In the example:

```
class Post(Model):
    comments = GenericRelation(Comment)
```

`post.comments` is a ReverseGenericManyToOneDescriptor instance.

CanonicalBuilding.**best_guess**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

CanonicalBuilding.**buildingsnapshot_set**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

CanonicalBuilding.**canonical_snapshot**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

CanonicalBuilding.**labels**
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`pizza.toppings` and `topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

CanonicalBuilding.**objects = <seed.models.CanonicalManager object>**

CanonicalBuilding.**raw_objects = <django.db.models.manager.Manager object>**

class seed.models.**CanonicalManager**
Bases: django.db.models.manager.Manager

Manager to add useful model filtering methods

get_queryset()
Return only active CanonicalBuilding rows.

class seed.models.**Column**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

The name of a column for a given organization.

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception Column.**MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

Column.**enum**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

child.parent is a ForwardManyToOneDescriptor instance.

Column.**mapped_mappings**
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

pizza.toppings and topping.pizzas are ManyToManyDescriptor instances.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

Column.**objects = <django.db.models.manager.Manager object>**

Column.**organization**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

child.parent is a ForwardManyToOneDescriptor instance.

Column.**raw_mappings**
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

pizza.toppings and topping.pizzas are ManyToManyDescriptor instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

`Column.`**`unit`**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

class `seed.models.`**`ColumnMapping`**(**args*, ***kwargs*)
Bases: `django.db.models.base.Model`

Stores previous user-defined column mapping.

We'll pull from this when pulling from varied, dynamic source data to present the user with previous choices for that same field in subsequent data loads.

exception `DoesNotExist`
Bases: `django.core.exceptions.ObjectDoesNotExist`

exception `ColumnMapping.`**`MultipleObjectsReturned`**
Bases: `django.core.exceptions.MultipleObjectsReturned`

`ColumnMapping.`**`column_mapped`**
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`pizza.toppings` and `topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

`ColumnMapping.`**`column_raw`**
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`pizza.toppings` and `topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

`ColumnMapping.`**`get_source_type_display`**(**moreargs*, ***morekwargs*)

`ColumnMapping.`**`is_concatenated`**()
Returns True if the ColumnMapping represents the concatenation of imported column names; else returns False.

`ColumnMapping.`**`is_direct`**()
Returns True if the ColumnMapping is a direct mapping from imported column name to either a BEDES column or a previously imported column. Returns False if the ColumnMapping represents a concatenation.

`ColumnMapping.`**`objects = <django.db.models.manager.Manager object>`**

ColumnMapping.**remove_duplicates**(*qs*, *m2m_type='column_raw'*)
Remove any other Column Mappings that use these columns.

Parameters

- **qs** – queryset of Column. These are the Columns in a M2M with this instance.
- **m2m_type** – str, the name of the field we're comparing against. Defaults to 'column_raw'.

ColumnMapping.**save**(**args*, ***kwargs*)
Overrides default model save to eliminate duplicate mappings.

> **Warning:** Other column mappings which have the same raw_columns in them will be removed!

ColumnMapping.**super_organization**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

child.parent is a ForwardManyToOneDescriptor instance.

ColumnMapping.**user**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

child.parent is a ForwardManyToOneDescriptor instance.

class seed.models.**Compliance**(*id*, *created*, *modified*, *compliance_type*, *start_date*, *end_date*, *deadline_date*, *project*)
Bases: django_extensions.db.models.TimeStampedModel

AUDITING_COMPLIANCE_CHOICE = 'Auditing'

BENCHMARK_COMPLIANCE_CHOICE = 'Benchmarking'

COMPLIANCE_CHOICES = (('Benchmarking', <django.utils.functional.__proxy__ object at 0x7f18fb7a6410>), ('Auditing

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception Compliance.**MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

Compliance.**RETRO_COMMISSIONING_COMPLIANCE_CHOICE = 'Retro Commissioning'**

Compliance.**get_compliance_type_display**(**moreargs*, ***morekwargs*)

Compliance.**get_next_by_created**(**moreargs*, ***morekwargs*)

Compliance.**get_next_by_modified**(**moreargs*, ***morekwargs*)

Compliance.**get_previous_by_created**(**moreargs*, ***morekwargs*)

Compliance.**get_previous_by_modified**(**moreargs*, ***morekwargs*)

Compliance.**objects = <django.db.models.manager.Manager object>**

Compliance.**project**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

Compliance.**to_dict**()

class seed.models.**CustomBuildingHeaders**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Specify custom building header mapping for display.

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception CustomBuildingHeaders.**MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

CustomBuildingHeaders.**building_headers**
A placeholder class that provides a way to set the attribute on the model.

CustomBuildingHeaders.**objects = <seed.managers.json.JsonManager object>**

CustomBuildingHeaders.**super_organization**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

class seed.models.**Enum**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Defines a set of enumerated types for a column.

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception Enum.**MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

Enum.**column_set**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

Enum.**enum_values**
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`pizza.toppings` and `topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

`Enum.objects = <django.db.models.manager.Manager object>`

class `seed.models.`**`EnumValue`**(**args*, ***kwargs*)
Bases: `django.db.models.base.Model`

Individual Enumerated Type values.

exception `DoesNotExist`
Bases: `django.core.exceptions.ObjectDoesNotExist`

exception `EnumValue.`**`MultipleObjectsReturned`**
Bases: `django.core.exceptions.MultipleObjectsReturned`

`EnumValue.`**`objects = <django.db.models.manager.Manager object>`**

`EnumValue.`**`values`**
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`pizza.toppings` and `topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

class `seed.models.`**`Meter`**(**args*, ***kwargs*)
Bases: `django.db.models.base.Model`

Meter specific attributes.

exception `DoesNotExist`
Bases: `django.core.exceptions.ObjectDoesNotExist`

exception `Meter.`**`MultipleObjectsReturned`**
Bases: `django.core.exceptions.MultipleObjectsReturned`

`Meter.`**`building_snapshot`**
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`pizza.toppings` and `topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

`Meter.`**`get_energy_type_display`**(**moreargs*, ***morekwargs*)

`Meter.`**`get_energy_units_display`**(**moreargs*, ***morekwargs*)

Meter.**objects = <django.db.models.manager.Manager object>**

Meter.**timeseries_data**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

class seed.models.**NonCanonicalProjectBuildings**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Holds a reference to all project buildings that do not point at a canonical building snapshot.

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception NonCanonicalProjectBuildings.**MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

NonCanonicalProjectBuildings.**objects = <django.db.models.manager.Manager object>**

NonCanonicalProjectBuildings.**projectbuilding**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

class seed.models.**Project**(*id*, *created*, *modified*, *name*, *slug*, *owner*, *last_modified_by*, *super_organization*, *description*, *status*)
Bases: django_extensions.db.models.TimeStampedModel

ACTIVE_STATUS = 1

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

Project.**INACTIVE_STATUS = 0**

exception Project.**MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

Project.**STATUS_CHOICES = ((0, <django.utils.functional.__proxy__ object at 0x7f18fb80d3d0>), (1, <django.utils.fun**

Project.**adding_buildings_status_percentage_cache_key**

Project.**building_snapshots**
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`pizza.toppings` and `topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

`Project.`**`compliance_set`**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

`Project.`**`get_compliance`**`()`

`Project.`**`get_next_by_created`**`(*moreargs, **morekwargs)`

`Project.`**`get_next_by_modified`**`(*moreargs, **morekwargs)`

`Project.`**`get_previous_by_created`**`(*moreargs, **morekwargs)`

`Project.`**`get_previous_by_modified`**`(*moreargs, **morekwargs)`

`Project.`**`get_status_display`**`(*moreargs, **morekwargs)`

`Project.`**`has_compliance`**

`Project.`**`last_modified_by`**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

`Project.`**`objects = <django.db.models.manager.Manager object>`**

`Project.`**`owner`**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

`Project.`**`project_building_snapshots`**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

Project.**removing_buildings_status_percentage_cache_key**

Project.**super_organization**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

child.parent is a ForwardManyToOneDescriptor instance.

Project.**to_dict**()

class seed.models.**ProjectBuilding**(*id*, *created*, *modified*, *building_snapshot*, *project*, *compliant*, *approved_date*, *approver*)
Bases: django_extensions.db.models.TimeStampedModel

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception ProjectBuilding.**MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

ProjectBuilding.**approver**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

child.parent is a ForwardManyToOneDescriptor instance.

ProjectBuilding.**building_snapshot**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

child.parent is a ForwardManyToOneDescriptor instance.

ProjectBuilding.**get_next_by_created**(**moreargs*, ***morekwargs*)

ProjectBuilding.**get_next_by_modified**(**moreargs*, ***morekwargs*)

ProjectBuilding.**get_previous_by_created**(**moreargs*, ***morekwargs*)

ProjectBuilding.**get_previous_by_modified**(**moreargs*, ***morekwargs*)

ProjectBuilding.**noncanonicalprojectbuildings_set**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

parent.children is a ReverseManyToOneDescriptor instance.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

ProjectBuilding.**objects = <django.db.models.manager.Manager object>**

ProjectBuilding.**project**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

ProjectBuilding.**to_dict**()

class seed.models.**StatusLabel**(*id*, *created*, *modified*, *name*, *color*, *super_organization*)
Bases: django_extensions.db.models.TimeStampedModel

BLUE_CHOICE = 'blue'

COLOR_CHOICES = (('red', <django.utils.functional.__proxy__ object at 0x7f18fb81b990>), ('blue', <django.utils.functio

DEFAULT_LABELS = ['Residential', 'Non-Residential', 'Violation', 'Compliant', 'Missing Data', 'Questionable Report', '

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

StatusLabel.**GRAY_CHOICE = 'gray'**

StatusLabel.**GREEN_CHOICE = 'green'**

StatusLabel.**LIGHT_BLUE_CHOICE = 'light blue'**

exception StatusLabel.**MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

StatusLabel.**ORANGE_CHOICE = 'orange'**

StatusLabel.**RED_CHOICE = 'red'**

StatusLabel.**WHITE_CHOICE = 'white'**

StatusLabel.**canonicalbuilding_set**
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`pizza.toppings` and `topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

StatusLabel.**get_color_display**(**moreargs*, ***morekwargs*)

StatusLabel.**get_next_by_created**(**moreargs*, ***morekwargs*)

StatusLabel.**get_next_by_modified**(**moreargs*, ***morekwargs*)

StatusLabel.**get_previous_by_created**(**moreargs*, ***morekwargs*)

StatusLabel.**get_previous_by_modified**(**moreargs*, ***morekwargs*)

StatusLabel.**objects = <django.db.models.manager.Manager object>**

StatusLabel.**super_organization**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

child.parent is a ForwardManyToOneDescriptor instance.

StatusLabel.**to_dict**()

class seed.models.**TimeSeries**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

For storing energy use over time.

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception TimeSeries.**MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

TimeSeries.**meter**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

child.parent is a ForwardManyToOneDescriptor instance.

TimeSeries.**objects = <django.db.models.manager.Manager object>**

class seed.models.**Unit**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Unit of measure for a Column Value.

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception Unit.**MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

Unit.**column_set**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

parent.children is a ReverseManyToOneDescriptor instance.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

Unit.**get_unit_type_display**(**moreargs*, ***morekwargs*)

Unit.**objects = <django.db.models.manager.Manager object>**

seed.models.**clean_canonicals**(*b1*, *b2*, *new_snapshot*)
Make sure that we don't leave dead limbs in our tree.

Parameters

- **b1** – BuildingSnapshot, parent 1
- **b2** – BuildingSnapshot, parent 2

- **new_snapshot** – BuildingSnapshot, child.

seed.models.**find_canonical_building_values**(*org*)
Get all canonical building snapshots' id info for an organization.

Parameters **org** – Organization inst.

Return type list of tuples, field values specified in BS_VALUES_LIST for all canonical buildings related to an organization.

NB: This does not return a queryset!

seed.models.**find_unmatched_buildings**(*import_file*)
Get unmatched building snapshots' id info from an import file.

Parameters **import_file** – ImportFile inst.

Return type list of tuples, field values specified in BS_VALUES_LIST.

NB: This does not return a queryset!

seed.models.**get_ancestors**(*building*)
gets all the non-raw, non-composite ancestors of a building

Recursive function to traverse the tree upward.

Parameters **building** – BuildingSnapshot inst.

Returns list of BuildingSnapshot inst., ancestors of building

```
    source_type {
        2: ASSESSED_BS,
        3: PORTFOLIO_BS,
        4: COMPOSITE_BS,
        6: GREEN_BUTTON_BS
    }
```

seed.models.**get_column_mapping**(*column_raw*, *organization*, *attr_name='column_mapped'*)
Callable provided to MCM to return a previously mapped field.

Parameters

- **column_raw** – str, the column name of the raw data.
- **organization** – Organization inst.
- **attr_name** – str, name of attribute on ColumnMapping to pull out. whether we're looking at a mapping from the perspective of a raw_column (like we do when creating a mapping), or mapped_column, (like when we're applying that mapping).

Returns list of mapped items, float representation of confidence.

seed.models.**get_column_mappings**(*organization*)
Returns dict of all the column mappings for an Organization's given source type

Parameters **organization** – inst, Organization.

Returns dict, list of dict.

Use this when actually performing mapping between data sources, but only call it after all of the mappings have been saved to the `ColumnMapping` table.

seed.models.**get_or_create_canonical**(*b1*, *b2=None*)
Gets most trusted Canonical Building.

Parameters

- **b1** – BuildingSnapshot model type.
- **b2** – BuildingSnapshot model type.

Return type CanonicalBuilding inst. Will contain PK.

NB: preference is given to existing snapshots' Canonical link.

seed.models.**get_sourced_attributes**(*snapshot*)
Return all the attribute names that get sourced.

seed.models.**initialize_canonical_building**(*snapshot*, *user_pk*)
Called to create a CanonicalBuilding from a single snapshot.

Parameters

- **snapshot** – BuildingSnapshot inst.
- **user_pk** – The user id of the user initiating the CanonicalBuilding

seed.models.**obj_to_dict**(*obj*, *include_m2m=True*)
serializes obj for a JSON friendly version tries to serialize JsonField

seed.models.**save_column_names**(*bs*, *mapping=None*)
Save unique column names for extra_data in this organization.

Basically this is a record of all the extra_data keys we've ever seen for a particular organization.

Parameters **bs** – BuildingSnapshot instance.

seed.models.**save_snapshot_match**(*b1_pk*, *b2_pk*, *confidence=None*, *user=None*, *match_type=None*, *default_pk=None*)
Saves a match between two models as a new snapshot; updates Canonical.

Parameters

- **b1_pk** – int, id for building snapshot.
- **b2_pk** – int, id for building snapshot.
- **confidence** – (optional) float, likelihood that two models are linked.
- **user** – (optional) User inst, last_modified_by for BuildingSnapshot.

Return type BuildingSnapshot instance, post save.

Determines which Canonical link should be used. If `canonical` is specified, we're probably changing a building's Canonical link, so use that Canonical Building. Otherwise, use the model we match against. If none exists, create it.

Update mapped fields in the new snapshot, update canonical links.

seed.models.**set_initial_sources**(*snapshot*)
Sets the PK for the original sources to self.

seed.models.**unmatch_snapshot_tree**(*building_pk*)
May or may not obviate `unmatch_snapshot`. Experimental.

Parameters **building_pk** – int - Primary Key for a BuildingSnapshot.

Warning: `unmatch_snapshot_tree` potentially modifies *years* of merged data. Anything descended from the `building_pk` will be deleted. The intent is to completely separate `building_pk`'s influence on the resultant canonical_snapshot. The user is saying that these are separate entities after all, yes?

Basically, this function works by getting a merge order list of children from the perspective of `building_pk` and a list of parents from the perspective of leaf node in the child tree. We take the difference between these

lists and call that the `remaining_ancestors` from which we reconstruct the merge tree for our CanonicalBuilding.

`building_pk` either gets a reactivated CanonicalBuilding, or a new one.

`seed.models.`**`update_building`**(*old_snapshot*, *updated_values*, *user*, **args*, ***kwargs*)
Creates a new snapshot with updated values.

6.14.7 Reconcile

6.14.8 Search

Search methods pertaining to buildings.

`seed.search.`**`build_json_params`**(*order_by*, *sort_reverse*)
returns db_columns, extra_data_sort, and updated order_by

Parameters **`order_by`** (*str*) – field to order_by

Returns tuple: db_columns: dict of known DB columns i.e. non-JsonField, extra_data_sort bool if order_by is in `extra_data` JsonField, order_by str if sort_reverse and DB column prepend a '-' for the django order_by

`seed.search.`**`build_shared_buildings_orgs`**(*orgs*)
returns a list of sibling and parent orgs

`seed.search.`**`convert_to_js_timestamp`**(*timestamp*)
converts a django/python datetime object to milliseconds since epoch duplicated code with seed utils due to circular imports

`seed.search.`**`create_building_queryset`**(*orgs*, *exclude*, *order_by*, *other_orgs=None*, *extra_data_sort=False*)
creates a queryset of buildings within orgs. If `other_orgs`, buildings in both orgs and other_orgs will be represented in the queryset.

Parameters

- **`orgs`** – queryset of Organization inst.
- **`exclude`** – django query exclude dict.
- **`order_by`** – django query order_by str.
- **`other_orgs`** – list of other orgs to `or` the query

`seed.search.`**`filter_other_params`**(*queryset*, *other_params*, *db_columns*)
applies a dictionary filter to the query set. Does some domain specific parsing, mostly to remove extra query params and deal with ranges. Ranges should be passed in as '<field name>__lte' or '<field name>__gte' e.g. other_params = {'gross_floor_area__lte': 50000}

Parameters

- **`Queryset queryset`** (*Django*) – queryset to be filtered
- **`other_params`** (*dict*) – dictionary to be parsed and applied to filter.
- **`db_columns`** (*dict*) – list of column names, extra_data blob outside these

Returns Django Queryset:

seed.search.**generate_paginated_results**(*queryset*, *number_per_page=25*, *page=1*, *whitelist_orgs=None*, *below_threshold=False*, *matching=True*)

Return a page of results as a list from the queryset for the given fields

Parameters

- **queryset** – optional queryset to filter from
- **number_per_page** (*int*) – optional number of results per page
- **page** (*int*) – optional page of results to get
- **whitelist_orgs** – a queryset returning the organizations in which all building fields can be returned, otherwise only the parent organization's `exportable_fields` should be returned. The `whitelist_orgs` are the orgs the request user belongs.
- **below_threshold** – True if less than the parent org's query threshold is greater than the number of queryset results. If True, only return buildings within whitelist_orgs.
- **matching** – Toggle expanded parent and children data, including coparent and confidence

Usage:

```
generate_paginated_results(q, 1)
```

Returns:

```
[
    {
        'gross_floor_area': 1710,
        'site_eui': 123,
        'tax_lot_id': 'a-tax-lot-id',
        'year_built': 2001
    }
]
```

seed.search.**get_building_fieldnames**()

returns a list of field names for the BuildingSnapshot class/model that will be searched against

seed.search.**get_orgs_w_public_fields**()

returns a list of orgs that have publicly shared fields

seed.search.**is_not_whitelist_building**(*parent_org*, *building*, *whitelist_orgs*)

returns false if a building is part of the whitelist_orgs

Parameters

- **parent_org** – the umbrella parent Organization instance.
- **building** – the BuildingSnapshot inst.
- **whitelist_orgs** – queryset of Organization instances.

Returns bool

seed.search.**mask_results**(*search_results*)

masks (deletes dict keys) for non-shared public fields

seed.search.**orchestrate_search_filter_sort**(*params*, *user*, *skip_sort=False*)

Given a parsed set of params, perform the search, filter, and sort for BuildingSnapshot's

seed.search.**paginate_results**(*request*, *search_results*)

returns a paginated list of dict results

seed.search.**parse_body**(*request*)
parses the request body for search params, q, etc

Parameters **request** – django wsgi request object

Returns dict

Example:

```
{
    'exclude': dict, exclude dict for django queryset
    'order_by': str, query order_by, defaults to 'tax_lot_id'
    'sort_reverse': bool, True if ASC, False if DSC
    'page': int, pagination page
    'number_per_page': int, number per pagination page
    'show_shared_buildings': bool, whether to search across all user's orgs
    'q': str, global search param
    'other_search_params': dict, filter params
    'project_id': str, project id if exists in body
}
```

seed.search.**process_search_params**(*params*, *user*, *is_api_request=False*)
Given a python representation of a search query, process it into the internal format that is used for searching, filtering, sorting, and pagination.

Parameters

- **params** – a python object representing the search query
- **user** – the user this search is for
- **is_api_request** – bool, boolean whether this search is being done as an api request.

Returns dict

Example:

```
{
    'exclude': dict, exclude dict for django queryset
    'order_by': str, query order_by, defaults to 'tax_lot_id'
    'sort_reverse': bool, True if ASC, False if DSC
    'page': int, pagination page
    'number_per_page': int, number per pagination page
    'show_shared_buildings': bool, whether to search across all user's orgs
    'q': str, global search param
    'other_search_params': dict, filter params
    'project_id': str, project id if exists in body
}
```

seed.search.**remove_results_below_q_threshold**(*search_results*)
removes buildings if total count of buildings grouped by org is less than their org's public query threshold

Parameters **search_results** (*list/queryset*) – search results

Returns list or queryset

seed.search.**search_buildings**(*q*, *fieldnames=None*, *queryset=None*)
returns a queryset for matching buildings

Parameters

- **or unicode q** (*str*) – search string

- **fieldnames** (*list*) – list of BuildingSnapshot model fieldnames (defaults to those generated by get_building_field_names())
- **queryset** – optional queryset to filter from, defaults to BuildingSnapshot.objects.none()

Returns

queryset queryset of matching buildings

seed.search.**search_public_buildings**(*request*, *orgs*)
returns a queryset or list of buildings matching the search params and count

Parameters

- **request** – wsgi request (Django) for parsing params
- **orgs** – list of Organization instances to search within

Returns tuple (search_results_list, result count)

6.14.9 Tasks

exception seed.tasks.**DuplicateDataError**(*id*)
Bases: exceptions.RuntimeError

seed.tasks.**apply_data_func**(*mappable_columns*)
Returns a function that captures mappable_types in a closure and will add a key to extra data if not in mappable_types else

seed.tasks.**apply_extra_data**(*model*, *key*, *value*)
Function sent to MCM to apply mapped columns into extra_data.

seed.tasks.**cache_first_rows**(*import_file*, *parser*)
Cache headers, and rows 2-6 for validation/viewing.

Parameters

- **import_file** – ImportFile inst.
- **parser** – unicode-csv.Reader instance.

Unfortunately, this is duplicated logic from data_importer, but since data_importer makes many faulty assumptions we need to do it differently.

seed.tasks.**get_cache_increment_value**(*chunk*)

seed.tasks.**get_canonical_id_matches**(*org_id*, *pm_id*, *tax_id*, *custom_id*)
Returns canonical snapshots that match at least one id.

seed.tasks.**get_canonical_snapshots**(*org_id*)
Return all of the BuildingSnapshots that are canonical for an org.

seed.tasks.**handle_id_matches**(*unmatched_bs*, *import_file*, *user_pk*)
"Deals with exact matches in the IDs of buildings.

seed.tasks.**handle_results**(*results*, *b_idx*, *can_rev_idx*, *unmatched_list*, *user_pk*)
Seek IDs and save our snapshot match.

Parameters

- **results** – list of tuples. [('match', 0.99999),...]
- **b_idx** – int, the index of the current building in the unmatched_list.
- **can_rev_idx** – dict, reverse index from match -> canonical PK.

- **user_pk** – user ID, used for AuditLog logging

Unmatched_list list of dicts, the result of a values_list query for unmatched BuildingSnapshots.

seed.tasks.**is_same_snapshot**(*s1*, *s2*)

6.14.10 Token Generator

token_generator.py - taken from django core master branch

needed a token check that would not expire after three days for sending a signup email

class seed.token_generators.**SignupTokenGenerator**

Bases: object

Strategy object used to generate and check tokens for the password reset mechanism.

check_token(*user*, *token*, *token_expires=True*)

Check that a password reset token is correct for a given user.

make_token(*user*)

Returns a token that can be used once to do a password reset for the given user.

6.14.11 URLs

6.14.12 Utils

6.14.13 Views

6.14.14 Module contents

6.15 Serializers Package

6.15.1 Submodules

6.15.2 Serializers

class seed.serializers.celery.**CeleryDatetimeSerializer**(*skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *encoding='utf-8'*, *default=None*)

Bases: json.encoder.JSONEncoder

default(*obj*)

static seed_decoder(*obj*)

static seed_dumps(*obj*)

static seed_loads(*obj*)

6.15.3 Labels

class seed.serializers.labels.**LabelSerializer**(**args*, ***kwargs*)
Bases: rest_framework.serializers.ModelSerializer

class Meta

extra_kwargs = {'super_organization': {'write_only': True}}

fields = ('id', 'name', 'color', 'organization_id', 'super_organization', 'is_applied')

model
alias of StatusLabel

LabelSerializer.**get_is_applied**(*obj*)

class seed.serializers.labels.**UpdateBuildingLabelsSerializer**(**args*, ***kwargs*)
Bases: rest_framework.serializers.Serializer

create(*validated_data*)

6.15.4 Module contents

6.16 URLs Package

6.16.1 Submodules

6.16.2 Accounts

6.16.3 APIs

6.16.4 Main

6.16.5 Projects

6.17 Utilities Package

6.17.1 Submodules

6.17.2 APIs

class seed.utils.api.**APIBypassCSRFMiddleware**
Bases: object

This middleware turns off CSRF protection for API clients.

It must come before CsrfViewMiddleware in settings.MIDDLEWARE_CLASSES.

process_view(*request*, **args*, ***kwargs*)
If this request is an API request, bypass CSRF protection.

seed.utils.api.**api_endpoint**(*fn*)
Decorator function to mark a view as allowed to authenticate via API key.

Decorator must be used before login_required or has_perm to set request.user for those decorators.

seed.utils.api.**clean_api_regex**(*url*)
Given a django-style url regex pattern, strip it down to a human-readable url.

TODO: If pks ever appear in the url, this will need to account for that.

seed.utils.api.**drf_api_endpoint**(*fn*)
Decorator to register a Django Rest Framework view with the list of API endpoints. Marks it with *is_api_endpoint = True* as well as appending it to the global *endpoints* list.

seed.utils.api.**format_api_docstring**(*docstring*)
Cleans up a python method docstring for human consumption.

seed.utils.api.**get_all_urls**(*urllist*, *prefix=''*)
Recursive generator that traverses entire tree of URLs, starting with urllist, yielding a tuple of (url_pattern, view_function) for each one.

seed.utils.api.**get_api_endpoints**()
Examines all views and returns those with is_api_endpoint set to true (done by the @api_endpoint decorator).

..:todo: this isn't particularly expensive now, but if the number of URLs grows a lot, it may be worth caching this somewhere.

seed.utils.api.**get_api_request_user**(*request*)
Determines if this is an API request and returns the corresponding user if so.

6.17.3 Buildings

seed.utils.buildings.**get_buildings_for_user_count**(*user*)
returns the number of buildings in a user's orgs

seed.utils.buildings.**get_columns**(*org_id*, *all_fields=False*)
Get default columns, to be overridden in future

Returns:

```
    title: HTML presented title of column
    sort_column: semantic name used by js and for searching DB
    class: HTML CSS class for row td elements
    title_class: HTML CSS class for column td elements
    type: 'string', 'number', 'date'
    min, max: the django filter key e.g. gross_floor_area__gte
    field_type: assessor, pm, or compliance (currently not used)
    sortable: determines if the column is sortable
    checked: initial state of "edit columns" modal
    static: True if option can be toggle (ID is false because it is
        always needed to link to the building detail page)
    link: signifies that the cell's data should link to a building detail
        page
```

seed.utils.buildings.**get_search_query**(*user*, *params*)

seed.utils.buildings.**get_source_type**(*import_file*, *source_type=''*)
Used for converting ImportFile source_type into an int.

seed.utils.buildings.**serialize_building_snapshot**(*b*, *pm_cb*, *building*)
returns a dict that's safe to JSON serialize

6.17.4 Constants

6.17.5 Mappings

seed.utils.mapping.**get_mappable_columns**(*exclude_fields=None*)
Get a list of all the columns we're able to map to.

seed.utils.mapping.**get_mappable_types**(*exclude_fields=None*)
Like get_mappable_columns, but with type information.

6.17.6 Organizations

seed.utils.organizations.**create_organization**(*user*, *org_name=''*, **args*, ***kwargs*)
Helper script to create a user/org relationship from scratch.

Parameters

- **user** – user inst.
- **org_name** – str, name of Organization we'd like to create.
- **kwargs** (*(optional)*) – 'role', int; 'status', str.

6.17.7 Projects

seed.utils.projects.**copy_buildings**(*source_project*, *target_project*, *buildings*, *select_all*, *search_params*, *user*)
copies buildings from source project to target project

Parameters

- **source_project_slug** – str, a slug to get a Project inst.
- **target_project_slug** – str, a slug to get a Project inst.
- **buildings** – list, list of source_facility_id as str to get BuildingSnapshot inst.
- **select_all** – bool, if the select all checkbox was checked. i.e. should we transfer all buildings in a project or just the buildings in the list

Search_params dict, params needed to generate a queryset of buildings, with keys (q, other_params, project_slug)

User User inst., django user instance needed for select all queryset

seed.utils.projects.**delete_matching_buildings**(*project*, *buildings*, *select_all*, *search_params*, *user*)
deletes buildings in a project that match search search params

Parameters

- **project_slug** – str, a slug to get a Project inst.
- **buildings** – list, list of source_facility_id as str to get BuildingSnapshot inst.
- **select_all** – bool, if the select all checkbox was checked. i.e. should we transfer all buildings in a project or just the buildings in the list

Search_params dict, params needed to generate a queryset of buildings, with keys (q, other_params, project_slug)

User User inst., django user instance needed for select all queryset

seed.utils.projects.**get_projects**(*building*, *organization*)
return an JSON friendly list of the building's projects

Parameters

- **building** – the BuildingSnapshot inst.
- **organization** – the Organization inst.

Returns list of projects

seed.utils.projects.**get_transfer_buildings**(*source_project*, *target_project*, *buildings*, *select_all*, *search_params*, *user*)
generates move or copy buildings queryset

Parameters

- **source_project_slug** – str, a slug to get a Project inst.
- **target_project_slug** – str, a slug to get a Project inst.
- **buildings** – list, list of source_facility_id as str to get BuildingSnapshot inst.
- **select_all** – bool, if the select all checkbox was checked. i.e. should we transfer all buildings in a project or just the buildings in the list

Search_params dict, params needed to generate a queryset of buildings, with keys (q, other_params, project_slug)

User User inst., django user instance needed for select all queryset

Rtype Queryset a django queryset of buildings to move or copy

seed.utils.projects.**move_buildings**(*source_project*, *target_project*, *buildings*, *select_all*, *search_params*, *user*)
moves buildings from source project to target project

Parameters

- **source_project_slug** – str, a slug to get a Project inst.
- **target_project_slug** – str, a slug to get a Project inst.
- **buildings** – list, list of source_facility_id as str to get BuildingSnapshot inst.
- **select_all** – bool, if the select all checkbox was checked. i.e. should we transfer all buildings in a project or just the buildings in the list

Search_params dict, params needed to generate a queryset of buildings, with keys (q, other_params, project_slug)

User User inst., django user instance needed for select all queryset

seed.utils.projects.**transfer_buildings**(*source_project_slug*, *target_project_slug*, *buildings*, *select_all*, *search_params*, *user*, *copy_flag=False*)
copies or moves buildings from one project to another

Parameters

- **source_project_slug** – str, a slug to get a Project inst.
- **target_project_slug** – str, a slug to get a Project inst.
- **buildings** – list, list of source_facility_id as str to get BuildingSnapshot inst.
- **select_all** – bool, if the select all checkbox was checked. i.e. should we transfer all buildings in a project or just the buildings in the list

Search_params dict, params needed to generate a queryset of buildings, with keys (q, other_params, project_slug)

User User inst., django user instance needed for select all queryset and to update the projects' "last changed" and "last changed by"

Copy_flag bool, True - copy buildings, False - move buildings

6.17.8 Time

seed.utils.time.**convert_datestr**(*datestr*)
Converts dates like *12/31/2010* into datetime objects.

seed.utils.time.**convert_to_js_timestamp**(*timestamp*)
converts a django/python datetime object to milliseconds since epoch

seed.utils.time.**parse_datetime**(*maybe_datetime*)
Process a datetime value that may be None, timestamp, strftime.

6.18 Views Package

6.18.1 Submodules

6.18.2 Accounts

seed.views.accounts.**add_org**(*request*, **args*, ***kwargs*)
Creates a new organization.

Payload:

```
{
    'organization_name': The name of the new org,
    'user_id': the user id of the owner of the new org,
}
```

Returns:

```
{
    'status': 'success' or 'error',
    'message': 'message, if any',
    'organization_id': The ID of the new org, if created.
}
```

seed.views.accounts.**add_user**(*request*, **args*, ***kwargs*)
Creates a new SEED user. One of 'organization_id' or 'org_name' is needed. Sends invitation email to the new user.

Payload:

```
{
    'organization_id': ID of an existing org to add the new user to,
    'org_name': Name of a new org to create with user as owner
    'first_name': First name of new user
    'last_name': Last name of new user
    'role': {
        'value': The permission level of new user within this org
            (one of member, viewer, owner)
```

```
        },
        'email': Email address of new user.
    }
```

Returns:

```
    {
        'status': 'success',
        'message': email address of new user,
        'org': name of the new org (or existing org),
        'org_created': True if new org created,
        'username': Username of new user
    }
```

seed.views.accounts.**add_user_to_organization**(*request*, **args*, ***kwargs*)

Adds an existing user to an organization.

Payload:

```
    {
        'organization_id': The ID of the organization,
        'user_id': the user id of the owner of the new org,
    }
```

Returns:

```
    {
        'status': 'success' or 'error',
        'message': 'message, if any',
    }
```

seed.views.accounts.**create_sub_org**(*request*, **args*, ***kwargs*)

Creates a child org of a parent org.

Payload:

```
    {
        'parent_org_id': ID of the parent org,
        'sub_org': {
            'name': Name of new sub org,
            'email': Email address of owner of sub org, which
                    must already exist
        }
    }
```

Returns:

```
    {
        'status': 'success' or 'error',
        'message': Error message, if any,
        'organization_id': ID of newly-created org
    }
```

seed.views.accounts.**generate_api_key**(*request*, **args*, ***kwargs*)

generates a new API key

Returns:

```
    {
        'status': 'success',
```

```
        'api_key': the new api key
    }
```

seed.views.accounts.**get_actions**(*request*, **args*, ***kwargs*)
returns all actions

seed.views.accounts.**get_cleansing_rules**(*request*, **args*, ***kwargs*)
Returns the cleansing rules for an org.

Parameters **request** –

GET Expects organization_id in the query string.

Returns:

```
    {
     'status': 'success',
     'in_range_checking': An array of in-range error rules,
     'missing_matching_field': An array of fields to verify existence,
     'missing_values': An array of fields to ignore missing values
    }
```

seed.views.accounts.**get_organization**(*request*, **args*, ***kwargs*)
Retrieves a single organization by id.

GET Expects ?organization_id=(:org_id)

Returns:

```
    {'status': 'success or error', 'message': 'error message, if any',
     'organization':
        {'name': org name,
         'org_id': org's identifier (used with Authorization header),
         'id': org's identifier,
         'number_of_users': count of members of org,
         'user_is_owner': True if the user is owner of this org,
         'user_role': The role of user in this org (owner, viewer, member),
         'owners': [
             {
              'first_name': the owner's first name,
              'last_name': the owner's last name,
              'email': the owner's email address,
              'id': the owner's identifier (int)
              }
             ]
          'sub_orgs': [ a list of orgs having this org as parent, in
                        the same format...],
          'is_parent': True if this org contains suborgs,
          'num_buildings': Count of buildings belonging to this org
        }
    }
```

seed.views.accounts.**get_organizations**(*request*, **args*, ***kwargs*)
Retrieves all orgs the user has access to.

Returns:

```
    {'organizations': [
        {'name': org name,
         'org_id': org's identifier (used with Authorization header),
         'id': org's identifier,
```

```
        'number_of_users': count of members of org,
        'user_is_owner': True if the user is owner of this org,
        'user_role': The role of user in this org (owner, viewer, member),
        'owners': [
                    {
                     'first_name': the owner's first name,
                     'last_name': the owner's last name,
                     'email': the owner's email address,
                     'id': the owner's identifier (int)
                    }
                  ]
        'sub_orgs': [ a list of orgs having this org as parent, in
                    the same format...],
        'is_parent': True if this org contains suborgs,
        'num_buildings': Count of buildings belonging to this org
       }...
      ]
    }
```

seed.views.accounts.**get_organizations_users**(*request*, **args*, ***kwargs*)

Retrieve all users belonging to an org.

Payload:

```
{'organization_id': org_id}
```

Returns:

```
{'status': 'success',
 'users': [
    {
     'first_name': the user's first name,
     'last_name': the user's last name,
     'email': the user's email address,
     'id': the user's identifier (int),
     'role': the user's role ('owner', 'member', 'viewer')
    }
  ]
}
```

Todo

check permissions that request.user is owner or admin and get more info about the users.

seed.views.accounts.**get_query_threshold**(*request*, **args*, ***kwargs*)

Returns the "query_threshold" for an org. Searches from members of sibling orgs must return at least this many buildings from orgs they do not belong to, or else buildings from orgs they don't belong to will be removed from the results.

GET Expects organization_id in the query string.

Returns:

```
{
 'status': 'success',
 'query_threshold': The minimum number of buildings that must be
     returned from a search to avoid squelching non-member-org results.
}
```

seed.views.accounts.**get_shared_buildings**(*request*, **args*, ***kwargs*)
gets the request user's show_shared_buildings attr

seed.views.accounts.**get_shared_fields**(*request*, **args*, ***kwargs*)
Retrieves all fields marked as shared for this org tree.

GET Expects organization_id in the query string.

Returns:

```
{
 'status': 'success',
 'shared_fields': [
     {
      "title": Display name of field,
      "sort_column": database/search name of field,
      "class": css used for field,
      "title_class": css used for title,
      "type": data type of field,
          (One of: 'date', 'floor_area', 'link', 'string', 'number')
      "field_type": classification of field.  One of:
          'contact_information', 'building_information',
          'assessor', 'pm',
      "sortable": True if buildings can be sorted on this field,
     }
     ...
   ],
   'public_fields': [
       {
          "title": Display name of field,
          "sort_column": database/search name of field,
          "class": css used for field,
          "title_class": css used for title,
          "type": data type of field,
            (One of: 'date', 'floor_area', 'link', 'string', 'number')
          "field_type": classification of field.  One of:
              'contact_information', 'building_information',
              'assessor', 'pm',
          "sortable": True if buildings can be sorted on this field,
         }
         ...
   ]
}
```

seed.views.accounts.**get_user_profile**(*request*, **args*, ***kwargs*)
Retrieves the request's user's first_name, last_name, email and api key if exists.

Returns:

```
{
    'status': 'success',
    'user': {
        'first_name': user's first name,
        'last_name': user's last name,
        'email': user's email,
        'api_key': user's API key
    }
}
```

seed.views.accounts.**get_users**(*request*, **args*, ***kwargs*)
Retrieves all users' email addresses and IDs. Only usable by superusers.

Returns:

```
{
    'users': [
        'email': 'Email address of user',
        'user_id': 'ID of user'
    ] ...
}
```

seed.views.accounts.**is_authorized**(*request*, **args*, ***kwargs*)
checks the auth for a given action, if user is the owner of the parent org then True is returned for each action

Payload:

```
{
    'organization_id': 2,
    'actions': ['can_invite_member', 'can_remove_member']
}
```

Parameters **actions** – from the json payload, a list of actions to check

Returns a dict of with keys equal to the actions, and values as bool

seed.views.accounts.**remove_user_from_org**(*request*, **args*, ***kwargs*)
Removes a user from an organization.

Payload:

```
{
    'organization_id': ID of the org,
    'user_id': ID of the user
}
```

Returns:

```
{
    'status': 'success' or 'error',
    'message': 'error message, if any'
}
```

seed.views.accounts.**reset_cleansing_rules**(*request*, **args*, ***kwargs*)
Resets an organization's data cleansing rules

Parameters **request** –

GET Expects organization_id in the query string.

Returns:

```
{
 'status': 'success',
 'in_range_checking': An array of in-range error rules,
 'missing_matching_field': An array of fields to verify existence,
 'missing_values': An array of fields to ignore missing values
}
```

seed.views.accounts.**save_cleansing_rules**(*request*, **args*, ***kwargs*)
Saves an organization's settings: name, query threshold, shared fields

Payload:

```
    {
        'organization_id: 2,
        'cleansing_rules': {
            'missing_matching_field': [
                {
                    'field': 'address_line_1',
                    'severity': 'error'
                }
            ],
            'missing_values': [
                {
                    'field': 'address_line_1',
                    'severity': 'error'
                }
            ],
            'in_range_checking': [
                {
                    'field': 'conditioned_floor_area',
                    'enabled': true,
                    'type': 'number',
                    'min': null,
                    'max': 7000000,
                    'severity': 'error',
                    'units': 'square feet'
                },
            ]
        }
    }
```

Returns:

```
    {
        'status': 'success',
    }
```

seed.views.accounts.**save_org_settings**(*request*, **args*, ***kwargs*)

Saves an organization's settings: name, query threshold, shared fields

Payload:

```
    {
        'organization_id: 2,
        'organization': {
            'query_threshold': 2,
            'name': 'demo org',
            'fields': [  # All internal sibling org shared fields
                {
                    'sort_column': database/search field name,
                        e.g. 'pm_property_id',
                }
            ],
            'public_fields': [  # All publicly shared fields
                {
                    'sort_column': database/search field name,
                        e.g. 'pm_property_id',
                }
            ],
        }
    }
```

Returns:

```
{
    'status': 'success or error',
    'message': 'error message, if any'
}
```

seed.views.accounts.**search_public_fields**(*request*, **args*, ***kwargs*)

Search across all public fields.

Payload:

```
{
     'q': a string to search on (optional),
     'show_shared_buildings': True to include buildings from other
         orgs in this user's org tree,
     'order_by': which field to order by (e.g. pm_property_id),
     'import_file_id': ID of an import to limit search to,
     'filter_params': { a hash of Django-like filter parameters to limit
         query.  See seed.search.filter_other_params.  If 'project__slug'
         is included and set to a project's slug, buildings will include
         associated labels for that project.
       }
     'page': Which page of results to retrieve (default: 1),
     'number_per_page': Number of buildings to retrieve per page
                        (default: 10),
}
```

Returns:

```
{
     'status': 'success',
     'buildings': [
      { all fields for buildings the request user has access to;
        e.g.:
       'canonical_building': the CanonicalBuilding ID of the building,
       'pm_property_id': ID of building (from Portfolio Manager),
       'address_line_1': First line of building's address,
       'property_name': Building's name, if any
        ...
       }...
      ]
     'number_matching_search': Total number of buildings matching search,
     'number_returned': Number of buildings returned for this page
}
```

seed.views.accounts.**set_default_organization**(*request*, **args*, ***kwargs*)

sets the user's default organization

seed.views.accounts.**set_password**(*request*, **args*, ***kwargs*)

sets/updates a user's password, follows the min requirement of django password validation settings in config/settings/common.py

Payload:

```
{
    'current_password': current_password,
    'password_1': password_1,
    'password_2': password_2
}
```

Returns:

```
{
    'status': 'success'
}
```

seed.views.accounts.**update_role**(*request*, **args*, ***kwargs*)
Sets a user's role within an organization.

Payload:

```
{
    'organization_id': organization's id,
    'user_id': user's id,
    'role': one of 'owner', 'member', 'viewer'
}
```

Returns:

```
{
    'status': 'success or error',
    'message': 'error message, if any'
}
```

seed.views.accounts.**update_user**(*request*, **args*, ***kwargs*)
Updates the request's user's first name, last name, and email

Payload:

```
{
 'user': {
          'first_name': :first_name,
          'last_name': :last_name,
          'email': :email
        }
}
```

Returns:

```
{
    'status': 'success',
    'user': {
        'first_name': user's first name,
        'last_name': user's last name,
        'email': user's email,
        'api_key': user's API key
    }
}
```

6.18.3 APIs

seed.views.api.**get_api_schema**(*request*, **args*, ***kwargs*)
Returns a hash of all API endpoints and their descriptions.

Returns:

```
{
    '/example/url/here': {
        'name': endpoint name,
        'description': endpoint description
```

```
        }...
    }
```

Todo

Should this require authentication? Should it limit the return to endpoints the user has authorization for?

Todo

Format docstrings better.

6.18.4 Main

seed.views.main.**angular_js_tests**(*request*)
Jasmine JS unit test code covering AngularJS unit tests and ran by ./manage.py harvest

seed.views.main.**create_dataset**(*request*, **args*, ***kwargs*)
Creates a new empty dataset (ImportRecord).

Payload:

```
    {
        "name": Name of new dataset, e.g. "2013 city compliance dataset"
        "organization_id": ID of the org this dataset belongs to
    }
```

Returns:

```
    {
        'status': 'success',
        'id': The ID of the newly-created ImportRecord,
        'name': The name of the newly-created ImportRecord
    }
```

seed.views.main.**create_pm_mapping**(*request*, **args*, ***kwargs*)
Create a mapping for PortfolioManager input columns.

Payload:

```
    {
        columns: [ "name1", "name2", ... , "nameN"],
    }
```

Returns:

```
    {
        success: true,
        mapping: [
            [
                "name1",
                "mapped1", {
                    bedes: true|false,
                    numeric: true|false
                }
            ],
            [
```

```
                "name2",
                "mapped2", {
                    bedes: true|false,
                    numeric: true|false
                }
            ],
            ...
            [
                "nameN",
                "mappedN", {
                    bedes: true|false,
                    numeric: true|false
                }
            ]
        ]
    }
    -- OR --
    {
        success: false,
        reason: "message goes here"
    }
```

seed.views.main.**delete_buildings**(*request*, **args*, ***kwargs*)

Deletes all BuildingSnapshots the user has selected.

Does not delete selected_buildings where the user is not a member or owner of the organization the selected building belongs. Since search shows buildings across all the orgs a user belongs, it's possible for a building to belong to an org outside of *org_id*.

DELETE Expects 'org_id' for the organization, and the search payload similar to add_buildings/create_project

Payload:

```
    {
        'organization_id': 2,
        'search_payload': {
            'selected_buildings': [2, 3, 4],
            'select_all_checkbox': False,
            'filter_params': ... // see search_buildings
        }
    }
```

Returns:

```
    {
        'status': 'success' or 'error'
    }
```

seed.views.main.**delete_dataset**(*request*, **args*, ***kwargs*)

Deletes all files from a dataset and the dataset itself.

DELETE Expects organization id and dataset id.

Payload:

```
    {
        "dataset_id": 1,
        "organization_id": 1
    }
```

Returns:

```
{
    'status': 'success' or 'error',
    'message': 'error message, if any'
}
```

seed.views.main.**delete_duplicates_from_import_file**(*request*, **args*, ***kwargs*)
Retrieves the number of matched and unmatched BuildingSnapshots for a given ImportFile record.

GET Expects import_file_id corresponding to the ImportFile in question.

Returns:

```
{
    "status": "success",
    "deleted": "Number of duplicates deleted"
}
```

seed.views.main.**delete_file**(*request*, **args*, ***kwargs*)
Deletes an ImportFile from a dataset.

Payload:

```
{
    "file_id": "ImportFile id",
    "organization_id": "current user organization id as integer"
}
```

Returns:

```
{
    'status': 'success' or 'error',
    'message': 'error message, if any'
}
```

seed.views.main.**delete_organization**(*request*, **args*, ***kwargs*)
Starts a background task to delete an organization and all related data.

GET Expects 'org_id' for the organization.

Returns:

```
{
    'status': 'success' or 'error',
    'progress_key': ID of background job, for retrieving job progress
}
```

seed.views.main.**delete_organization_buildings**(*request*, **args*, ***kwargs*)
Starts a background task to delete all BuildingSnapshots in an org.

GET Expects 'org_id' for the organization.

Returns:

```
{
    'status': 'success' or 'error',
    'progress_key': ID of background job, for retrieving job progress
}
```

seed.views.main.**export_buildings**(*request*, **args*, ***kwargs*)
Begins a building export process.

Payload:

```
{
    "export_name": "My Export",
    "export_type": "csv",
    "selected_buildings": [1234,], (optional list of building ids)
    "selected_fields": optional list of fields to export
    "select_all_checkbox": True // optional, defaults to False
}
```

Returns:

```
{
    "success": True,
    "status": "success",
    "export_id": export_id; see export_buildings_download,
    "total_buildings": count of buildings,
}
```

seed.views.main.**export_buildings_download**(*request*, **args*, ***kwargs*)

Provides the url to a building export file.

Payload:

```
{
    "export_id": export_id from export_buildings
}
```

Returns:

```
{
    'success': True or False,
    'status': 'success or error',
    'message': 'error message, if any',
    'url': The url to the exported file.
}
```

seed.views.main.**export_buildings_progress**(*request*, **args*, ***kwargs*)

Returns current progress on building export process.

Payload:

```
{
    "export_id": export_id from export_buildings
}
```

Returns:

```
{
    'success': True,
    'status': 'success or error',
    'message': 'error message, if any',
    'buildings_processed': number of buildings exported
}
```

seed.views.main.**get_PM_filter_by_counts**(*request*, **args*, ***kwargs*)

Retrieves the number of matched and unmatched BuildingSnapshots for a given ImportFile record.

GET Expects import_file_id corresponding to the ImportFile in question.

Returns:

```
{
    'status': 'success',
    'matched': Number of BuildingSnapshot objects that have matches,
    'unmatched': Number of BuildingSnapshot objects with no matches.
}
```

seed.views.main.**get_aggregated_building_report_data**(*request*, **args*, ***kwargs*)

This method returns a set of aggregated building data for graphing. It expects as parameters

GET

Parameters

- **start_date** – The starting date for the data series with the format *YYYY-MM-DDThh:mm:ss+hhmm*
- **end_date** – The starting date for the data series with the format *YYYY-MM-DDThh:mm:ss+hhmm*
- **x_var** – The variable name to be assigned to the "x" value in the returned data series
- **y_var** – The variable name to be assigned to the "y" value in the returned data series
- **organization_id** – The organization to be used when querying data.

The x_var values should be from the following set of variable names:

•site_eui

•source_eui

•site_eui_weather_normalized

•source_eui_weather_normalized

•energy_score

The y_var values should be from the following set of variable names:

•gross_floor_area

•use_description

•year_built

This method includes building record count information as part of the result JSON in a property called "building_counts."

This property provides data on the total number of buildings available in each 'year ending' group, as well as the subset of those buildings that have actual data to graph. By sending these values in the result we allow the client to easily build a message like "200 of 250 buildings in this group have data."

Returns:

```
{
    "status": "success",
    "chart_data": [
        {
            "yr_e": x - group by year ending
            "x": x, - median value in group
            "y": y - average value thing
        },
        {
            "yr_e": x
            "x": x,
```

```
                "y": y
            }
            ...
        ],
        "building_counts": [
            {
                "yr_e": string for year ending - group by
                "num_buildings": number of buildings in query results
                "num_buildings_w_data": number of buildings with valid data in this group, BOTH x an
            },
            ...
        ]
        "num_buildings": total number of buildings in query results,
        "num_buildings_w_data": total number of buildings with valid data in query results
    }
```

—

parameters:

- name: x_var description: Name of column in building snapshot database to be used for "x" axis required: true type: string paramType: query
- name: y_var description: Name of column in building snapshot database to be used for "y" axis required: true type: string paramType: query
- start_date: description: The start date for the entire dataset. required: true type: string paramType: query
- end_date: description: The end date for the entire dataset. required: true type: string paramType: query
- name: organization_id description: User's organization which should be used to filter building query results required: true type: string paramType: query

type:

status: required: true type: string

chart_data: required: true type: array

building_counts: required: true type: array

num_buildings: required: true type: string

num_buildings_w_data: required: true type: string

status code:

- code: 400 message: Bad request, only GET method is available
- code: 401 message: Not authenticated
- code: 403 message: Insufficient rights to call this procedure

seed.views.main.**get_building**(*request*, **args*, ***kwargs*)

Retrieves a building. If user doesn't belong to the building's org, fields will be masked to only those shared within the parent org's structure.

GET Expects building_id and organization_id in query string. building_id should be the *canonical_building* ID for the building, not the BuildingSnapshot id.

Returns:

```
    {
         'status': 'success or error',
         'message': 'error message, if any',
         'building': {'id': the building's id,
                      'canonical_building': the canonical building ID,
                      other fields this user has access to...
         },
         'imported_buildings': [ A list of buildings imported to create
                                   this building's record, in the same
                                   format as 'building'
                                 ],
         'projects': [
            // A list of the building's projects
            {
                "building": {
                    "approved_date":07/30/2014,
                    "compliant": null,
                    "approver": "demo@seed.lbl.gov"
                    "approved_date": "07/30/2014"
                    "compliant": null
                    "label": {
                        "color": "red",
                        "name": "non compliant",
                        id: 1
                    }
                }
                "description": null
                "id": 3
                "is_compliance": false
                "last_modified_by_id": 1
                "name": "project 1"
                "owner_id": 1
                "slug": "project-1"
                "status": 1
                "super_organization_id": 1
            },
            . . .
        ],
         'user_role': role of user in this org,
         'user_org_id': the org id this user belongs to
    }
```

seed.views.main.**get_building_report_data**(*request*, **args*, ***kwargs*)

This method returns a set of x,y building data for graphing. It expects as parameters

GET

Parameters

- **start_date** – The starting date for the data series with the format *YYYY-MM-DD*
- **end_date** – The starting date for the data series with the format *YYYY-MM-DD*
- **x_var** – The variable name to be assigned to the "x" value in the returned data series # NOQA
- **y_var** – The variable name to be assigned to the "y" value in the returned data series # NOQA
- **organization_id** – The organization to be used when querying data.

The x_var values should be from the following set of variable names:

- site_eui
- source_eui
- site_eui_weather_normalized
- source_eui_weather_normalized
- energy_score

The y_var values should be from the following set of variable names:

- gross_floor_area
- use_description
- year_built

This method includes building record count information as part of the result JSON in a property called "building_counts."

This property provides data on the total number of buildings available in each 'year ending' group, as well as the subset of those buildings that have actual data to graph. By sending these values in the result we allow the client to easily build a message like "200 of 250 buildings in this group have data."

Returns:

```
{
    "status": "success",
    "chart_data": [
        {
            "id" the id of the building,
            "yr_e": the year ending value for this data point
            "x": value for x var,
            "y": value for y var,
        },
        ...
    ],
    "building_counts": [
        {
            "yr_e": string for year ending
            "num_buildings": number of buildings in query results
            "num_buildings_w_data": number of buildings with valid data in query results
        },
        ...
    ]
    "num_buildings": total number of buildings in query results,
    "num_buildings_w_data": total number of buildings with valid data in the query results  # NC
}
```

—

parameters:

- name: x_var description: Name of column in building snapshot database to be used for "x" axis required: true type: string paramType: query
- name: y_var description: Name of column in building snapshot database to be used for "y" axis required: true type: string paramType: query
- start_date: description: The start date for the entire dataset. required: true type: string paramType: query

- end_date: description: The end date for the entire dataset. required: true type: string paramType: query
- name: organization_id description: User's organization which should be used to filter building query results required: true type: string paramType: query
- name: aggregate description: Aggregates data based on internal rules (given x and y var) required: true type: string paramType: query

type:

status: required: true type: string

chart_data: required: true type: array

num_buildings: required: true type: string

num_buildings_w_data: required: true type: string

status codes:

- code: 400 message: Bad request, only GET method is available
- code: 401 message: Not authenticated
- code: 403 message: Insufficient rights to call this procedure

`seed.views.main.`**`get_building_summary_report_data`**(*request*, **args*, ***kwargs*)

This method returns basic, high-level data about a set of buildings, filtered by organization ID.

It expects as parameters

GET

Parameters

- **`start_date`** – The starting date for the data series with the format *YYYY-MM-DD*
- **`end_date`** – The starting date for the data series with the format *YYYY-MM-DD*

Returns:

```
{
    "status": "success",
    "summary_data":
    {
        "num_buildings": "number of buildings returned from query",
        "avg_eui": "average EUI for returned buildings",
        "avg_energy_score": "average energy score for returned buildings"
    }
}
```

Units for return values are as follows:

property	units
avg_eui	kBtu/ft2/yr

—

parameters:

- name: organization_id description: User's organization which should be used to filter building query results required: true type: string paramType: query
- start_date: description: The start date for the entire dataset. required: true type: string paramType: query

- end_date: description: The end date for the entire dataset. required: true type: string paramType: query

type:

status: required: true type: string

summary_data: required: true type: object

status codes:

- code: 400 message: Bad request, only GET method is available
- code: 401 message: Not authenticated
- code: 403 message: Insufficient rights to call this procedure

seed.views.main.**get_column_mapping_suggestions**(*request*, **args*, ***kwargs*)

Returns suggested mappings from an uploaded file's headers to known data fields.

Payload:

```
{
    'import_file_id': The ID of the ImportRecord to examine,
    'org_id': The ID of the user's organization
}
```

Returns:

```
{
    'status': 'success',
    'suggested_column_mappings': {
        column header from file: [ (destination_column, score) ...]
        ...
    },
    'building_columns': [ a list of all possible columns ],
    'building_column_types': [a list of column types corresponding to building_columns],
}
```

..todo: The response of this method may not be correct. verify.

seed.views.main.**get_columns**(*request*, **args*, ***kwargs*)

returns a JSON list of columns a user can select as his/her default

GET Expects organization_id in the query string.

seed.views.main.**get_coparents**(*request*, **args*, ***kwargs*)

Returns the nodes in the BuildingSnapshot tree that can be unmatched.

GET Expects organization_id and building_id in the query string

Returns:

```
{
    'status': 'success',
    'coparents': [
        {
            "id": 333,
            "coparent": 223,
            "child": 443,
            "parents": [],
            "canonical_building_id": 1123
        },
        {
```

```
                "id": 223,
                "coparent": 333,
                "child": 443,
                "parents": [],
                "canonical_building_id": 1124
            },
            ...
        ]
    }
```

seed.views.main.**get_dataset**(*request*, **args*, ***kwargs*)

Retrieves ImportFile objects for one ImportRecord.

GET Expects dataset_id for an ImportRecord in the query string.

Returns:

```
    {
        'status': 'success',
        'dataset': {
            'name': Name of ImportRecord,
            'number_of_buildings': Total number of buildings in all ImportFiles for this dataset,
            'id': ID of ImportRecord,
            'updated_at': Timestamp of when ImportRecord was last modified,
            'last_modified_by': Email address of user making last change,
            'importfiles': [
                {
                   'name': Name of associated ImportFile, e.g. 'buildings.csv',
                   'number_of_buildings': Count of buildings in this file,
                   'number_of_mappings': Number of mapped headers to fields,
                   'number_of_cleanings': Number of fields cleaned,
                   'source_type': Type of file (see source_types),
                   'id': ID of ImportFile (needed for most operations)
                }
             ],
             ...
        },
            ...
    }
```

seed.views.main.**get_datasets**(*request*, **args*, ***kwargs*)

Retrieves all datasets for the user's organization.

GET Expects 'organization_id' of org to retrieve datasets from in query string.

Returns:

```
    {
        'status': 'success',
        'datasets':  [
            {
                'name': Name of ImportRecord,
                'number_of_buildings': Total number of buildings in all ImportFiles,
                'id': ID of ImportRecord,
                'updated_at': Timestamp of when ImportRecord was last modified,
                'last_modified_by': Email address of user making last change,
                'importfiles': [
                    {
                        'name': Name of associated ImportFile, e.g. 'buildings.csv',
                        'number_of_buildings': Count of buildings in this file,
```

```
                        'number_of_mappings': Number of mapped headers to fields,
                        'number_of_cleanings': Number of fields cleaned,
                        'source_type': Type of file (see source_types),
                        'id': ID of ImportFile (needed for most operations)
                    }
                ],
                ...
            },
            ...
        ]
    }
```

seed.views.main.**get_datasets_count**(*request*, **args*, ***kwargs*)

Retrieves the number of datasets for an org.

GET Expects organization_id in the query string.

Returns:

```
    {
        'status': 'success',
        'datasets_count': Number of datasets belonging to this org.
    }
```

seed.views.main.**get_default_building_detail_columns**(*request*, **args*, ***kwargs*)

Get default columns for building detail view.

front end is expecting a JSON object with an array of field names

Returns:

```
    {
        "columns": ["project_id", "name", "gross_floor_area"]
    }
```

seed.views.main.**get_default_columns**(*request*, **args*, ***kwargs*)

Get default columns for building list view.

front end is expecting a JSON object with an array of field names

Returns:

```
    {
        "columns": ["project_id", "name", "gross_floor_area"]
    }
```

seed.views.main.**get_first_five_rows**(*request*, **args*, ***kwargs*)

Retrieves the first five rows of an ImportFile.

Payload:

```
    {
        'import_file_id': The ID of the ImportFile
    }
```

Returns:

```
    {
        'status': 'success',
        'first_five_rows': [
            [list of strings of header row],
            [list of strings of first data row],
```

```
            ...
            [list of strings of fifth data row]
        ]
    }
```

seed.views.main.**get_import_file**(*request*, **args*, ***kwargs*)

Retrieves details about an ImportFile.

GET Expects import_file_id in the query string.

Returns:

```
    {
        'status': 'success',
        'import_file': {
            "name": Name of the uploaded file,
            "number_of_buildings": number of buildings in the file,
            "number_of_mappings": number of mapped columns,
            "number_of_cleanings": number of cleaned fields,
            "source_type": type of data in file, e.g. 'Assessed Raw'
            "number_of_matchings": Number of matched buildings in file,
            "id": ImportFile ID,
            'dataset': {
                'name': Name of ImportRecord file belongs to,
                'id': ID of ImportRecord file belongs to,
                'importfiles': [  # All ImportFiles in this ImportRecord, with
                    # requested ImportFile first:
                    {'name': Name of file,
                     'id': ID of ImportFile
                    }
                    ...
                ]
            }
        }
    }
```

seed.views.main.**get_match_tree**(*request*, **args*, ***kwargs*)

returns the BuildingSnapshot tree

GET Expects organization_id and building_id in the query string

Returns:

```
    {
        'status': 'success',
        'match_tree': [ // array of all the members of the tree
            {
                "id": 333,
                "coparent": 223,
                "child": 443,
                "parents": [],
                "canonical_building_id": 1123
            },
            {
                "id": 223,
                "coparent": 333,
                "child": 443,
                "parents": [],
                "canonical_building_id": 1124
            },
            {
```

```
                "id": 443,
                "coparent": null,
                "child": 9933,
                "parents": [333, 223],
                "canonical_building_id": 1123
            },
            {
                "id": 9933,
                "coparent": null,
                "child": null,
                "parents": [443],
                "canonical_building_id": 1123
            },
            ...
        ]
    }
```

seed.views.main.**get_raw_column_names**(*request*, **args*, ***kwargs*)

Retrieves a list of all column names from an ImportFile.

Payload:

```
{
    'import_file_id': The ID of the ImportFile
}
```

Returns:

```
{
    'status': 'success',
    'raw_columns': [
        list of strings of the header row of the ImportFile
    ]
}
```

seed.views.main.**get_raw_report_data**(*from_date*, *end_date*, *orgs*, *x_var*, *y_var*)

This method returns data used to generate graphing reports. It expects as parameters

GET

Parameters

- **from_date** – The starting date for the data series. Date object.
- **end_date** – The starting date for the data series with the format. Date object.
- **x_var** – The variable name to be assigned to the "x" value in the returned data series.
- **y_var** – The variable name to be assigned to the "y" value in the returned data series.
- **orgs** – The organizations to be used when querying data.

The x and y variables should be column names in the BuildingSnapshot table. In theory they could be in the extra_data too and this works but is currently disabled.

Returns:

```
bldg_counts:  dict that looks like {year_ending : {"buildings_with_data": set(canonical ids), "b
                This is a collection of all year_ending dates and ids
                the canonical buildings that have data for that year
                and those that have files with that year_ending but no
                valid data point
                E.G.
```

```
                    "bldg_counts"     (pending)
                        __len__    int: 8
                        2000-12-31 (140037191378512)    dict: {'buildings_w_data': set([35897, 35898
                        2001-12-31 (140037292480784)    dict: {'buildings_w_data': set([35897, 35898
    data:   dict that looks like {canonical_id : { year_ending : {'x': x_value, 'y': y_value', 'rele
            This is the actual data for the building.  The top level key is
            the canonical_id then the next level is the year_ending and
            under that is the actual data.  NOTE:  If the year has files
            for a building but no valid data there will be an entry for
            that year but the x and y values will be None.

            E.G.
            "data"     (pending)
                __len__    int: 2
                35897 (28780560)    defaultdict: defaultdict(<type 'dict'>, {datetime.date(2001, 12,
                    __len__    int: 8
                    2000-12-31 (140037191378512)    dict: {'y': 400000.0, 'x': 28.2, 'release_date':
                    2001-12-31 (140037292480784)    dict: {'y': 95.0, 'x': 88.0, 'release_date': dat
```

seed.views.main.**get_total_number_of_buildings_for_user**(*request*, **args*, ***kwargs*)
gets a count of all buildings in the user's organizations

seed.views.main.**home**(*request*, **args*, ***kwargs*)
the main view for the app Sets in the context for the django template:

- **app_urls**: a json object of all the URLs that is loaded in the JS global namespace
- **username**: the request user's username (first and last name)
- **AWS_UPLOAD_BUCKET_NAME**: S3 direct upload bucket
- **AWS_CLIENT_ACCESS_KEY**: S3 direct upload client key
- **FILE_UPLOAD_DESTINATION**: 'S3' or 'filesystem'

seed.views.main.**progress**(*request*, **args*, ***kwargs*)
Get the progress (percent complete) for a task.

Payload:

```
{
    'progress_key': The progress key from starting a background task
}
```

Returns:

```
{
    'progress_key': The same progress key,
    'progress': Percent completion
}
```

seed.views.main.**public_search**(*request*, **args*, ***kwargs*)
the public API unauthenticated endpoint

see `search_buildings` for the non-public version

seed.views.main.**remap_buildings**(*request*, **args*, ***kwargs*)
Re-run the background task to remap buildings as if it hadn't happened at all. Deletes mapped buildings for a given ImportRecord, resets status.

NB: will not work if buildings have been merged into CanonicalBuilings.

Payload:

```
{
    'file_id': The ID of the ImportFile to be remapped
}
```

Returns:

```
{
    'status': 'success' or 'error',
    'progress_key': ID of background job, for retrieving job progress
}
```

seed.views.main.**save_column_mappings**(*request*, **args*, ***kwargs*)

Saves the mappings between the raw headers of an ImportFile and the destination fields in the BuildingSnapshot model.

Valid source_type values are found in `seed.models.SEED_DATA_SOURCES`

Payload:

```
{
    "import_file_id": ID of the ImportFile record,
    "mappings": [
        ["destination_field": "raw_field"],  #direct mapping
        ["destination_field2":
            ["raw_field1", "raw_field2"],  #concatenated mapping
        ...
    ]
}
```

Returns:

```
{'status': 'success'}
```

seed.views.main.**save_match**(*request*, **args*, ***kwargs*)

Adds or removes a match between two BuildingSnapshots. Creating a match creates a new BuildingSnapshot with merged data.

Payload:

```
{
    'organization_id': current user organization id,
    'source_building_id': ID of first BuildingSnapshot,
    'target_building_id': ID of second BuildingSnapshot,
    'create_match': True to create match, False to remove it,
    'organization_id': ID of user's organization
}
```

Returns:

```
{
    'status': 'success',
    'child_id': The ID of the newly-created BuildingSnapshot
                containing merged data from the two parents.
}
```

seed.views.main.**save_raw_data**(*request*, **args*, ***kwargs*)

Starts a background task to import raw data from an ImportFile into BuildingSnapshot objects.

Payload:

```
{
    'file_id': The ID of the ImportFile to be saved
}
```

Returns:

```
{
    'status': 'success' or 'error',
    'progress_key': ID of background job, for retrieving job progress
}
```

seed.views.main.**search_building_snapshots**(*request*, **args*, ***kwargs*)

Retrieves a paginated list of BuildingSnapshots matching search params.

Payload:

```
{
    'q': a string to search on (optional),
    'order_by': which field to order by (e.g. pm_property_id),
    'import_file_id': ID of an import to limit search to,
    'filter_params': {
        a hash of Django-like filter parameters to limit query.  See seed.search.filter_other_pa
    }
    'page': Which page of results to retrieve (default: 1),
    'number_per_page': Number of buildings to retrieve per page (default: 10),
}
```

Returns:

```
{
    'status': 'success',
    'buildings': [
        {
            'pm_property_id': ID of building (from Portfolio Manager),
            'address_line_1': First line of building's address,
            'property_name': Building's name, if any
        }...
    ]
    'number_matching_search': Total number of buildings matching search,
    'number_returned': Number of buildings returned for this page
}
```

seed.views.main.**search_buildings**(*request*, **args*, ***kwargs*)

Retrieves a paginated list of CanonicalBuildings matching search params.

Payload:

```
{
    'q': a string to search on (optional),
    'show_shared_buildings': True to include buildings from other orgs in this user's org tree,
    'order_by': which field to order by (e.g. pm_property_id),
    'import_file_id': ID of an import to limit search to,
    'filter_params': {
        a hash of Django-like filter parameters to limit query.  See seed.search.filter_other_pa
        If 'project__slug' is included and set to a project's slug, buildings will include assoc
        for that project.
    }
    'page': Which page of results to retrieve (default: 1),
    'number_per_page': Number of buildings to retrieve per page (default: 10),
}
```

Returns:

```
{
    'status': 'success',
    'buildings': [
        {
            all fields for buildings the request user has access to, e.g.:
                'canonical_building': the CanonicalBuilding ID of the building,
                'pm_property_id': ID of building (from Portfolio Manager),
                'address_line_1': First line of building's address,
                'property_name': Building's name, if any
            ...
        }...
    ]
    'number_matching_search': Total number of buildings matching search,
    'number_returned': Number of buildings returned for this page
}
```

seed.views.main.**set_default_building_detail_columns**(*request*, **args*, ***kwargs*)

seed.views.main.**set_default_columns**(*request*, **args*, ***kwargs*)

seed.views.main.**start_mapping**(*request*, **args*, ***kwargs*)

Starts a background task to convert imported raw data into BuildingSnapshots, using user's column mappings.

Payload:

```
{
    'file_id': The ID of the ImportFile to be mapped
}
```

Returns:

```
{
    'status': 'success' or 'error',
    'progress_key': ID of background job, for retrieving job progress
}
```

seed.views.main.**start_system_matching**(*request*, **args*, ***kwargs*)

Starts a background task to attempt automatic matching between buildings in an ImportFile with other existing buildings within the same org.

Payload:

```
{
    'file_id': The ID of the ImportFile to be matched
}
```

Returns:

```
{
    'status': 'success' or 'error',
    'progress_key': ID of background job, for retrieving job progress
}
```

seed.views.main.**update_building**(*request*, **args*, ***kwargs*)

Manually updates a building's record. Creates a new BuildingSnapshot for the resulting changes.

PUT

Payload:

```
{
    "organization_id": "organization id as integer",
    "building":
        {
            "canonical_building": "canonical building ID as integer"
            "fieldname": "value",
            "...": "Remaining fields in the BuildingSnapshot; see get_columns() endpoint for com
        }
}
```

Returns:

```
{
    "status": "success",
    "child_id": "The ID of the newly-created BuildingSnapshot"
}
```

seed.views.main.**update_dataset**(*request*, **args*, ***kwargs*)

Updates the name of a dataset.

Payload:

```
{
    'dataset': {
        'id': The ID of the Import Record,
        'name': The new name for the ImportRecord
    }
}
```

Returns:

```
{
    'status': 'success' or 'error',
    'message': 'error message, if any'
}
```

seed.views.main.**version**(*request*, **args*, ***kwargs*)

Returns the SEED version and current git sha

6.18.5 Meters

seed.views.meters.**add_meter_to_building**(*request*, **args*, ***kwargs*)

Will add a building to an existing meter.

Payload:

```
{
    'organization_id': 435,
    'building_id': 342,
    'meter_name': 'Unit 34.',
    'energy_type': 'Electricity',
    'energy_units': 'kWh'
}
```

seed.views.meters.**add_timeseries**(*request*, **args*, ***kwargs*)

Add time series data for a meter.

Payload:

```
    {
        'organization_id': 435,
        'meter_id': 34,
        'timeseries': [
            {
                'begin_time': 2342342232,
                'end_time': 23423433433,
                'cost': 232.23,
            }...
        ]
    }
```

seed.views.meters.**get_meters**(*request*, **args*, ***kwargs*)
Returns all of the meters for a building.

Expected GET params:

building_id: int, unique identifier for a (canonical) building.

seed.views.meters.**get_timeseries**(*request*, **args*, ***kwargs*)
Return all time series data for a building, grouped by meter.

Expected GET params:

meter_id: int, unique identifier for the meter. offset: int, the offset from the most recent meter data to begin showing. num: int, the number of results to show.

6.18.6 Projects

seed.views.projects.**add_buildings_to_project**(*request*, **args*, ***kwargs*)
Adds buildings to a project.

Payload:

```
    {
     'project':
         {
          'project_slug': Identifier of project to add buildings to,
          'selected_buildings': A list of building IDs to add to project
         }
    }
```

Returns:

```
    {
     'status': 'success' or 'error',
     'message': 'error message, if any',
     'project_loading_cache_key': Identifier for the background job, to
         determine the job's progress
    }
```

seed.views.projects.**create_project**(*request*, **args*, ***kwargs*)
Creates a new project. @TODO: What's a compliance_type?

Payload:

```
    {
     'organization_id': ID of org to associate new project with,
     'project': {
       'name': name of new project,
```

```
        'compliance_type': description of type of compliance,
        'description': description of new project,
        'end_date': Timestamp for when project ends,
        'deadline_date': Timestamp for compliance deadline
      }
    }
```

Returns:

```
    {
     'status': 'success' or 'error',
     'message': 'error message, if any',
     'project_slug': Identifier of new project, if project successfully
                     created
    }
```

seed.views.projects.**delete_project**(*request*, **args*, ***kwargs*)

Deletes a project.

Payload:

```
    {
     'project_slug': identifier (slug) for the project
     'organization_id': ID of the org the project belongs to
    }
```

Returns:

```
    {
     'status': 'success or error',
     'message': 'error message, if any'
    }
```

seed.views.projects.**get_adding_buildings_to_project_status_percentage**(*request*, **args*, ***kwargs*)

Returns percentage complete of background task for adding building to project.

Payload:

```
    {
        'project_loading_cache_key': Job identifier from add_buildings_to_project.
    }
```

Returns:

```
    {
        'status': 'success',
            'progress_object': {
                'status': job status,
                'progress': percent job done (out of 100),
                'progress_key': progress_key for job,
                'numerator': number buildings added,
                'denominator': total number of building to add
        }
    }
```

seed.views.projects.**get_project**(*request*, **args*, ***kwargs*)

Retrieves details about a project.

GET Expects the project's identifier (slug) as project_slug in the query string. Expects an organization_id (to which project belongs) in the query string.

Returns:

```
{
 'name': project's name,
 'slug': project's identifier,
 'status': 'active',
 'number_of_buildings': Count of buildings associated with project
 'last_modified': Timestamp when project last changed
 'last_modified_by': {
    'first_name': first name of user that made last change,
    'last_name': last name,
    'email': email address,
    },
 'is_compliance': True if project is a compliance project,
 'compliance_type': Description of compliance type,
 'deadline_date': Timestamp of when compliance is due,
 'end_date': Timestamp of end of project
}
```

seed.views.projects.**get_projects**(*request*, **args*, ***kwargs*)

Retrieves all projects for a given organization.

GET Expects organization_id in query string.

Returns:

```
{
    'status': 'success',
    'projects': [
        {
            'name': project's name,
            'slug': project's identifier,
            'status': 'active',
            'number_of_buildings': Count of buildings associated with project
            'last_modified': Timestamp when project last changed
            'last_modified_by': {
                'first_name': first name of user that made last change,
                'last_name': last name,
                'email': email address,
            },
            'is_compliance': True if project is a compliance project,
            'compliance_type': Description of compliance type,
            'deadline_date': Timestamp of when compliance is due,
            'end_date': Timestamp of end of project
        }...
    ]
}
```

seed.views.projects.**get_projects_count**(*request*, **args*, ***kwargs*)

Returns the number of projects within the org tree to which a user belongs. Counts projects in parent orgs and sibling orgs.

GET Expects organization_id for the user's org.

Returns:

```
{
    'status': 'success',
```

```
        'projects_count': count of projects
    }
```

seed.views.projects.**move_buildings**(*request*, **args*, ***kwargs*)

Moves buildings from one project to another.

Payload:

```
    {
        "buildings": [
            "00010811",
            "00010809"
        ],
        "copy": true to copy the buildings, false to move,
        "search_params": {
            "filter_params": {
                "project__slug": "proj-1"
            },
            "project_slug": 34,
            "q": "hotels"
        },
        "select_all_checkbox": false,
        "source_project_slug": "proj-1",
        "target_project_slug": "proj-2"
    }
```

Returns:

```
    { 'status': 'success' }
```

seed.views.projects.**remove_buildings_from_project**(*request*, **args*, ***kwargs*)

Removes buildings from a project.

Payload:

```
    {
     'project':
         {
          'slug': Identifier of project to remove buildings from,
          'selected_buildings': A list of building IDs to remove
         }
    }
```

Returns:

```
    {
     'status': 'success' or 'error',
     'message': 'error message, if any',
     'project_removing_cache_key': Identifier for the background job, to
         determine the job's progress
    }
```

seed.views.projects.**update_project**(*request*, **args*, ***kwargs*)

Updates an existing project's details and compliance info.

Payload:

```
    {
     'project': {
       'project_slug': Identifier of project to update,
       'name': new name for project,
```

```
        'is_compliance': true or false,
        'compliance_type': (optional if 'is_compliance' is false)
             description of type of compliance,
        'end_date': (optional if 'is_compliance' is false) Timestamp for
             when project ends,
        'deadline_date': (optional if 'is_compliance' is false) Timestamp
             for compliance deadline
     }
    }
```

Returns:

```
    {
     'status': 'success' or 'error',
     'message': 'error message, if any'
    }
```

seed.views.projects.**update_project_building**(*request*, **args*, ***kwargs*)

Updates extra information about the building/project relationship. In particular, whether the building is compliant and who approved it.

Payload:

```
    {
        'project_slug': identifier of project,
        'building_id': ID of building,
        'label': {
            'id': Identifier of label to apply.
        }
    }
```

Returns:

```
    {
        'status': 'success',
        'approved_date': Timestamp of change (now),
        'approver': Email address of user making change.
    }
```

6.18.7 Module contents

CHAPTER 7

Developer Resources

7.1 General Notes

7.1.1 Flake Settings

Flake is used to statically verify code syntax. If the developer is running flake from the command line, they should ignore the following checks in order to emulate the same checks as the CI machine.

Code	Description
E402	module level import not at top of file
E501	line too long (82 characters) or max-line = 100
E731	do not assign a lambda expression, use a def
W503	line break occurred before a binary operator

To run flake locally call:

```
tox -e flake8
```

Note: tox will soon be removed from the development process. Instructions will be updated.

7.2 Django Notes

Both Django and AngurlarJS are used for url routing. Django routes are in *seed/urls/main.py*

7.2.1 AWS S3

Amazon AWS S3 Expires headers should be set on the AngularJS partials if using S3 with the management command: set_s3_expires_headers_for_angularjs_partials

Example:

```
python manage.py set_s3_expires_headers_for_angularjs_partials --verbosity=3
```

The default user invite reply-to email can be overridden in the config/settings/common.py file. The *SERVER_EMAIL* settings var is the reply-to email sent along with new account emails.

```
# config/settings/common.py
PASSWORD_RESET_EMAIL = 'reset@seed.lbl.gov'
SERVER_EMAIL = 'no-reply@seed.lbl.gov'
```

7.3 AngularJS Integration Notes

7.3.1 Template Tags

Angular and Django both use *{{* and *}}* as variable delimiters, and thus the AngularJS variable delimiters are renamed *{$* and *$}*.

```
window.BE.apps.seed = angular.module('BE.seed', ['ngRoute', "ngCookies"], function ($interpolateProvi
        $interpolateProvider.startSymbol("{$");
        $interpolateProvider.endSymbol("$}");
    }
);
```

7.3.2 Django CSRF Token and AJAX Requests

For ease of making angular *$http* requests, we automatically add the CSRF token to all *$http* requests as recommended by http://django-angular.readthedocs.io/en/latest/integration.html#xmlhttprequest

```
window.BE.apps.seed.run(function ($http, $cookies) {
    $http.defaults.headers.common['X-CSRFToken'] = $cookies['csrftoken'];
});
```

7.3.3 Routes and Partials or Views

Routes in *static/seed/js/seed.js* (the normal angularjs *app.js*)

```
window.BE.apps.seed.config(['$routeProvider', function ($routeProvider) {
        $routeProvider
            .when('/', {
                templateUrl: static_url + '/seed/partials/home.html'
            })
            .when('/projects', {
                controller: 'project_list_controller',
                templateUrl: static_url + '/seed/partials/projects.html'
            })
            .when('/buildings', {
                templateUrl: static_url + '/seed/partials/buildings.html'
            })
            .when('/admin', {
                controller: 'seed_admin_controller',
                templateUrl: static_url + '/seed/partials/admin.html'
            })
            .otherwise({ redirectTo: '/' });
    }]);
```

HTML partials in *static/seed/partials/*

on production and staging servers on AWS, or for the partial html templates loaded on S3, or a CDN, the external resource should be added to the white list in *static/seed/js/seed/js*

```
// white list for s3
window.BE.apps.seed.config(function( $sceDelegateProvider ) {
$sceDelegateProvider.resourceUrlWhitelist([
    // localhost
    'self',
    // AWS s3
    'https://be-*.amazonaws.com/**'
    ]);
});
```

7.4 Logging

Information about error logging can be found here - https://docs.djangoproject.com/en/1.7/topics/logging/

Below is a standard set of error messages from Django.

A logger is configured to have a log level. This log level describes the severity of the messages that the logger will handle. Python defines the following log levels:

```
DEBUG: Low level system information for debugging purposes
INFO: General system information
WARNING: Information describing a minor problem that has occurred.
ERROR: Information describing a major problem that has occurred.
CRITICAL: Information describing a critical problem that has occurred.
```

Each message that is written to the logger is a Log Record. The log record is stored in the web server & Celery

7.5 Testing

JS tests can be run with Jasmine at the url *app/angular_js_tests/*.

Python unit tests are run with

```
python manage.py test --settings=config.settings.test
```

Run coverage using

```
coverage run manage.py test --settings=config.settings.test
coverage report --fail-under=83
```

Python compliance uses PEP8 with flake8

```
flake8
# or
tox -e flake8
```

JS Compliance uses jshint

```
jshint seed/static/seed/js
```

CHAPTER 8

License

CHAPTER 9

Help

9.1 For SEED-Platform Users

Please visit our User Support website for tutorials and documentation to help you learn how to use SEED-Platform.

https://sites.google.com/a/lbl.gov/seed/

There is also a link to the SEED-Platform Users forum, where you can connect with other users.

https://groups.google.com/forum/#!forum/seed-platform-users

For direct help on a specific problem, please email: SEED-Support@lists.lbl.gov

9.2 For SEED-Platform Developers

The Open Source code is available on the Github organization SEED-Platform:

https://github.com/SEED-platform

Please join the SEED-Platform Dev forum where you can connect with other developers.

https://groups.google.com/forum/#!forum/seed-platform-dev

CHAPTER 10

Updating this documentation

This python code documentation was generated by running the following:

```
$ pip install -r requirements/local.txt
$ sphinx-apidoc -o docs/source/modules . seed/lib/mcm seed/lib/superperms terrain.py
$ cd docs
$ make html
```

CHAPTER 11

Indices and tables

- genindex
- modindex
- search

C

config.storage, 72
config.template_context, 72
config.tests, 72
config.utils, 72
config.views, 72
config.wsgi, 73

S

seed, 140
seed.audit_logs.models, 63
seed.audit_logs.tests, 64
seed.audit_logs.urls, 65
seed.audit_logs.views, 65
seed.cleansing.models, 67
seed.cleansing.tasks, 69
seed.cleansing.tests, 69
seed.cleansing.urls, 69
seed.cleansing.views, 69
seed.common.mapper, 70
seed.common.util, 71
seed.common.views, 71
seed.data, 73
seed.data.bedes, 73
seed.data_importer, 86
seed.data_importer.managers, 73
seed.data_importer.models, 73
seed.data_importer.urls, 83
seed.data_importer.utils, 83
seed.data_importer.views, 32
seed.decorators, 111
seed.factory, 112
seed.features, 86
seed.green_button, 89
seed.green_button.tests, 87
seed.green_button.tests.test_xml_importer, 86
seed.green_button.xml_importer, 87
seed.landing, 94
seed.landing.forms, 89
seed.landing.management, 89
seed.landing.management.commands, 89
seed.landing.management.commands.update_eula, 89
seed.landing.models, 89
seed.landing.tests, 93
seed.landing.urls, 93
seed.landing.views, 93
seed.lib, 95
seed.lib.exporter, 94
seed.management, 99
seed.management.commands, 99
seed.management.commands.set_s3_expires_headers_for 99
seed.managers, 97
seed.managers.json, 97
seed.managers.tests, 97
seed.managers.tests.test_json_manager, 97
seed.mappings, 97
seed.mappings.mapper, 95
seed.mappings.seed_mappings, 96
seed.models, 112
seed.public, 98
seed.public.models, 97
seed.search, 136
seed.serializers, 141
seed.serializers.celery, 140
seed.serializers.labels, 141
seed.tasks, 139
seed.templatetags.breadcrumbs, 99
seed.test_helpers, 101
seed.test_helpers.factory.helpers, 101
seed.test_helpers.factory.lib.chomsky, 100
seed.tests.test_admin_views, 102
seed.tests.test_decorators, 102
seed.tests.test_exporters, 103
seed.tests.test_models, 103
seed.tests.test_tasks, 104
seed.tests.test_views, 106

seed.tests.tests, 110
seed.tests.util, 110
seed.token_generators, 140
seed.urls, 140
seed.urls.accounts, 141
seed.urls.api, 141
seed.urls.main, 141
seed.urls.projects, 141
seed.utils, 140
seed.utils.api, 141
seed.utils.buildings, 142
seed.utils.constants, 143
seed.utils.mapping, 143
seed.utils.organizations, 143
seed.utils.projects, 143
seed.utils.time, 145
seed.views, 177
seed.views.accounts, 24
seed.views.api, 24
seed.views.main, 33
seed.views.meters, 172
seed.views.projects, 173

A

abstract (seed.data_importer.models.NotDeletableModel.Meta attribute), 80
acquire_lock() (in module seed.data_importer.utils), 83
action_response (seed.audit_logs.models.AuditLog attribute), 63
ACTIVE_STATUS (seed.models.Project attribute), 129
add_buildings_to_project() (in module seed.views.projects), 173
add_files_url (seed.data_importer.models.ImportRecord attribute), 77
add_meter_to_building() (in module seed.views.meters), 172
add_org() (in module seed.views.accounts), 24
add_timeseries() (in module seed.views.meters), 172
add_user() (in module seed.views.accounts), 25
add_user_to_organization() (in module seed.views.accounts), 25
adding_buildings_status_percentage_cache_key (seed.models.Project attribute), 129
address_line_1_source (seed.models.BuildingSnapshot attribute), 113
address_line_2_source (seed.models.BuildingSnapshot attribute), 113
AdminViewsTest (class in seed.tests.test_admin_views), 102
ajax_request() (in module seed.decorators), 111
api_endpoint() (in module seed.utils.api), 141
api_error() (in module seed.common.views), 71
api_success() (in module seed.common.views), 71
APIBypassCSRFMiddleware (class in seed.utils.api), 141
app_namespace (seed.data_importer.models.ImportRecord attribute), 77
apply() (seed.common.mapper.Mapping method), 70
apply_data_func() (in module seed.tasks), 139
apply_extra_data() (in module seed.tasks), 139
apply_map() (in module seed.common.util), 71
approver (seed.models.ProjectBuilding attribute), 131
as_collection() (in module seed.green_button.xml_importer), 87
as_json() (seed.common.mapper.MapItem method), 70
assert_expected_mappings() (seed.tests.test_views.TestMCMViews method), 109
assert_fn_mapping() (seed.green_button.tests.test_xml_importer.GreenButt method), 86
assert_models_created() (seed.green_button.tests.test_xml_importer.GreenB method), 86
ASSESSOR_FIELDS (seed.cleansing.models.Cleansing attribute), 67
ASSESSOR_FIELDS_BY_COLUMN (seed.cleansing.models.Cleansing attribute), 67
AttributeOption (class in seed.models), 112
AttributeOption.DoesNotExist, 112
AttributeOption.MultipleObjectsReturned, 112
audit_logs (seed.models.CanonicalBuilding attribute), 122
AUDITING_COMPLIANCE_CHOICE (seed.models.Compliance attribute), 126
AuditLog (class in seed.audit_logs.models), 63
AuditLog.DoesNotExist, 63
AuditLog.MultipleObjectsReturned, 63
auditlog_set (seed.landing.models.SEEDUser attribute), 90
AuditLogManager (class in seed.audit_logs.models), 64
AuditLogModelTests (class in seed.audit_logs.tests), 64
AuditLogQuerySet (class in seed.audit_logs.models), 64
AuditLogViewTests (class in seed.audit_logs.tests), 65

B

base_fields (seed.landing.forms.LoginForm attribute), 89
batch_qs() (in module seed.lib.exporter), 95
BENCHMARK_COMPLIANCE_CHOICE (seed.models.Compliance attribute), 126
best_guess (seed.models.CanonicalBuilding attribute), 123
best_guess_canonical_building (seed.models.BuildingSnapshot attribute), 114
block_number_source (seed.models.BuildingSnapshot attribute), 114

BLUE_CHOICE (seed.models.StatusLabel attribute), 132
body (seed.tests.util.FakeRequest attribute), 110
breadcrumb() (in module seed.templatetags.breadcrumbs), 99
breadcrumb_root() (in module seed.templatetags.breadcrumbs), 99
breadcrumb_url() (in module seed.templatetags.breadcrumbs), 100
breadcrumb_url_root() (in module seed.templatetags.breadcrumbs), 100
BreadcrumbNode (class in seed.templatetags.breadcrumbs), 99
bs1_data (seed.tests.test_models.TestBuildingSnapshot attribute), 103
bs2_data (seed.tests.test_models.TestBuildingSnapshot attribute), 103
build_json_params() (in module seed.search), 136
build_shared_buildings_orgs() (in module seed.search), 136
building_certification_source (seed.models.BuildingSnapshot attribute), 114
building_count_source (seed.models.BuildingSnapshot attribute), 114
building_data() (in module seed.green_button.xml_importer), 87
building_headers (seed.models.CustomBuildingHeaders attribute), 127
building_model_content_type (seed.data_importer.models.BuildingImportRecord attribute), 73
building_record (seed.data_importer.models.BuildingImportRecord attribute), 74
building_snapshot (seed.models.BuildingAttributeVariant attribute), 113
building_snapshot (seed.models.Meter attribute), 128
building_snapshot (seed.models.ProjectBuilding attribute), 131
building_snapshot() (seed.factory.SEEDFactory class method), 112
building_snapshots (seed.models.Project attribute), 129
building_variant (seed.models.AttributeOption attribute), 112
BuildingAttributeVariant (class in seed.models), 112
BuildingAttributeVariant.DoesNotExist, 113
BuildingAttributeVariant.MultipleObjectsReturned, 113
BuildingDetailViewTests (class in seed.tests.test_views), 106
BuildingImportRecord (class in seed.data_importer.models), 73
BuildingImportRecord.DoesNotExist, 73
BuildingImportRecord.MultipleObjectsReturned, 73
buildingimportrecord_set (seed.data_importer.models.ImportRecord attribute), 77
BuildingSnapshot (class in seed.models), 113
BuildingSnapshot.DoesNotExist, 113
BuildingSnapshot.MultipleObjectsReturned, 113
buildingsnapshot_set (seed.data_importer.models.ImportFile attribute), 75
buildingsnapshot_set (seed.landing.models.SEEDUser attribute), 90
buildingsnapshot_set (seed.models.CanonicalBuilding attribute), 123

C

cache_first_rows() (in module seed.tasks), 139
cache_first_rows() (seed.data_importer.models.ImportFile method), 75
cache_key() (seed.cleansing.models.Cleansing static method), 67
CachedS3BotoStorage (class in config.storage), 72
canonical_building (seed.models.BuildingSnapshot attribute), 114
canonical_for_ds (seed.models.BuildingSnapshot attribute), 114
canonical_snapshot (seed.models.CanonicalBuilding attribute), 123
CanonicalBuilding (class in seed.models), 122
CanonicalBuilding.DoesNotExist, 122
CanonicalBuilding.MultipleObjectsReturned, 122
canonicalbuilding_set (seed.models.BuildingSnapshot attribute), 115
canonicalbuilding_set (seed.models.StatusLabel attribute), 132
CanonicalManager (class in seed.models), 123
CeleryDatetimeSerializer (class in seed.serializers.celery), 140
check_token() (seed.token_generators.SignupTokenGenerator method), 140
child_tree (seed.models.BuildingSnapshot attribute), 115
children (seed.models.BuildingSnapshot attribute), 115
chunk_iterable() (in module seed.data_importer.utils), 83
city_source (seed.models.BuildingSnapshot attribute), 115
clean() (seed.models.BuildingSnapshot method), 115
clean_api_regex() (in module seed.utils.api), 142
clean_canonicals() (in module seed.models), 133
cleaned_data_rows (seed.data_importer.models.ImportFile attribute), 75
CLEANING_ACTIVE_CACHE_KEY (seed.data_importer.models.ImportFile attribute), 75
CLEANING_ACTIVE_CACHE_KEY_GENERATOR() (seed.data_importer.models.ImportFile class method), 75

CLEANING_PROGRESS_KEY (seed.data_importer.models.ImportFile attribute), 75
cleaning_progress_pct (seed.data_importer.models.ImportFile attribute), 75
CLEANING_QUEUED_CACHE_KEY (seed.data_importer.models.ImportFile attribute), 75
CLEANING_QUEUED_CACHE_KEY_GENERATOR() (seed.data_importer.models.ImportFile class method), 75
cleanse() (seed.cleansing.models.Cleansing method), 67
Cleansing (class in seed.cleansing.models), 67
CleansingDataSample (class in seed.cleansing.tests), 69
CleansingDataTestCoveredBuilding (class in seed.cleansing.tests), 69
CleansingDataTestPM (class in seed.cleansing.tests), 69
CleansingViewTests (class in seed.cleansing.tests), 69
co_parent (seed.models.BuildingSnapshot attribute), 115
co_parents (seed.models.BuildingSnapshot attribute), 115
coercion_mapping_active (seed.data_importer.models.ImportFile attribute), 75
coercion_mapping_queued (seed.data_importer.models.ImportFile attribute), 75
CoercionRobot (class in seed.data_importer.utils), 83
COLOR_CHOICES (seed.models.StatusLabel attribute), 132
Column (class in seed.models), 124
Column.DoesNotExist, 124
Column.MultipleObjectsReturned, 124
column_mapped (seed.models.ColumnMapping attribute), 125
column_raw (seed.models.ColumnMapping attribute), 125
column_set (seed.models.Enum attribute), 127
column_set (seed.models.Unit attribute), 133
ColumnMapping (class in seed.models), 125
ColumnMapping.DoesNotExist, 125
ColumnMapping.MultipleObjectsReturned, 125
columnmapping_set (seed.landing.models.SEEDUser attribute), 90
combined_model_and_field (seed.data_importer.models.TableColumnMapping attribute), 81
Command (class in seed.landing.management.commands.update_eula), 89
Command (class in seed.management.commands.set_s3_expires_headers_for_angularjs_partials), 99
Compliance (class in seed.models), 126
Compliance.DoesNotExist, 126
Compliance.MultipleObjectsReturned, 126
COMPLIANCE_CHOICES (seed.models.Compliance attribute), 126
compliance_set (seed.models.Project attribute), 130
ComplianceTestCase (class in seed.tests.tests), 110
conditioned_floor_area_source (seed.models.BuildingSnapshot attribute), 115
config.storage (module), 72
config.template_context (module), 72
config.tests (module), 72
config.utils (module), 72
config.views (module), 72
config.wsgi (module), 73
construct_obj_row() (in module seed.lib.exporter), 95
content_object (seed.audit_logs.models.AuditLog attribute), 63
content_type (seed.audit_logs.models.AuditLog attribute), 63
convert_datestr() (in module seed.utils.time), 145
convert_to_js_timestamp() (in module seed.search), 136
convert_to_js_timestamp() (in module seed.utils.time), 145
copy_buildings() (in module seed.utils.projects), 143
create() (seed.serializers.labels.UpdateBuildingLabelsSerializer method), 141
create_building_queryset() (in module seed.search), 136
create_crumb() (in module seed.templatetags.breadcrumbs), 100
create_crumb_first() (in module seed.templatetags.breadcrumbs), 100
create_dataset() (in module seed.views.main), 33
create_map() (in module seed.common.util), 71
create_models() (in module seed.green_button.xml_importer), 87
create_note() (in module seed.audit_logs.views), 65
create_organization() (in module seed.utils.organizations), 143
create_pm_mapping() (in module seed.views.main), 34
create_project() (in module seed.views.projects), 173
create_sub_org() (in module seed.views.accounts), 25
custom_id_1_source (seed.models.BuildingSnapshot attribute), 115
CustomBuildingHeaders (class in seed.models), 127
CustomBuildingHeaders.DoesNotExist, 127
CustomBuildingHeaders.MultipleObjectsReturned, 127

D

dashboard_url (seed.data_importer.models.ImportRecord attribute), 79
data_rows (seed.data_importer.models.ImportFile attribute), 75
data_type_check() (seed.cleansing.models.Cleansing method), 67

datacoercion_errors (seed.data_importer.models.TableColumnMapping attribute), 81
DataCoercionMapping (class in seed.data_importer.models), 74
DataCoercionMapping.DoesNotExist, 74
DataCoercionMapping.MultipleObjectsReturned, 74
datacoercionmapping_set (seed.data_importer.models.TableColumnMapping attribute), 81
datacoercions (seed.data_importer.models.TableColumnMapping attribute), 81
DataImporterViewTests (class in seed.tests.test_views), 106
de_camel_case() (in module config.utils), 72
declared_fields (seed.landing.forms.LoginForm attribute), 89
DecoratorMixin() (in module seed.decorators), 111
default() (seed.serializers.celery.CeleryDatetimeSerializer method), 140
default_building_detail_custom_columns (seed.landing.models.SEEDUser attribute), 90
default_custom_columns (seed.landing.models.SEEDUser attribute), 90
DEFAULT_LABELS (seed.models.StatusLabel attribute), 132
default_manager (seed.data_importer.models.ImportFile attribute), 75
default_manager (seed.data_importer.models.ImportRecord attribute), 77
default_organization (seed.landing.models.SEEDUser attribute), 90
DefaultColumnsViewTests (class in seed.tests.test_views), 107
delete() (seed.data_importer.models.ImportRecord method), 77
delete() (seed.data_importer.models.NotDeletableModel method), 80
delete_buildings() (in module seed.views.main), 34
delete_dataset() (in module seed.views.main), 35
delete_duplicates_from_import_file() (in module seed.views.main), 35
delete_file() (in module seed.views.main), 35
delete_matching_buildings() (in module seed.utils.projects), 143
delete_organization() (in module seed.views.main), 36
delete_organization_buildings() (in module seed.views.main), 36
delete_project() (in module seed.views.projects), 174
delete_rules() (seed.cleansing.models.Rules static method), 68
delete_url (seed.data_importer.models.ImportRecord attribute), 78
destination_django_field (seed.data_importer.models.TableColumnMapping attribute), 81
destination_django_field_choices (seed.data_importer.models.TableColumnMapping attribute), 81
destination_django_field_has_choices (seed.data_importer.models.TableColumnMapping attribute), 81
display_as_in_progress (seed.data_importer.models.ImportRecord attribute), 78
district_source (seed.models.BuildingSnapshot attribute), 116
DjangoFunctionalFactory (class in seed.test_helpers.factory.helpers), 101
drf_api_endpoint() (in module seed.utils.api), 142
duplicate (seed.models.BuildingSnapshot attribute), 116
DuplicateDataError, 139

E

email_user() (seed.landing.models.SEEDUser method), 91
energy_alerts_source (seed.models.BuildingSnapshot attribute), 116
energy_score_source (seed.models.BuildingSnapshot attribute), 116
energy_type() (in module seed.green_button.xml_importer), 87
energy_units() (in module seed.green_button.xml_importer), 88
Enum (class in seed.models), 127
enum (seed.models.Column attribute), 124
Enum.DoesNotExist, 127
Enum.MultipleObjectsReturned, 127
enum_values (seed.models.Enum attribute), 127
EnumValue (class in seed.models), 128
EnumValue.DoesNotExist, 128
EnumValue.MultipleObjectsReturned, 128
estimated_seconds_remaining (seed.data_importer.models.ImportRecord attribute), 78
expected_mappings (seed.tests.test_views.TestMCMViews attribute), 109
export() (seed.lib.exporter.Exporter method), 94
export_buildings() (in module seed.views.main), 36
export_buildings_download() (in module seed.views.main), 37
export_buildings_progress() (in module seed.views.main), 37
export_csv() (seed.lib.exporter.Exporter method), 94
export_generation_pct_complete (seed.data_importer.models.ImportFile attribute), 75
EXPORT_PCT_COMPLETE_CACHE_KEY (seed.data_importer.models.ImportFile at-

tribute), 75
EXPORT_QUEUED_CACHE_KEY (seed.data_importer.models.ImportFile attribute), 75
export_ready (seed.data_importer.models.ImportFile attribute), 75
EXPORT_READY_CACHE_KEY (seed.data_importer.models.ImportFile attribute), 75
export_url (seed.data_importer.models.ImportFile attribute), 75
export_xls() (seed.lib.exporter.Exporter method), 94
Exporter (class in seed.lib.exporter), 94
extra_data (seed.models.BuildingSnapshot attribute), 116
extra_data_sources (seed.models.BuildingSnapshot attribute), 116
extra_kwargs (seed.serializers.labels.LabelSerializer.Meta attribute), 141

F

FakeClient (class in seed.tests.util), 110
FakeRequest (class in seed.tests.util), 110
field (seed.public.models.SharedBuildingField attribute), 98
fields (seed.serializers.labels.LabelSerializer.Meta attribute), 141
fields_from_queryset() (seed.lib.exporter.Exporter static method), 94
fields_to_save (seed.data_importer.models.TableColumnMapping attribute), 81
filename() (seed.lib.exporter.Exporter method), 94
filename_only (seed.data_importer.models.ImportFile attribute), 75
files (seed.data_importer.models.ImportRecord attribute), 78
filter_other_params() (in module seed.search), 136
find_canonical_building_values() (in module seed.models), 134
find_duplicates() (in module seed.common.util), 71
find_unmatched_buildings() (in module seed.models), 134
first_five_rows (seed.data_importer.models.TableColumnMapping attribute), 81
first_row (seed.data_importer.models.TableColumnMapping attribute), 81
first_row_columns (seed.data_importer.models.ImportFile attribute), 75
force_restart_cleaning_url (seed.data_importer.models.ImportFile attribute), 76
form (seed.data_importer.models.ImportRecord attribute), 78
format_api_docstring() (in module seed.utils.api), 142
friendly_destination_field (seed.data_importer.models.TableColumnMapping attribute), 81
friendly_destination_model (seed.data_importer.models.TableColumnMapping attribute), 81
friendly_destination_model_and_field (seed.data_importer.models.TableColumnMapping attribute), 81
from_portfolio_manager (seed.data_importer.models.ImportFile attribute), 76

G

generate_chomsky() (in module seed.test_helpers.factory.lib.chomsky), 100
generate_key() (seed.landing.models.SEEDUser method), 91
generate_paginated_results() (in module seed.search), 136
generate_url (seed.data_importer.models.ImportFile attribute), 76
generation_date_source (seed.models.BuildingSnapshot attribute), 116
GET (seed.tests.util.FakeRequest attribute), 110
get() (seed.common.mapper.Mapping method), 70
get() (seed.tests.util.FakeClient method), 110
get_absolute_url() (seed.landing.models.SEEDUser method), 91
get_adding_buildings_to_project_status_percentage() (in module seed.views.projects), 174
get_aggregated_building_report_data() (in module seed.views.main), 37
get_all_urls() (in module seed.utils.api), 142
get_ancestors() (in module seed.models), 134
get_api_endpoints() (in module seed.utils.api), 142
get_api_request_user() (in module seed.utils.api), 142
get_api_schema() (in module seed.views.api), 24
get_attr_source() (in module seed.mappings.mapper), 95
get_audit_type_display() (seed.audit_logs.models.AuditLog method), 63
get_building() (in module seed.views.main), 39
get_building_attrs() (in module seed.mappings.mapper), 95
get_building_fieldnames() (in module seed.search), 137
get_building_logs() (in module seed.audit_logs.views), 66
get_building_report_data() (in module seed.views.main), 40
get_building_summary_report_data() (in module seed.views.main), 42
get_buildings_for_user_count() (in module seed.utils.buildings), 142
get_cache_increment_value() (in module seed.tasks), 139
get_canonical_id_matches() (in module seed.tasks), 139

get_canonical_snapshots() (in module seed.tasks), 139
get_category_display() (seed.cleansing.models.Rules method), 68
get_cleansing_results() (in module seed.cleansing.views), 69
get_cleansing_rules() (in module seed.views.accounts), 26
get_color_display() (seed.models.StatusLabel method), 132
get_column_mapping() (in module seed.models), 134
get_column_mapping_suggestions() (in module seed.views.main), 43
get_column_mappings() (in module seed.models), 134
get_columns() (in module seed.utils.buildings), 142
get_compliance() (seed.models.Project method), 130
get_compliance_type_display() (seed.models.Compliance method), 126
get_coparents() (in module seed.views.main), 43
get_core_pk_column() (in module seed.data_importer.utils), 84
get_csv() (in module seed.cleansing.views), 69
get_dataset() (in module seed.views.main), 44
get_datasets() (in module seed.views.main), 44
get_datasets_count() (in module seed.views.main), 45
get_energy_type_display() (seed.models.Meter method), 128
get_energy_units_display() (seed.models.Meter method), 128
get_field_name_from_model() (in module seed.lib.exporter), 95
get_field_type_display() (seed.public.models.SharedBuildingField method), 98
get_field_value_from_instance() (in module seed.lib.exporter), 95
get_first_five_rows() (in module seed.views.main), 45
get_full_name() (seed.landing.models.SEEDUser method), 91
get_import_file() (in module seed.views.main), 45
get_is_applied() (seed.serializers.labels.LabelSerializer method), 141
get_lock_time() (in module seed.data_importer.utils), 84
get_mappable_columns() (in module seed.utils.mapping), 143
get_mappable_types() (in module seed.utils.mapping), 143
get_match_tree() (in module seed.views.main), 46
get_match_type_display() (seed.models.BuildingSnapshot method), 117
get_meters() (in module seed.views.meters), 173
get_next_by_created() (seed.audit_logs.models.AuditLog method), 63
get_next_by_created() (seed.data_importer.models.ImportFile method), 76
get_next_by_created() (seed.models.BuildingSnapshot method), 117
get_next_by_created() (seed.models.Compliance method), 126
get_next_by_created() (seed.models.Project method), 130
get_next_by_created() (seed.models.ProjectBuilding method), 131
get_next_by_created() (seed.models.StatusLabel method), 132
get_next_by_created() (seed.public.models.SharedBuildingField method), 98
get_next_by_date_joined() (seed.landing.models.SEEDUser method), 91
get_next_by_modified() (seed.audit_logs.models.AuditLog method), 63
get_next_by_modified() (seed.data_importer.models.ImportFile method), 76
get_next_by_modified() (seed.models.BuildingSnapshot method), 117
get_next_by_modified() (seed.models.Compliance method), 126
get_next_by_modified() (seed.models.Project method), 130
get_next_by_modified() (seed.models.ProjectBuilding method), 131
get_next_by_modified() (seed.models.StatusLabel method), 132
get_next_by_modified() (seed.public.models.SharedBuildingField method), 98
get_old_create_canonical() (in module seed.models), 134
get_organization() (in module seed.views.accounts), 26
get_organizations() (in module seed.views.accounts), 27
get_organizations_users() (in module seed.views.accounts), 27
get_orgs_w_public_fields() (in module seed.search), 137
get_PM_filter_by_counts() (in module seed.views.main), 37
get_pm_mapping() (in module seed.common.mapper), 71
get_previous_by_created() (seed.audit_logs.models.AuditLog method), 63
get_previous_by_created() (seed.data_importer.models.ImportFile method), 76
get_previous_by_created() (seed.models.BuildingSnapshot method), 117
get_previous_by_created() (seed.models.Compliance method), 126
get_previous_by_created() (seed.models.Project method), 130
get_previous_by_created() (seed.models.ProjectBuilding method), 131

get_previous_by_created() (seed.models.StatusLabel method), 132
get_previous_by_created() (seed.public.models.SharedBuildingField method), 98
get_previous_by_date_joined() (seed.landing.models.SEEDUser method), 91
get_previous_by_modified() (seed.audit_logs.models.AuditLog method), 63
get_previous_by_modified() (seed.data_importer.models.ImportFile method), 76
get_previous_by_modified() (seed.models.BuildingSnapshot method), 117
get_previous_by_modified() (seed.models.Compliance method), 126
get_previous_by_modified() (seed.models.Project method), 130
get_previous_by_modified() (seed.models.ProjectBuilding method), 131
get_previous_by_modified() (seed.models.StatusLabel method), 132
get_previous_by_modified() (seed.public.models.SharedBuildingField method), 98
get_prog_key() (in module seed.decorators), 112
get_progress() (in module seed.cleansing.views), 69
get_project() (in module seed.views.projects), 174
get_projects() (in module seed.utils.projects), 143
get_projects() (in module seed.views.projects), 175
get_projects_count() (in module seed.views.projects), 175
get_query_threshold() (in module seed.views.accounts), 28
get_queryset() (seed.audit_logs.models.AuditLogManager method), 64
get_queryset() (seed.data_importer.managers.NotDeletedManager method), 73
get_queryset() (seed.managers.json.JsonManager method), 97
get_queryset() (seed.models.CanonicalManager method), 124
get_raw_column_names() (in module seed.views.main), 47
get_search_query() (in module seed.utils.buildings), 142
get_severity_display() (seed.cleansing.models.Rules method), 68
get_shared_fields() (in module seed.views.accounts), 28
get_short_name() (seed.landing.models.SEEDUser method), 91
get_source_id() (in module seed.mappings.mapper), 95
get_source_type() (in module seed.utils.buildings), 142
get_source_type_display() (seed.models.BuildingSnapshot method), 117
get_source_type_display() (seed.models.ColumnMapping method), 125
get_sourced_attributes() (in module seed.models), 135
get_status_display() (seed.data_importer.models.ImportRecord method), 78
get_status_display() (seed.models.Project method), 130
get_timeseries() (in module seed.views.meters), 173
get_transfer_buildings() (in module seed.utils.projects), 144
get_type_display() (seed.cleansing.models.Rules method), 68
get_unit_type_display() (seed.models.Unit method), 133
get_upload_details() (in module seed.data_importer.views), 32
get_user_profile() (in module seed.views.accounts), 29
get_value_source_display() (seed.models.AttributeOption method), 112
GetDatasetsViewsTests (class in seed.tests.test_views), 107
GRAY_CHOICE (seed.models.StatusLabel attribute), 132
GREEN_CHOICE (seed.models.StatusLabel attribute), 132
GreenButtonXMLImportTests (class in seed.green_button.tests.test_xml_importer), 86
GreenButtonXMLParsingTests (class in seed.green_button.tests.test_xml_importer), 86
gross_floor_area_source (seed.models.BuildingSnapshot attribute), 117
groups (seed.landing.models.SEEDUser attribute), 91

H

handle() (seed.landing.management.commands.update_eula.Command method), 89
handle() (seed.management.commands.set_s3_expires_headers_for_angular method), 99
handle_id_matches() (in module seed.tasks), 139
handle_results() (in module seed.tasks), 139
handle_s3_upload_complete() (in module seed.data_importer.views), 32
has_children (seed.models.BuildingSnapshot attribute), 117
has_compliance (seed.models.Project attribute), 130
help (seed.landing.management.commands.update_eula.Command attribute), 89
help (seed.management.commands.set_s3_expires_headers_for_angularjs_p attribute), 99

I

import_file (seed.data_importer.models.TableColumnMapping attribute), 81
import_file (seed.models.BuildingSnapshot attribute), 117
import_record (seed.data_importer.models.BuildingImportRecord attribute), 74
import_record (seed.data_importer.models.ImportFile attribute), 76
import_xml() (in module seed.green_button.xml_importer), 88
ImportFile (class in seed.data_importer.models), 74
ImportFile.DoesNotExist, 75
ImportFile.MultipleObjectsReturned, 75
importfile_set (seed.data_importer.models.ImportRecord attribute), 78
ImportFileViewsTests (class in seed.tests.test_views), 107
ImportRecord (class in seed.data_importer.models), 77
ImportRecord.DoesNotExist, 77
ImportRecord.MultipleObjectsReturned, 77
importrecord_set (seed.landing.models.SEEDUser attribute), 91
in_range_checking() (seed.cleansing.models.Cleansing method), 67
INACTIVE_STATUS (seed.models.Project attribute), 129
initialize_cache() (seed.cleansing.models.Cleansing static method), 68
initialize_canonical_building() (in module seed.models), 135
initialize_rules() (seed.cleansing.models.Rules static method), 68
interval_block_data() (in module seed.green_button.xml_importer), 88
interval_data() (in module seed.green_button.xml_importer), 88
invalid_test_cc_number() (seed.test_helpers.factory.helpers.DjangoFunctionalFactory class method), 101
is_concatenated() (seed.models.ColumnMapping method), 125
is_direct() (seed.models.ColumnMapping method), 125
is_mapped (seed.data_importer.models.TableColumnMapping attribute), 82
is_mapping_or_cleaning (seed.data_importer.models.ImportRecord attribute), 78
is_not_in_progress (seed.data_importer.models.ImportRecord attribute), 78
is_not_whitelist_building() (in module seed.search), 137
is_same_snapshot() (in module seed.tasks), 140

J

json_order_by() (seed.managers.json.JsonQuerySet method), 97
JsonManager (class in seed.managers.json), 97
JsonQuerySet (class in seed.managers.json), 97

K

keys() (seed.common.mapper.Mapping method), 70

L

labels (seed.models.CanonicalBuilding attribute), 123
LabelSerializer (class in seed.serializers.labels), 141
LabelSerializer.Meta (class in seed.serializers.labels), 141
landing_page() (in module seed.landing.views), 93
last_modified_by (seed.data_importer.models.ImportRecord attribute), 78
last_modified_by (seed.models.BuildingSnapshot attribute), 117
last_modified_by (seed.models.Project attribute), 130
last_modified_user (seed.landing.models.SEEDUser attribute), 91
LIGHT_BLUE_CHOICE (seed.models.StatusLabel attribute), 132
local_file (seed.data_importer.models.ImportFile attribute), 76
local_uploader() (in module seed.data_importer.views), 32
lock_and_track() (in module seed.decorators), 112
locked (seed.tests.test_decorators.TestDecorators attribute), 102
log_action() (seed.audit_logs.models.AuditLogManager method), 64
logentry_set (seed.landing.models.SEEDUser attribute), 91
login_view() (in module seed.landing.views), 93
LoginForm (class in seed.landing.forms), 89
lookup_hash() (seed.data_importer.utils.CoercionRobot method), 83
lot_number_source (seed.models.BuildingSnapshot attribute), 117

M

MainViewTests (class in seed.tests.test_views), 107
make_fake_mappings() (in module seed.tests.util), 110
make_fake_snapshot() (in module seed.tests.util), 110
make_key() (seed.data_importer.utils.CoercionRobot method), 83
make_token() (seed.token_generators.SignupTokenGenerator method), 140
MapItem (class in seed.common.mapper), 70
mapped_mappings (seed.models.Column attribute), 124
Mapping (class in seed.common.mapper), 70
MAPPING_ACTIVE_KEY (seed.data_importer.models.ImportRecord attribute), 77

MAPPING_QUEUED_KEY (seed.data_importer.models.ImportRecord attribute), 77
MappingConfiguration (class in seed.common.mapper), 70
mark_merge_started() (seed.data_importer.models.ImportRecord method), 78
mark_merged() (seed.data_importer.models.ImportRecord method), 78
mask_results() (in module seed.search), 137
match_progress_key (seed.data_importer.models.ImportRecord attribute), 78
matched_buildings (seed.data_importer.models.ImportRecord attribute), 78
MatchTreeTests (class in seed.tests.test_views), 107
media (seed.landing.forms.LoginForm attribute), 89
merge_building() (in module seed.mappings.mapper), 95
merge_extra_data() (in module seed.mappings.mapper), 96
merge_progress_key (seed.data_importer.models.ImportRecord attribute), 78
merge_progress_url (seed.data_importer.models.ImportFile attribute), 76
merge_progress_url (seed.data_importer.models.ImportRecord attribute), 78
merge_seconds_remaining_key (seed.data_importer.models.ImportRecord attribute), 78
merge_status (seed.data_importer.models.ImportRecord attribute), 78
merge_status_key (seed.data_importer.models.ImportRecord attribute), 78
merge_url (seed.data_importer.models.ImportRecord attribute), 78
META (seed.tests.util.FakeRequest attribute), 110
META_BEDES (seed.common.mapper.Mapping attribute), 70
META_NUMERIC (seed.common.mapper.Mapping attribute), 70
META_TYPE (seed.common.mapper.Mapping attribute), 70
Meter (class in seed.models), 128
meter (seed.models.TimeSeries attribute), 133
Meter.DoesNotExist, 128
Meter.MultipleObjectsReturned, 128
meter_data() (in module seed.green_button.xml_importer), 88
meters (seed.models.BuildingSnapshot attribute), 117
missing_buildings (seed.data_importer.models.ImportRecord attribute), 78
missing_matching_field() (seed.cleansing.models.Cleansing method), 68
missing_request_keys() (in module seed.common.views), 71
missing_values() (seed.cleansing.models.Cleansing method), 68
model (seed.serializers.labels.LabelSerializer.Meta attribute), 141
modified_import_records (seed.landing.models.SEEDUser attribute), 92
move_buildings() (in module seed.utils.projects), 144
move_buildings() (in module seed.views.projects), 176

N

new_buildings (seed.data_importer.models.ImportRecord attribute), 78
NonCanonicalProjectBuildings (class in seed.models), 129
NonCanonicalProjectBuildings.DoesNotExist, 129
NonCanonicalProjectBuildings.MultipleObjectsReturned, 129
noncanonicalprojectbuildings_set (seed.models.ProjectBuilding attribute), 131
NotDeletableModel (class in seed.data_importer.models), 80
NotDeletableModel.Meta (class in seed.data_importer.models), 80
NotDeletedManager (class in seed.data_importer.managers), 73
num_buildings_imported_total (seed.data_importer.models.ImportRecord attribute), 79
num_cells (seed.data_importer.models.ImportFile attribute), 76
num_cleaning_complete (seed.data_importer.models.ImportFile attribute), 76
num_cleaning_remaining (seed.data_importer.models.ImportFile attribute), 76
num_cleaning_total (seed.data_importer.models.ImportFile attribute), 76
num_coercion_errors (seed.data_importer.models.ImportRecord attribute), 79
num_columns (seed.data_importer.models.ImportRecord attribute), 79
num_failed_tablecolumnmappings (seed.data_importer.models.ImportFile attribute), 76
num_failed_tablecolumnmappings (seed.data_importer.models.ImportRecord attribute), 79
num_files (seed.data_importer.models.ImportRecord attribute), 79

num_files_cleaned (seed.data_importer.models.ImportRecord attribute), 79
num_files_mapped (seed.data_importer.models.ImportRecord attribute), 79
num_files_merged (seed.data_importer.models.ImportRecord attribute), 79
num_files_to_clean (seed.data_importer.models.ImportRecord attribute), 79
num_files_to_map (seed.data_importer.models.ImportRecord attribute), 79
num_files_to_merge (seed.data_importer.models.ImportRecord attribute), 79
num_mapping_complete (seed.data_importer.models.ImportFile attribute), 76
num_mapping_remaining (seed.data_importer.models.ImportFile attribute), 76
num_mapping_total (seed.data_importer.models.ImportFile attribute), 76
num_matched_buildings (seed.data_importer.models.ImportRecord attribute), 79
num_missing_buildings (seed.data_importer.models.ImportRecord attribute), 79
num_new_buildings (seed.data_importer.models.ImportRecord attribute), 79
num_not_ready_for_import (seed.data_importer.models.ImportRecord attribute), 79
num_ready_for_import (seed.data_importer.models.ImportRecord attribute), 79
num_rows (seed.data_importer.models.ImportRecord attribute), 79
num_validation_errors (seed.data_importer.models.ImportRecord attribute), 79

O

obj_to_dict() (in module seed.models), 135
objects (seed.audit_logs.models.AuditLog attribute), 63
objects (seed.cleansing.models.Rules attribute), 68
objects (seed.data_importer.models.BuildingImportRecord attribute), 74
objects (seed.data_importer.models.DataCoercionMapping attribute), 74
objects (seed.data_importer.models.ImportFile attribute), 76
objects (seed.data_importer.models.ImportRecord attribute), 79
objects (seed.data_importer.models.RangeValidationRule attribute), 81
objects (seed.data_importer.models.TableColumnMapping attribute), 82
objects (seed.data_importer.models.ValidationOutlier attribute), 82
objects (seed.data_importer.models.ValidationRule attribute), 82
objects (seed.landing.models.SEEDUser attribute), 92
objects (seed.models.AttributeOption attribute), 112
objects (seed.models.BuildingAttributeVariant attribute), 113
objects (seed.models.BuildingSnapshot attribute), 118
objects (seed.models.CanonicalBuilding attribute), 123
objects (seed.models.Column attribute), 124
objects (seed.models.ColumnMapping attribute), 125
objects (seed.models.Compliance attribute), 126
objects (seed.models.CustomBuildingHeaders attribute), 127
objects (seed.models.Enum attribute), 128
objects (seed.models.EnumValue attribute), 128
objects (seed.models.Meter attribute), 128
objects (seed.models.NonCanonicalProjectBuildings attribute), 129
objects (seed.models.Project attribute), 130
objects (seed.models.ProjectBuilding attribute), 131
objects (seed.models.StatusLabel attribute), 132
objects (seed.models.TimeSeries attribute), 133
objects (seed.models.Unit attribute), 133
objects (seed.public.models.SharedBuildingField attribute), 98
occupied_floor_area_source (seed.models.BuildingSnapshot attribute), 118
option_list (seed.management.commands.set_s3_expires_headers_for_angu attribute), 99
options (seed.models.BuildingAttributeVariant attribute), 113
ORANGE_CHOICE (seed.models.StatusLabel attribute), 132
orchestrate_search_filter_sort() (in module seed.search), 137
org (seed.cleansing.models.Rules attribute), 68
org (seed.public.models.SharedBuildingField attribute), 98
organization (seed.audit_logs.models.AuditLog attribute), 64
organization (seed.models.Column attribute), 124
organizationuser_set (seed.landing.models.SEEDUser attribute), 92
orgs (seed.landing.models.SEEDUser attribute), 92
owner (seed.data_importer.models.ImportRecord attribute), 79
owner (seed.models.Project attribute), 130
owner_address_source (seed.models.BuildingSnapshot attribute), 118
owner_city_state_source (seed.models.BuildingSnapshot attribute), 118
owner_email_source (seed.models.BuildingSnapshot attribute), 118

owner_postal_code_source (seed.models.BuildingSnapshot attribute), 118
owner_source (seed.models.BuildingSnapshot attribute), 118
owner_telephone_source (seed.models.BuildingSnapshot attribute), 119

P

paginate_results() (in module seed.search), 137
parent_tree (seed.models.BuildingSnapshot attribute), 119
parents (seed.models.BuildingSnapshot attribute), 119
parse_body() (in module seed.search), 137
parse_datetime() (in module seed.utils.time), 145
password_reset() (in module seed.landing.views), 94
password_reset_complete() (in module seed.landing.views), 94
password_reset_confirm() (in module seed.landing.views), 94
password_reset_done() (in module seed.landing.views), 94
password_set() (in module seed.landing.views), 94
path (seed.tests.util.FakeRequest attribute), 110
pct_merge_complete (seed.data_importer.models.ImportRecord attribute), 79
pct_premerge_complete (seed.data_importer.models.ImportRecord attribute), 79
percent_files_cleaned (seed.data_importer.models.ImportRecord attribute), 79
percent_files_mapped (seed.data_importer.models.ImportRecord attribute), 79
percent_files_ready_to_merge (seed.data_importer.models.ImportRecord attribute), 79
percent_ready_for_import (seed.data_importer.models.ImportRecord attribute), 79
percent_ready_for_import_by_file_count (seed.data_importer.models.ImportRecord attribute), 79
pk (seed.tests.test_decorators.TestDecorators attribute), 102
PM (seed.common.mapper.Programs attribute), 71
pm() (seed.common.mapper.MappingConfiguration method), 70
pm_property_id_source (seed.models.BuildingSnapshot attribute), 119
POST (seed.tests.util.FakeRequest attribute), 110
post() (seed.tests.util.FakeClient method), 110
postal_code_source (seed.models.BuildingSnapshot attribute), 119
pre_merge_url (seed.data_importer.models.ImportRecord attribute), 79
prefixed_pk() (seed.data_importer.models.ImportRecord method), 79
premerge_estimated_seconds_remaining (seed.data_importer.models.ImportRecord attribute), 79
premerge_progress_key (seed.data_importer.models.ImportRecord attribute), 80
premerge_progress_url (seed.data_importer.models.ImportFile attribute), 76
premerge_progress_url (seed.data_importer.models.ImportRecord attribute), 80
premerge_seconds_remaining_key (seed.data_importer.models.ImportRecord attribute), 80
PRIMARY (seed.managers.json.JsonQuerySet attribute), 97
process_search_params() (in module seed.search), 138
process_view() (seed.utils.api.APIBypassCSRFMiddleware method), 141
Programs (class in seed.common.mapper), 71
progress() (in module seed.views.main), 47
Project (class in seed.models), 129
project (seed.models.Compliance attribute), 126
project (seed.models.ProjectBuilding attribute), 131
Project.DoesNotExist, 129
Project.MultipleObjectsReturned, 129
project_building_snapshots (seed.models.BuildingSnapshot attribute), 119
project_building_snapshots (seed.models.Project attribute), 130
project_set (seed.landing.models.SEEDUser attribute), 92
project_set (seed.models.BuildingSnapshot attribute), 119
ProjectBuilding (class in seed.models), 131
projectbuilding (seed.models.NonCanonicalProjectBuildings attribute), 129
ProjectBuilding.DoesNotExist, 131
ProjectBuilding.MultipleObjectsReturned, 131
projectbuilding_set (seed.landing.models.SEEDUser attribute), 92
ProjectTestCase (class in seed.tests.tests), 110
property_name_source (seed.models.BuildingSnapshot attribute), 120
property_notes_source (seed.models.BuildingSnapshot attribute), 120
prune_data() (seed.cleansing.models.Cleansing method), 68

Q

qs_to_rows() (in module seed.lib.exporter), 95
queue_update_status_for_import_record() (in module seed.data_importer.models), 83

QUEUED_TCM_DATA_KEY (seed.data_importer.models.ImportFile attribute), 75
QUEUED_TCM_SAVE_COUNTER_KEY (seed.data_importer.models.ImportFile attribute), 75

R

rand_bool() (seed.test_helpers.factory.helpers.DjangoFunctionalFactory class method), 101
rand_city() (seed.test_helpers.factory.helpers.DjangoFunctionalFactory class method), 101
rand_city_suffix() (seed.test_helpers.factory.helpers.DjangoFunctionalFactory class method), 101
rand_currency() (seed.test_helpers.factory.helpers.DjangoFunctionalFactory class method), 101
rand_date() (seed.test_helpers.factory.helpers.DjangoFunctionalFactory class method), 101
rand_domain() (seed.test_helpers.factory.helpers.DjangoFunctionalFactory class method), 101
rand_email() (seed.test_helpers.factory.helpers.DjangoFunctionalFactory class method), 101
rand_float() (seed.test_helpers.factory.helpers.DjangoFunctionalFactory class method), 101
rand_int() (seed.test_helpers.factory.helpers.DjangoFunctionalFactory class method), 101
rand_name() (seed.test_helpers.factory.helpers.DjangoFunctionalFactory class method), 101
rand_phone() (seed.test_helpers.factory.helpers.DjangoFunctionalFactory class method), 101
rand_plant_name() (seed.test_helpers.factory.helpers.DjangoFunctionalFactory class method), 101
rand_str() (seed.test_helpers.factory.helpers.DjangoFunctionalFactory class method), 101
rand_street_address() (seed.test_helpers.factory.helpers.DjangoFunctionalFactory class method), 101
rand_street_suffix() (seed.test_helpers.factory.helpers.DjangoFunctionalFactory class method), 101
random_conversation() (seed.test_helpers.factory.helpers.DjangoFunctionalFactory class method), 101
RangeValidationRule (class in seed.data_importer.models), 80
rangevalidationrule (seed.data_importer.models.ValidationRule attribute), 82
RangeValidationRule.DoesNotExist, 80
RangeValidationRule.MultipleObjectsReturned, 80
raw_columns_expected (seed.tests.test_views.TestMCMViews attribute), 109
raw_mappings (seed.models.Column attribute), 124
raw_objects (seed.data_importer.models.ImportFile attribute), 76
raw_objects (seed.data_importer.models.ImportRecord attribute), 80
raw_objects (seed.models.CanonicalBuilding attribute), 123
ready_for_import (seed.data_importer.models.ImportRecord attribute), 80
ready_to_import (seed.data_importer.models.ImportFile attribute), 76
recent_sale_date_source (seed.models.BuildingSnapshot attribute), 120
recurse_tree() (seed.models.BuildingSnapshot method), 120
RED_CHOICE (seed.models.StatusLabel attribute), 132
release_date_source (seed.models.BuildingSnapshot attribute), 120
release_lock() (in module seed.data_importer.utils), 84
remap_buildings() (in module seed.views.main), 47
remove_buildings_from_project() (in module seed.views.projects), 176
remove_duplicates() (seed.models.ColumnMapping method), 125
remove_results_below_q_threshold() (in module seed.search), 138
remove_user_from_org() (in module seed.views.accounts), 29
removing_buildings_status_percentage_cache_key (seed.models.Project attribute), 130
render() (seed.templatetags.breadcrumbs.BreadcrumbNode method), 99
render() (seed.templatetags.breadcrumbs.UrlBreadcrumbNode method), 99
ReportViewsTests (class in seed.tests.test_views), 108
require_organization_id() (in module seed.decorators), 112
REQUIRED_FIELDS (seed.landing.models.SEEDUser attribute), 90
RequireOrganizationIDTests (class in seed.tests.test_decorators), 102
reset_cleansing_rules() (in module seed.views.accounts), 29
reset_results() (seed.cleansing.models.Cleansing method), 68
restore_defaults() (seed.cleansing.models.Rules static method), 69
RETRO_COMMISSIONING_COMPLIANCE_CHOICE (seed.models.Compliance attribute), 126
robots_txt() (in module config.views), 72
rule (seed.data_importer.models.ValidationOutlier attribute), 82
Rules (class in seed.cleansing.models), 68
Rules.DoesNotExist, 68
Rules.MultipleObjectsReturned, 68

S

save() (config.storage.CachedS3BotoStorage method), 72
save() (seed.audit_logs.models.AuditLog method), 64

save() (seed.data_importer.models.DataCoercionMapping method), 74
save() (seed.data_importer.models.ImportFile method), 76
save() (seed.data_importer.models.TableColumnMapping method), 82
save() (seed.landing.models.SEEDUser method), 93
save() (seed.models.BuildingSnapshot method), 120
save() (seed.models.ColumnMapping method), 126
save_cleansing_rules() (in module seed.views.accounts), 29
save_column_mappings() (in module seed.views.main), 47
save_column_names() (in module seed.models), 135
SAVE_COUNTER_CACHE_KEY (seed.data_importer.models.ImportFile attribute), 75
save_import_meta_url (seed.data_importer.models.ImportRecord attribute), 80
save_match() (in module seed.views.main), 48
save_org_settings() (in module seed.views.accounts), 30
save_raw_data() (in module seed.views.main), 48
save_snapshot_match() (in module seed.models), 135
save_to_cache() (seed.cleansing.models.Cleansing method), 68
save_variant() (in module seed.mappings.mapper), 96
search_building_snapshots() (in module seed.views.main), 48
search_buildings() (in module seed.search), 138
search_buildings() (in module seed.views.main), 49
search_public_buildings() (in module seed.search), 139
search_url (seed.data_importer.models.ImportRecord attribute), 80
SearchBuildingSnapshotsViewTests (class in seed.tests.test_views), 108
SearchViewTests (class in seed.tests.test_views), 108
second_to_fifth_rows (seed.data_importer.models.ImportFile attribute), 76
seed (module), 140
seed.audit_logs.models (module), 63
seed.audit_logs.tests (module), 64
seed.audit_logs.urls (module), 65
seed.audit_logs.views (module), 65
seed.cleansing.models (module), 67
seed.cleansing.tasks (module), 69
seed.cleansing.tests (module), 69
seed.cleansing.urls (module), 69
seed.cleansing.views (module), 69
seed.common.mapper (module), 70
seed.common.util (module), 71
seed.common.views (module), 71
seed.data (module), 73
seed.data.bedes (module), 73
seed.data_importer (module), 86
seed.data_importer.managers (module), 73
seed.data_importer.models (module), 73
seed.data_importer.urls (module), 83
seed.data_importer.utils (module), 83
seed.data_importer.views (module), 32
seed.decorators (module), 111
seed.factory (module), 112
seed.features (module), 86
seed.green_button (module), 89
seed.green_button.tests (module), 87
seed.green_button.tests.test_xml_importer (module), 86
seed.green_button.xml_importer (module), 87
seed.landing (module), 94
seed.landing.forms (module), 89
seed.landing.management (module), 89
seed.landing.management.commands (module), 89
seed.landing.management.commands.update_eula (module), 89
seed.landing.models (module), 89
seed.landing.tests (module), 93
seed.landing.urls (module), 93
seed.landing.views (module), 93
seed.lib (module), 95
seed.lib.exporter (module), 94
seed.management (module), 99
seed.management.commands (module), 99
seed.management.commands.set_s3_expires_headers_for_angularjs_partial (module), 99
seed.managers (module), 97
seed.managers.json (module), 97
seed.managers.tests (module), 97
seed.managers.tests.test_json_manager (module), 97
seed.mappings (module), 97
seed.mappings.mapper (module), 95
seed.mappings.seed_mappings (module), 96
seed.models (module), 112
seed.public (module), 98
seed.public.models (module), 97
seed.search (module), 136
seed.serializers (module), 141
seed.serializers.celery (module), 140
seed.serializers.labels (module), 141
seed.tasks (module), 139
seed.templatetags.breadcrumbs (module), 99
seed.test_helpers (module), 101
seed.test_helpers.factory.helpers (module), 101
seed.test_helpers.factory.lib.chomsky (module), 100
seed.tests.test_admin_views (module), 102
seed.tests.test_decorators (module), 102
seed.tests.test_exporters (module), 103
seed.tests.test_models (module), 103
seed.tests.test_tasks (module), 104
seed.tests.test_views (module), 106
seed.tests.tests (module), 110

seed.tests.util (module), 110
seed.token_generators (module), 140
seed.urls (module), 140
seed.urls.accounts (module), 141
seed.urls.api (module), 141
seed.urls.main (module), 141
seed.urls.projects (module), 141
seed.utils (module), 140
seed.utils.api (module), 141
seed.utils.buildings (module), 142
seed.utils.constants (module), 143
seed.utils.mapping (module), 143
seed.utils.organizations (module), 143
seed.utils.projects (module), 143
seed.utils.time (module), 145
seed.views (module), 140, 177
seed.views.accounts (module), 24
seed.views.api (module), 24
seed.views.main (module), 33
seed.views.meters (module), 172
seed.views.projects (module), 173
seed_decoder() (seed.serializers.celery.CeleryDatetimeSerializer static method), 140
seed_dumps() (seed.serializers.celery.CeleryDatetimeSerializer static method), 140
seed_loads() (seed.serializers.celery.CeleryDatetimeSerializer static method), 140
SEEDFactory (class in seed.factory), 112
SEEDUser (class in seed.landing.models), 89
SEEDUser.DoesNotExist, 90
SEEDUser.MultipleObjectsReturned, 90
sentry_js() (in module config.template_context), 72
serialize_building_snapshot() (in module seed.utils.buildings), 142
session_key() (in module config.template_context), 72
set_initial_sources() (in module seed.models), 135
setUp() (seed.audit_logs.tests.AuditLogModelTests method), 64
setUp() (seed.audit_logs.tests.AuditLogViewTests method), 65
setUp() (seed.cleansing.tests.CleansingDataSample method), 69
setUp() (seed.cleansing.tests.CleansingDataTestCoveredBuilding method), 69
setUp() (seed.cleansing.tests.CleansingDataTestPM method), 69
setUp() (seed.cleansing.tests.CleansingViewTests method), 69
setUp() (seed.green_button.tests.test_xml_importer.GreenButtonImportTests method), 86
setUp() (seed.green_button.tests.test_xml_importer.GreenButtonXMLParsing method), 86
setUp() (seed.landing.tests.UserLoginTest method), 93
setUp() (seed.managers.tests.test_json_manager.TestJsonManager method), 97
setUp() (seed.tests.test_admin_views.AdminViewsTest method), 102
setUp() (seed.tests.test_decorators.RequireOrganizationIDTests method), 102
setUp() (seed.tests.test_decorators.TestDecorators method), 102
setUp() (seed.tests.test_exporters.TestExporters method), 103
setUp() (seed.tests.test_models.TestBuildingSnapshot method), 103
setUp() (seed.tests.test_models.TestColumnMapping method), 104
setUp() (seed.tests.test_tasks.TestCleaner method), 105
setUp() (seed.tests.test_tasks.TestTasks method), 105
setUp() (seed.tests.test_tasks.TestTasksXLS method), 106
setUp() (seed.tests.test_tasks.TestTasksXLSX method), 106
setUp() (seed.tests.test_views.BuildingDetailViewTests method), 106
setUp() (seed.tests.test_views.DataImporterViewTests method), 106
setUp() (seed.tests.test_views.DefaultColumnsViewTests method), 107
setUp() (seed.tests.test_views.GetDatasetsViewsTests method), 107
setUp() (seed.tests.test_views.ImportFileViewsTests method), 107
setUp() (seed.tests.test_views.MainViewTests method), 107
setUp() (seed.tests.test_views.MatchTreeTests method), 108
setUp() (seed.tests.test_views.ReportViewsTests method), 108
setUp() (seed.tests.test_views.SearchBuildingSnapshotsViewTests method), 108
setUp() (seed.tests.test_views.SearchViewTests method), 108
setUp() (seed.tests.test_views.TestMCMViews method), 109
setUp() (seed.tests.tests.UtilsTests method), 110
SharedBuildingField (class in seed.public.models), 97
SharedBuildingField.DoesNotExist, 98
SharedBuildingField.MultipleObjectsReturned, 98
sign_policy_document() (in module seed.data_importer.views), 33
signup() (in module seed.landing.views), 94
SignupTokenGenerator (class in seed.token_generators), 140
site_eui (seed.models.BuildingSnapshot attribute), 120
site_eui_weather_normalized_source (seed.models.BuildingSnapshot attribute),

120
source_eui_source (seed.models.BuildingSnapshot attribute), 121
source_eui_weather_normalized_source (seed.models.BuildingSnapshot attribute), 121
source_string_sha (seed.data_importer.models.DataCoercionMapping attribute), 74
source_string_sha (seed.data_importer.models.TableColumnMapping attribute), 82
space_alerts_source (seed.models.BuildingSnapshot attribute), 121
start_mapping() (in module seed.views.main), 50
start_merge_url (seed.data_importer.models.ImportRecord attribute), 80
start_system_matching() (in module seed.views.main), 50
state_province_source (seed.models.BuildingSnapshot attribute), 121
STATUS_CHOICES (seed.models.Project attribute), 129
status_denominator (seed.data_importer.models.ImportRecord attribute), 80
status_is_live (seed.data_importer.models.ImportRecord attribute), 80
status_numerator (seed.data_importer.models.ImportRecord attribute), 80
status_percent (seed.data_importer.models.ImportRecord attribute), 80
status_url (seed.data_importer.models.ImportRecord attribute), 80
StatusLabel (class in seed.models), 132
StatusLabel.DoesNotExist, 132
StatusLabel.MultipleObjectsReturned, 132
subdirectory() (seed.lib.exporter.Exporter method), 94
subdirectory_from_export_id() (seed.lib.exporter.Exporter static method), 94
summary_analysis_active (seed.data_importer.models.ImportRecord attribute), 80
SUMMARY_ANALYSIS_ACTIVE_KEY() (seed.data_importer.models.ImportRecord class method), 77
summary_analysis_queued (seed.data_importer.models.ImportRecord attribute), 80
SUMMARY_ANALYSIS_QUEUED_KEY() (seed.data_importer.models.ImportRecord class method), 77
super_organization (seed.data_importer.models.ImportRecord attribute), 80
super_organization (seed.models.BuildingSnapshot attribute), 121
super_organization (seed.models.ColumnMapping attribute), 126
super_organization (seed.models.CustomBuildingHeaders attribute), 127
super_organization (seed.models.Project attribute), 131
super_organization (seed.models.StatusLabel attribute), 132

T

TABLE (seed.managers.json.JsonQuerySet attribute), 97
table_column_mapping (seed.data_importer.models.DataCoercionMapping attribute), 74
table_column_mapping (seed.data_importer.models.ValidationRule attribute), 83
TableColumnMapping (class in seed.data_importer.models), 81
TableColumnMapping.DoesNotExist, 81
TableColumnMapping.MultipleObjectsReturned, 81
tablecolumnmapping_formset() (seed.data_importer.models.ImportFile method), 76
tablecolumnmapping_set (seed.data_importer.models.ImportFile attribute), 76
tablecolumnmappings (seed.data_importer.models.ImportFile attribute), 77
tablecolumnmappings_failed (seed.data_importer.models.ImportFile attribute), 77
tax_lot_id_source (seed.models.BuildingSnapshot attribute), 121
tcm_errors_json (seed.data_importer.models.ImportFile attribute), 77
tcm_fields_to_save (seed.data_importer.models.ImportFile attribute), 77
tcm_json (seed.data_importer.models.ImportFile attribute), 77
tearDown() (seed.green_button.tests.test_xml_importer.GreenButtonXMLI method), 86
tearDown() (seed.green_button.tests.test_xml_importer.GreenButtonXMLP method), 86
tearDown() (seed.tests.test_exporters.TestExporters method), 103
tearDown() (seed.tests.test_views.DefaultColumnsViewTests method), 107
tempfile (seed.lib.exporter.Exporter attribute), 95
test_add_org() (seed.tests.test_admin_views.AdminViewsTest method), 102
test_add_org_dupe() (seed.tests.test_admin_views.AdminViewsTest method), 102
test_add_user_existing_org() (seed.tests.test_admin_views.AdminViewsTest method), 102
test_add_user_new_org() (seed.tests.test_admin_views.AdminViewsTest method), 102

test_add_user_no_org() (seed.tests.test_admin_views.AdminViewsTests method), 102
test_as_collection() (seed.green_button.tests.test_xml_importer.GreenButtonXMLParsingTests method), 86
test_audit() (seed.audit_logs.tests.AuditLogModelTests method), 64
test_audit_save() (seed.audit_logs.tests.AuditLogModelTests method), 64
test_audit_update() (seed.audit_logs.tests.AuditLogModelTests method), 64
test_basic_compliance_creation() (seed.tests.tests.ComplianceTestCase method), 110
test_basic_project_creation() (seed.tests.tests.ProjectTestCase method), 110
test_build_cleaner() (seed.tests.test_tasks.TestCleaner method), 105
test_building_data() (seed.green_button.tests.test_xml_importer.GreenButtonXMLParsingTests method), 86
test_cached_first_row_order() (seed.tests.test_tasks.TestTasks method), 105
test_cc_number() (seed.test_helpers.factory.helpers.DjangoFunctionalFactory class method), 101
test_clean() (seed.tests.test_models.TestBuildingSnapshot method), 103
test_cleanse() (seed.cleansing.tests.CleansingDataSample method), 69
test_cleanse() (seed.cleansing.tests.CleansingDataTestCoveredBuilding method), 69
test_cleanse() (seed.cleansing.tests.CleansingDataTestPM method), 69
test_create_child() (seed.tests.test_models.TestBuildingSnapshot method), 103
test_create_dataset() (seed.tests.test_views.TestMCMViews method), 109
test_create_models() (seed.green_button.tests.test_xml_importer.GreenButtonXMLParsingTests method), 86
test_create_note() (seed.audit_logs.tests.AuditLogViewTests method), 65
test_create_pm_mapping() (seed.tests.test_views.MainViewTests method), 107
test_csv_export() (seed.tests.test_exporters.TestExporters method), 103
test_csv_export_extra_data() (seed.tests.test_exporters.TestExporters method), 103
test_data_model_assumptions() (seed.tests.test_exporters.TestExporters method), 103
test_delete_dataset() (seed.tests.test_views.GetDatasetsViewsTests method), 107
test_duplicates_from_import_file() (seed.tests.test_views.ImportFileViewsTests method), 107
test_delete_file() (seed.tests.test_views.ImportFileViewsTests method), 107
test_delete_organization() (seed.tests.test_tasks.TestTasks method), 105
test_delete_organization_buildings() (seed.tests.test_tasks.TestTasks method), 105
test_delete_organization_doesnt_delete_user_if_multiple_memberships() (seed.tests.test_tasks.TestTasks method), 105
test_energy_type() (seed.green_button.tests.test_xml_importer.GreenButton method), 87
test_energy_units() (seed.green_button.tests.test_xml_importer.GreenButton method), 87
test_export_buildings() (seed.tests.test_views.MainViewTests method), 107
test_export_buildings_empty() (seed.tests.test_views.MainViewTests method), 107
test_export_buildings_progress() (seed.tests.test_views.MainViewTests method), 107
test_generic_relation() (seed.audit_logs.tests.AuditLogModelTests method), 64
test_get_aggregated_building_report_data() (seed.tests.test_views.ReportViewsTests method), 108
test_get_all_audit_logs_for_an_org() (seed.audit_logs.tests.AuditLogModelTests method), 65
test_get_ancestors() (seed.tests.test_tasks.TestTasks method), 105
test_get_building() (seed.tests.test_views.BuildingDetailViewTests method), 106
test_get_building_includes_green_button() (seed.tests.test_views.BuildingDetailViewTests method), 106
test_get_building_logs() (seed.audit_logs.tests.AuditLogViewTests method), 65
test_get_building_report_data() (seed.tests.test_views.ReportViewsTests method), 108
test_get_building_summary_report_data() (seed.tests.test_views.ReportViewsTests method), 108
test_get_building_with_deleted_dataset() (seed.tests.test_views.BuildingDetailViewTests method), 106
test_get_building_with_project() (seed.tests.test_views.BuildingDetailViewTests method), 106

test_get_buildings_count_for_user() (seed.tests.tests.UtilsTests method), 110
test_get_cleansing_results() (seed.cleansing.tests.CleansingViewTests method), 69
test_get_column_mapping() (seed.tests.test_models.TestBuildingSnapshot method), 103
test_get_column_mapping_suggestions() (seed.tests.test_views.TestMCMViews method), 109
test_get_column_mapping_suggestions_pm_file() (seed.tests.test_views.TestMCMViews method), 109
test_get_column_mapping_suggestions_with_columns() (seed.tests.test_views.TestMCMViews method), 109
test_get_column_mappings() (seed.tests.test_models.TestBuildingSnapshot method), 103
test_get_columns() (seed.tests.test_views.DefaultColumnsViewTests method), 107
test_get_coparents() (seed.tests.test_views.MatchTreeTests method), 108
test_get_csv() (seed.cleansing.tests.CleansingViewTests method), 69
test_get_dataset() (seed.tests.test_views.GetDatasetsViewsTests method), 107
test_get_datasets() (seed.tests.test_views.GetDatasetsViewsTests method), 107
test_get_default_columns_initial_state() (seed.tests.test_views.DefaultColumnsViewTests method), 107
test_get_default_columns_with_set_columns() (seed.tests.test_views.DefaultColumnsViewTests method), 107
test_get_first_five_rows() (seed.tests.test_views.DataImporterViewTests method), 107
test_get_first_five_rows_with_newlines() (seed.tests.test_views.DataImporterViewTests method), 107
test_get_import_file() (seed.tests.test_views.ImportFileViewsTests method), 107
test_get_match_tree() (seed.tests.test_views.BuildingDetailViewTests method), 106
test_get_match_tree_from_child() (seed.tests.test_views.BuildingDetailViewTests method), 106
test_get_pm_filter_by_counts() (seed.tests.test_views.ImportFileViewsTests method), 107
test_get_prog_key() (seed.tests.test_decorators.TestDecorators method), 102
test_get_progress() (seed.cleansing.tests.CleansingViewTests method), 69
test_get_raw_column_names() (seed.tests.test_views.DataImporterViewTests method), 107
test_get_raw_column_names() (seed.tests.test_views.TestMCMViews method), 109
test_get_tip() (seed.tests.test_models.TestBuildingSnapshot method), 103
test_handle_id_matches_duplicate_data() (seed.tests.test_tasks.TestTasks method), 105
test_home() (seed.tests.test_views.MainViewTests method), 107
test_import_xml() (seed.green_button.tests.test_xml_importer.GreenButton method), 86
test_increment_cache() (seed.tests.test_decorators.TestDecorators method), 102
test_interval_block_data() (seed.green_button.tests.test_xml_importer.GreenButtonXMLPar method), 87
test_interval_data() (seed.green_button.tests.test_xml_importer.GreenButto method), 87
test_is_concatenated() (seed.tests.test_models.TestColumnMapping method), 104
test_is_direct() (seed.tests.test_models.TestColumnMapping method), 104
test_is_same_snapshot() (seed.tests.test_tasks.TestTasks method), 105
test_locking() (seed.tests.test_decorators.TestDecorators method), 102
test_locking_w_exception() (seed.tests.test_decorators.TestDecorators method), 102
test_map_data() (seed.tests.test_tasks.TestTasks method), 105
test_mapping_w_concat() (seed.tests.test_tasks.TestTasks method), 105
test_match_buildings() (seed.tests.test_tasks.TestTasks method), 105
test_match_duplicate_buildings() (seed.tests.test_tasks.TestTasks method), 105
test_match_no_canonical_buildings() (seed.tests.test_tasks.TestTasks method), 105
test_match_no_matches() (seed.tests.test_tasks.TestTasks method), 105
test_merge_extra_data() (seed.tests.test_models.TestBuildingSnapshot method), 104
test_merge_extra_data_does_not_override_with_blank_data() (seed.tests.test_models.TestBuildingSnapshot

method), 104
test_merge_extra_data_no_data() (seed.tests.test_models.TestBuildingSnapshot method), 104
test_meter_data() (seed.green_button.tests.test_xml_importer.GreenButtonXMLImporterTests method), 87
test_model___unicode__() (seed.audit_logs.tests.AuditLogModelTests method), 65
test_no_unmatched_buildings() (seed.tests.test_tasks.TestTasks method), 105
test_note() (seed.audit_logs.tests.AuditLogModelTests method), 65
test_note_save() (seed.audit_logs.tests.AuditLogModelTests method), 65
test_order_by_returns_all_buildings() (seed.managers.tests.test_json_manager.TestJsonManager method), 97
test_parent_tree_coparents() (seed.tests.test_views.MatchTreeTests method), 108
test_progress() (seed.tests.test_decorators.TestDecorators method), 103
test_progress() (seed.tests.test_views.TestMCMViews method), 109
test_recurse_tree() (seed.tests.test_models.TestBuildingSnapshot method), 104
test_remap_buildings() (seed.tests.test_views.TestMCMViews method), 109
test_remove_child() (seed.tests.test_models.TestBuildingSnapshot method), 104
test_repr() (seed.tests.test_models.TestCanonicalBuilding method), 104
test_require_organization_id_fail_no_key() (seed.tests.test_decorators.RequireOrganizationIDTests method), 102
test_require_organization_id_fail_not_numeric() (seed.tests.test_decorators.RequireOrganizationIDTests method), 102
test_require_organization_id_success_integer() (seed.tests.test_decorators.RequireOrganizationIDTests method), 102
test_require_organization_id_success_string() (seed.tests.test_decorators.RequireOrganizationIDTests method), 102
test_reset_mapped_w_matching_done() (seed.tests.test_views.TestMCMViews method), 109
test_reset_mapped_w_previous_matches() (seed.tests.test_views.TestMCMViews method), 109
test_save_column_mappings() (seed.tests.test_views.TestMCMViews method), 109
test_save_column_mappings_idempotent() (seed.tests.test_views.TestMCMViews method), 109
test_save_column_mappings_with_concat() (seed.tests.test_views.TestMCMViews method), 109
test_save_match_audit_log() (seed.tests.test_views.BuildingDetailViewTests method), 106
test_save_match_invalid_org() (seed.tests.test_views.BuildingDetailViewTests method), 106
test_save_match_wrong_perms_different_building_orgs() (seed.tests.test_views.BuildingDetailViewTests method), 106
test_save_match_wrong_perms_org_id() (seed.tests.test_views.BuildingDetailViewTests method), 106
test_save_raw_data() (seed.tests.test_tasks.TestTasks method), 105
test_save_raw_data_batch_iterator() (seed.tests.test_tasks.TestTasks method), 105
test_save_snapshot_match_default_to_first_building() (seed.tests.test_models.TestBuildingSnapshot method), 104
test_save_snapshot_match_default_to_second_building() (seed.tests.test_models.TestBuildingSnapshot method), 104
test_save_unmatch_audit_log() (seed.tests.test_views.BuildingDetailViewTests method), 106
test_seach_active_canonicalbuildings() (seed.tests.test_views.SearchViewTests method), 108
test_search_building_snapshots() (seed.tests.test_views.SearchBuildingSnapshotsViewTests method), 108
test_search_case_insensitive_exact_match() (seed.tests.test_views.SearchViewTests method), 108
test_search_empty_column() (seed.tests.test_views.SearchViewTests method), 108
test_search_exact_exclude_filter() (seed.tests.test_views.SearchViewTests method), 108
test_search_exact_match() (seed.tests.test_views.SearchViewTests method), 108
test_search_exclude_filter() (seed.tests.test_views.SearchViewTests method), 108

test_search_extra_data() (seed.tests.test_views.SearchViewTests method), 108
test_search_extra_data_empty_column() (seed.tests.test_views.SearchViewTests method), 108
test_search_extra_data_exact_exclude_filter() (seed.tests.test_views.SearchViewTests method), 108
test_search_extra_data_exact_match() (seed.tests.test_views.SearchViewTests method), 108
test_search_extra_data_exclude_filter() (seed.tests.test_views.SearchViewTests method), 108
test_search_extra_data_non_empty_column() (seed.tests.test_views.SearchViewTests method), 108
test_search_extra_data_non_existent_column() (seed.tests.test_views.SearchViewTests method), 109
test_search_filter_date_range_ISO8601() (seed.tests.test_views.SearchViewTests method), 109
test_search_filter_range() (seed.tests.test_views.SearchViewTests method), 109
test_search_not_empty_column() (seed.tests.test_views.SearchViewTests method), 109
test_search_sort() (seed.tests.test_views.SearchViewTests method), 109
test_separates_system_and_possible_match_types() (seed.tests.test_tasks.TestTasks method), 105
test_set_default_columns() (seed.tests.test_views.DefaultColumnsViewTests method), 107
test_signup_process() (seed.tests.test_admin_views.AdminViewsTest method), 102
test_signup_process_force_lowercase_email() (seed.tests.test_admin_views.AdminViewsTest method), 102
test_simple_login() (seed.cleansing.tests.CleansingDataTestCase method), 69
test_simple_login() (seed.landing.tests.UserLoginTest method), 93
test_sort_extra_data() (seed.tests.test_views.SearchViewTests method), 109
test_source_attributions() (seed.tests.test_models.TestBuildingSnapshot method), 104
test_tos_login() (seed.landing.tests.UserLoginTest method), 93
test_unmatch_snapshot_tree_last_match() (seed.tests.test_models.TestBuildingSnapshot method), 104
test_unmatch_snapshot_tree_prior_match() (seed.tests.test_models.TestBuildingSnapshot method), 104
test_unmatch_snapshot_tree_retains_canonical_snapshot() (seed.tests.test_models.TestBuildingSnapshot method), 104
test_update_building() (seed.tests.test_models.TestBuildingSnapshot method), 104
test_update_building_audit_log() (seed.tests.test_views.BuildingDetailViewTests method), 106
test_update_building_with_dates() (seed.tests.test_models.TestBuildingSnapshot method), 104
test_update_dataset() (seed.tests.test_views.GetDatasetsViewsTests method), 107
test_update_note() (seed.audit_logs.tests.AuditLogViewTests method), 65
test_xls_export() (seed.tests.test_exporters.TestExporters method), 103
TestBuildingSnapshot (class in seed.tests.test_models), 103
TestCanonicalBuilding (class in seed.tests.test_models), 104
TestCleaner (class in seed.tests.test_tasks), 104
TestColumnMapping (class in seed.tests.test_models), 104
TestDecorators (class in seed.tests.test_decorators), 102
TestException, 103
TestExporters (class in seed.tests.test_exporters), 103
TestJsonManager (class in seed.managers.tests.test_json_manager), 97
TestMCMViews (class in seed.tests.test_views), 109
TestTasks (class in seed.tests.test_tasks), 105
TestTasksXLS (class in seed.tests.test_tasks), 105
TestTasksXLSX (class in seed.tests.test_tasks), 106
TimeSeries (class in seed.models), 133
TimeSeries.DoesNotExist, 133
TimeSeries.MultipleObjectsReturned, 133
time_zone (seed.models.Meter attribute), 129
tip (seed.models.BuildingSnapshot attribute), 122
to_dict() (seed.audit_logs.models.AuditLog method), 64
to_dict() (seed.models.BuildingSnapshot method), 122
to_dict() (seed.models.Compliance method), 127
to_dict() (seed.models.Project method), 131
to_dict() (seed.models.ProjectBuilding method), 132
to_dict() (seed.models.StatusLabel method), 133
to_json (seed.data_importer.models.ImportRecord attribute), 80
total_correct_mappings (seed.data_importer.models.ImportRecord attribute), 80

total_file_size (seed.data_importer.models.ImportRecord attribute), 80
transfer_buildings() (in module seed.utils.projects), 144
typeof_request_values() (in module seed.common.views), 72

U

Unit (class in seed.models), 133
unit (seed.models.Column attribute), 125
Unit.DoesNotExist, 133
Unit.MultipleObjectsReturned, 133
unlocked (seed.tests.test_decorators.TestDecorators attribute), 103
unmatch_snapshot_tree() (in module seed.models), 135
update() (seed.audit_logs.models.AuditLogQuerySet method), 64
update_building() (in module seed.models), 136
update_building() (in module seed.views.main), 50
update_dataset() (in module seed.views.main), 51
update_note() (in module seed.audit_logs.views), 66
update_project() (in module seed.views.projects), 176
update_project_building() (in module seed.views.projects), 177
update_role() (in module seed.views.accounts), 31
update_status_from_dcm() (in module seed.data_importer.models), 83
update_status_from_import_file() (in module seed.data_importer.models), 83
update_status_from_import_record() (in module seed.data_importer.models), 83
update_status_from_tcm() (in module seed.data_importer.models), 83
update_tcms_from_save() (seed.data_importer.models.ImportFile method), 77
update_user() (in module seed.views.accounts), 31
UpdateBuildingLabelsSerializer (class in seed.serializers.labels), 141
UPDATING_TCMS_KEY (seed.data_importer.models.ImportFile attribute), 75
UrlBreadcrumbNode (class in seed.templatetags.breadcrumbs), 99
use_description_source (seed.models.BuildingSnapshot attribute), 122
use_for_related_fields (seed.audit_logs.models.AuditLogManager attribute), 64
use_for_related_fields (seed.data_importer.managers.NotDeletedManager attribute), 73
user (seed.audit_logs.models.AuditLog attribute), 64
user (seed.models.ColumnMapping attribute), 126
user_agreement (seed.landing.models.SEEDUser attribute), 93
user_permissions (seed.landing.models.SEEDUser attribute), 93
UserLoginTest (class in seed.landing.tests), 93
USERNAME_FIELD (seed.landing.models.SEEDUser attribute), 90
UtilsTests (class in seed.tests.tests), 110

V

valid_export_type() (seed.lib.exporter.Exporter method), 95
valid_test_cc_number() (seed.test_helpers.factory.helpers.DjangoFunctiona class method), 101
validation_rules (seed.data_importer.models.TableColumnMapping attribute), 82
ValidationOutlier (class in seed.data_importer.models), 82
ValidationOutlier.DoesNotExist, 82
ValidationOutlier.MultipleObjectsReturned, 82
validationoutlier_set (seed.data_importer.models.ValidationRule attribute), 83
ValidationRule (class in seed.data_importer.models), 82
ValidationRule.DoesNotExist, 82
ValidationRule.MultipleObjectsReturned, 82
validationrule_ptr (seed.data_importer.models.RangeValidationRule attribute), 81
validationrule_set (seed.data_importer.models.TableColumnMapping attribute), 82
values (seed.models.EnumValue attribute), 128
variants (seed.models.BuildingSnapshot attribute), 122
version() (in module seed.views.main), 51

W

WHITE_CHOICE (seed.models.StatusLabel attribute), 132
worksheet_progress_json (seed.data_importer.models.ImportRecord attribute), 80
worksheet_url (seed.data_importer.models.ImportRecord attribute), 80

Y

year_built_source (seed.models.BuildingSnapshot attribute), 122
year_ending_source (seed.models.BuildingSnapshot attribute), 122