ReMo Portal Documentation

Release 0.2.5

Mozilla

Sep 01, 2018

Contents

This is the developer's documentation for ReMo Portal.

CHAPTER 1

About ReMo Portal

Mozilla Reps web tools and portal is the next phase of tools for the Mozilla Reps program. Its purpose is to provide the required tools for the day to day operations of the hundreds of Reps signed up in the program. Diversity of the Reps and Geographic distribution are key features of the program and those tools should help unify community practices and tools of mozillians around the world.

To learn more about it and the release schedule visit ReMo Portal project page.

For comments and/or questions about ReMo Portal or this documentation please ping us on #remo-dev.

CHAPTER 2

Contents

2.1 Installation

2.1.1 VirtualEnv Installation

Getting your own development environment.

Preparing Your System

Prerequisites: You 'll need python, virtualenv, pip, git and mysql-server.

- For debian based systems:

```
$ sudo apt-get install python-dev python-pip python-virtualenv git mysql-server␣
↪libmysqlclient-dev \
  libxslt1.1 libxml2 libxml2-dev libxslt1-dev libffi-dev
```

For other Linux distributions, you can consult the documentation of your distribution.

Build the Environment

When you want to start contributing. . .

1. Fork the main ReMo repository (https://github.com/mozilla/remo) on GitHub.
2. Clone your fork to your local machine:

```
$ git clone git@github.com:YOUR_USERNAME/remo.git remo
(lots of output - be patient...)
```

3. Create your python virtual environment.:

```
$ cd remo/
$ virtualenv --no-site-packages venv
```

4. Activate your python virtual environment.:

```
$ source venv/bin/activate
```

5. Install development requirements.:

```
(venv)$ python ./bin/pipstrap.py
(venv)$ pip install --require-hashes --no-deps -r requirements/dev.txt
```

Note: When you activate your python virtual environment 'venv' (virtual environment's root directory name) will be prepended to your PS1.

Note: Since you are using a virtual environment all the python packages you will install while the environment is active, will be available only within this environment. Your system's python libraries will remain intact.

6. Configure your local ReMo installation.:

```
(venv)$ cp env-dist .env
```

Uncomment the line for *DATABASE_URL* in *.env* to point to the *localhost* hostname.

7. Setting up a MySQL database for development:

Install the MySQL server. Many Linux distributions provide an installable package. If your OS does not, you can find downloadable install packages on the MySQL site.

8. Start the mysql client program as the mysql root user:

```
$ mysql -u root -p
Enter password: ........
mysql>
```

9. Create a `remo` database:

```
mysql> create database remo character set utf8;
```

10. Sync DB.:

```
(venv)$ ./manage.py migrate --noinput
```

11. Create an admin account.

Create your own admin account:

```
(venv)$ ./manage.py createsuperuser
```

12. Update product_details package.

Package *product_details* provides information about countries. We use it in country selection lists. The information get pulled form mozilla's SVN, so we need to fetch it at least once. To update run:

```
(venv)$ ./manage.py update_product_details
```

13. Collect static files.

    Various packages provide static files. We need to collect them in the STATIC_DIR:

```
(venv)$ ./manage.py collectstatic
```

14. Load demo data (optional).

    Depending on what you are going to develop you may need to have some demo data.

    To load *demo users* run (within your virtual env):

```
(venv)$ ./manage.py loaddata demo_users
```

    To load *demo functional areas* run:

```
(venv)$ ./manage.py loaddata demo_functional_areas
```

    To load *demo mobilizing expertise* run:

```
(venv)$ ./manage.py loaddata demo_mobilising_skills
```

    To load *demo mobilizing learning interests* run:

```
(venv)$ ./manage.py loaddata demo_mobilising_interests
```

    To load *demo events* run:

```
(venv)$ ./manage.py loaddata demo_events
```

    To fetch *bugzilla bugs* run:

```
(venv)$ ./manage.py fetch_bugs
```

    Note: Fetching bugzilla bug requires a Mozilla Reps Admin account on Bugzilla. Ping *nemo-yiannis* or *tasos* on #remo-dev to give you access if your project requires it.

15. Run tests:

```
(venv)$ ./manage.py test
```

2.1.2 Docker Installation

Getting your own development environment.

Preparing Your System

1. You need to install docker in your system. The installation guide covers many operating systems but for now we only support Linux.
2. We are using an orchestration tool for docker called docker-compose that helps us automate the procedure of initiating our docker containers required for development. Installation instructions can be found in Compose's documentation. *Version required*: 1.0.1 or newer.

Build the Environment

When you want to start contributing...

1. Fork the main ReMo repository.
2. Clone your fork to your local machine:

```
$ git clone git@github.com:YOUR_USERNAME/remo.git remo
(lots of output - be patient...)
$ cd remo
```

3. Configure your local ReMo installation:

```
$ cp env-dist .env
```

4. Update the product details:

```
$ docker-compose run web python manage.py update_product_details -f
```

5. Create the database tables and run the migrations:

```
$ docker-compose run web python manage.py migrate --noinput
```

6. Create your own admin account:

```
$ docker-compose run web ./manage.py createsuperuser
```

7. Add demo users:

```
$ docker-compose run web ./manage.py loaddata demo_users
```

8. Add demo functional areas:

```
$ docker-compose run web ./manage.py loaddata demo_functional_areas
```

9. Add demo mobilizing expertise:

```
$ docker-compose run web ./manage.py loaddata demo_mobilising_skills
```

10. Add demo mobilizing learning interests:

```
$ docker-compose run web ./manage.py loaddata demo_mobilising_interests
```

11. Add demo events:

```
$ docker-compose run web ./manage.py loaddata demo_events
```

Running ReMo

1. Run ReMo:

```
$ docker-compose up
(lots of output - be patient...) or
$ docker-compose run --rm --service-ports web
(this enables the output of print() on the docker output)
```

2. Open the local site and develop!
3. Run tests:

```
$ docker-compose run web ./manage.py test
```

2.1.3 Static files setup

Installing LESS Preprocessor

1. **Install Node.js** for vagrant users
   - Install prerequisites:

     ```
     ~$ sudo apt-get install g++ libssl-dev build-essential
     ```

   - Download latest Node.js source code (eg.):

     ```
     ~$ wget http://nodejs.org/dist/v0.10.20/node-v0.10.20.tar.gz
     ```

   - Extract source code:

     ```
     ~$ tar -zxf node-v0.10.20.tar.gz
     ```

   - To build Node.js, run inside the extracted folder:

     ```
     ~$ ./configure
     ~$ make
     ~$ sudo make install
     ```

   Note: For other development environments (eg. virtualenv), follow the Node.js installation guide or use your package manager if a package is available.

2. **Install lessc using npm**:

```
~$ sudo npm install -g less
```

Settings

COMPRESS_ENABLED
: If set to `True` django serves static files compressed.

COMPRESS_OFFLINE
: If set to `True` static files will be compressed outside request/response loop. If set to `False` static files will be processed on user requests.

2.1.4 Fire up the development server

These are the steps to run locally your development server.

VirtualEnv Installation

1. **Start django devserver.**

   Within your virtual environment you can start django devserver by running:

```
(venv)$ ./manage.py runserver
```

2. **Visit our local installation of the ReMo Portal.**

   You are done! Point Firefox to .

2.1.5 Creating demo data using factories

In order to populate our development environment with data in an automated way, we have implemented model factories using Factory Boy.

Factory Boy is a fixtures replacement for Python. For more details visit the project's documentation.

Using factories

By default, model factories get instantiated using the associated model fields. On top of that we provide additional *attributes* to add extra functionality in object creation. For example:

- To create a single model object from a factory class (eg. `UserFactory`) use the `create()` method:

```
user = UserFactory.create()
```

- To create multiple model objects at once (eg. 10) use the `create_batch()` method:

```
users = UserFactory.create_batch(10)
```

- To customize your demo data you can override model attributes (eg. `username`):

```
user = UserFactory.create(username='example')
```

`remo` factory classes

Here is the list of the implemented model factories we have in `remo` and their associated PostGeneration methods that help in some complex definitions of our models.

`remo.profiles` factory classes

- **`UserFactory`**
  - **`groups`: List of strings with group names to add to user groups** (eg. `['Rep', 'Council']`)
- **`UserProfileFactory`**
  - `functional_areas`: List of `FunctionalArea` objects to add to user functional areas
  - `random_functional_areas`: `Boolean`. Populates `UserProfile` with random functional areas of random length.
  - `initial_council`: `Boolean`. `UserProfile` object has itself as mentor.
- `FunctionalAreaFactory`

`remo.events` factory classes

- **`EventFactory`**
  - `categories`: List of `FunctionalArea` objects to add to event categories.
  - `random_categories`: `Boolean`
- `AttendanceFactory`

`remo.reports` factory classes

- `NGReportFactory`
- `ActivityFactory`
- `CampaignFactory`

`remo.remozilla` factory classes

- **`BugFactory`**
  - `add_cc_users`: List of users to add to bug cc field

`remo.voting` factory classes

- `PollFactory`
- `VoteFactory`
- `RadioPollFactory`
- `RadioPollChoiceFactory`
- `RangePollFactory`
- `RangePollChoiceFactory`

Factory examples

- To `create_batch` of users (eg. 10) with random functional areas that belong to the initial council

```
from remo.profiles.tests import UserFactory

kwargs = {
    'groups': ['Reps', 'Mentor', 'Council'],
    'userprofile__random_functional_areas': True,
    'userprofile__initial_council': True
}

users = UserFactory.create_batch(10, **kwargs)
```

- To `create_batch` of past events (eg. 10) with random categories and 10 attendees

```
from remo.events.tests import EventFactory, AttendanceFactory

events = EventFactory.create_batch(10, random_categories=True)

for event in events:
    AttendanceFactory.create_batch(10, event=event)
```

Note: The above script creates new users for `event.owner`, `event.attendance.user` and new swag and budget bugs.

- To `create` a poll with 10 radio and range poll choices

```
from remo.voting.tests import *

poll = PollFactory.create()

radio_poll = RadioPollFactory.create(poll=poll)
range_poll = RangePollFactory.create(poll=poll)

radio_poll_choices = RadioPollChoiceFactory.create_batch(10, radio_poll=radio_
↪poll)
range_poll_choices = RangePollChoiceFactory.create_batch(10, range_poll=range_
↪poll)
```

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Index

C