ver1_0 Documentation

Release

Frank Grove

December 08, 2012

CONTENTS

CHAPTER

ONE

ABOUT

1.1 What is Open Assembly?

Open Assembly is an open source internet decision making framework. Main features include

- Create ideas and upload content and vote on the content
- Browse ideas based on collective approval, controversy, and more
- Users can create groups to host their ideas with variable settings for decision making and user inclusion
- Trust Network that will provide Open Assembly with the tools to deter Sock Puppets
- Coming Soon: Gamification concepts such as currency and classes.

The goal is to develop a fully functional Augmented Reality Game where users act out actions in the real world and are rewarded in the virtual world, a global forum where ideas can be peer-reviewed and tested, allowing users to achieve critical mass on the best ideas to change the world.

1.1.1 Technology

OA is built on Django-nonrel and MongoDB. We use Redis to provide caching and pub/sub. A node.js server allows OA to host dynamic chat and notification messages. We also have provided a Solr search server configured with OA to allow efficient and powerful search capabilities.

OA doesn't follow the traditional Django views style. For more info check out *Decoupling Design and Development*

1.1.2 History

Open Assembly

1.1.3 License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

- Neither the name of the Open Assembly nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

CHAPTER

TWO

INSTALLATION

2.1 Installing Development Server

First make sure you have all the requirements installed to run a development server. Some servers such as Celery and node.js depend on Redis so they must be started in the right order.

2.1.1 Open Assembly Installation

Install Git and HG if these version control libraries isn't already installed.

```
sudo apt-get install git-core mercurial
```

We recommend PIP and VirtualEnv to satisfy dependencies.

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

```
sudo pip install virtualenv
```

Now setup the structure of the development folder and create the OA virtualenv

```
mkdir OA
```

```
cd OA
```

```
git clone git://github.com/fragro/Open-Assembly.git
```

```
mkdir OA_ENV
```

```
virtualenv OA_ENV
```

```
source OA_ENV/bin/activate
```

```
cd Open-Assembly/ver1_0
```

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

2.1.2 The MongoDB server

This should be sufficient for debian servers.

```
sudo apt-get install mongodb
```

2.1.3 Redis Server

Go here and install the newest stable version or follow these instructions.

If you aren't using Redis for anything else we recommend placing the redis-2.4.17 directory in the OA folder.

```
wget http://redis.googlecode.com/files/redis-2.4.17.tar.gz

tar xzf redis-2.4.17.tar.gz

cd redis-2.4.17

make
```

2.1.4 Node.js

Install from source (check here for the latest version):

```
wget http://nodejs.org/dist/v0.8.11/node-v0.8.11.tar.gz

tar xzf node-v0.8.11.tar.gz

cd node-v0.8.11

make

sudo make install
```

Now you need to install the dependencies. Goto Open-Assembly/oanode/ and run the command

```
npm install
```

2.1.5 Solr

If you aren't using Solr for anything else we recommend placing the apache-solr-3.6.1 directory in the OA folder.

```
wget http://apache.mesi.com.ar/lucene/solr/3.6.1/apache-solr-3.6.1.tgz

tar xzf apache-solr-3.6.1.tgz
```

Now replace the schema.xml in your local version with OA's schema.xml, which contains the necessary hooks to our database. First remove the old schema. Assuming the Solr directory is in OA/

```
rm apache-solr-3.6.1/example/solr/conf/schema.xml
```

Now grab the schema from Open-Assembly/solr/conf/schema.xml

```
cp Open-Assembly/solr/conf/schema.xml apache-solr-3.6.1/example/solr/conf/
```

Now the Solr server should be ready to jive with our Django DB schema.

2.1.6 Run the Development Server

Now Open a Terminal, navigate to Open-Assembly/ver1_0/openassembly and Run the Django Server. Remember that if you installed your dependencies in a virtualenv using the command `source OA_ENV/bin/activate` you must be in that virtual environment when running these from your shell.

```
python manage.py syncdb
```

Next we will transfer the static files from the various modules into our static_dev_server folder. You need to run this command every time you add a new file to a static folder or add a new module with static files. More on static files in Django.

```
python manage.py collectstatic
```

We want to rebuild the index in Solr once you have accumulated some data in your development environment, if you want to modify the search design or code. If this is your first time starting the server you can skip this step. The production server will take care of this with a cron job.

```
python manage.py rebuild_index
```

If syncdb fails the first time, a second try should succeed.

```
python manage.py runserver
```

Start Redis Server

Open a new terminal, go to the location where you installed redis and run the following command.

```
src/redis-server
```

WARNING: You must run the Redis server before running the node.js or Celery servers

Start Celery Server

Navigate back to the Open-Assembly/ver1_0/openassembly folder where the Django server is located. OA uses django-celery to run background tasks.

```
python manage.py celeryd
```

For more debug information in Celery inlude the DEBUG flag.

```
python manage.py celeryd -l DEBUG
```

Start Solr Server

Navigate to the OA/ directory in a new terminal.

```
cd apache-solr-3.6.1/example
```

```
java -jar start.jar
```

Start Node.js Server

Navigate to the Open-Assembly/oanode directory in a new terminal.

```
node server.js
```

Amazon S3 Support

To setup your OA application for images, create a file called 'local_environment.json' in your home folder. The contents should look something like this, except substituted for your own variables from S3. The mountpoint "/home/user/media/" could be any existing directory on your filesystem. S3FS_ACCESSKEY, S3FS_SECRETKEY and S3FS_BUCKET must be setup from your S3 management console. If this is not available the django-storages will default to HashStorage.

```
{"S3FS_ACCESSKEY": "ASIODUAS27FSAS2",
        "S3FS_BUCKET": "openassembly-store",
        "S3FS_MOUNTPOINT": "/home/user/media/",
        "S3FS_SECRETKEY": "aos8ddas8foafkl2l2oka9sk9akdo2"
}
```

Usage

You should be ready to go with your dev Redis, Django, Celery, Solr, and Node.js servers up and running. Using Chrome, Firefox, Safari, or Opera and goto Admin Setup to create an administrative account with the username 'admin' and password 'password'. Now you can begin to create groups and test content to develop on.

For help in understanind the OA user interface checkout our tutorial.

2.2 Deploying Production Server

To push to production we recommend Dotcloud. It is actually much easier to push OA to production through dotcloud when compared to setting up the development server, because the server stack is built automatically. With the following instructions you can deploy an online version of OA for free.

2.2.1 Using Dotcloud

Dotcloud makes deploying Open Assembly easy. First create an account with dotcloud and install the CLI here

First clone from git if you did not do so setting up a development server. This leads to the development repository, which may be unstable from time to time. We are starting a release cycle and will soon have a stable package available.

```
git clone git://github.com/fragro/Open-Assembly.git
```

Next you just need to create a sandbox app in dotcloud. Replace ''appname" with what you want to call your deployment of OA.

```
dotcloud create appname
```

First you need to specify some important environment variables from S3 and your Email host. First the required environment variables for S3 Amazon cloud server, where image files are stored.

Amazon S3 Support

To setup your OA application for images, create a file called 'local_environment.json' in your home folder. The contents should look something like this, except substituted for your own variables from S3. The mountpoint

"/home/user/media/" could be any existing directory on your filesystem. S3FS_ACCESSKEY, S3FS_SECRETKEY and S3FS_BUCKET must be setup from your S3 management console.

```
dotcloud env set \
    'S3FS_ACCESSKEY=MYSECRETACCESSKEY' \
    'S3FS_BUCKET=openassembly-store' \
    'S3FS_SECRETKEY=MYSECRETS3FSKEY'
```

Note if you do not have S3 or want to use a different method of file/image storage, please see the settings.py file in ver1_0/openassembly and change the value of DEFAULT_FILE_STORAGE to specify the storages backed you want. For more information on the different backends, see django storages documentation .

OA also requires Setting of EMAIL_HOST_USER, EMAIL_HOST and EMAIL_PASSWORD within the dotcloud environment variables. This allows you to easily include your own email host.

You can modify the local version before you push to dotcloud.

```
dotcloud env set \
        'EMAIL_PASSWORD=mysecretpassword' \
        'EMAIL_HOST_USER=myemail@gmail.com' \
        'EMAIL_HOST=smtp.gmail.com' \

DEFAULT_FROM_EMAIL = env['EMAIL_HOST_USER']
EMAIL_USE_TLS = True
EMAIL_HOST = env['EMAIL_HOST']
EMAIL_HOST_USER = env['EMAIL_HOST_USER']
EMAIL_HOST_PASSWORD = env['EMAIL_PASSWORD']
EMAIL_PORT = 587
```

You also must set the EMAIL_PASSWORD environment variable in Dotcloud environment variables.

```
dotcloud var set appname EMAIL_PASSWORD=mysecretpassword
```

You'll also need to setup reCaptcha to keep those pesky spam bots off your back. Go to the reCaptcha website to get a Public and Private key from Google. Set those environment variables the same as you would the S3 settings.

```
dotcloud env set \
    'RECAPTCHA_PUBLIC_KEY=6LehG9oSAAAAAD256YWh5x_STpHRlEIxd3TKR3is' \
    'RECAPTCHA_PRIVATE_KEY=6LehG9oSAAAAAKU-4rViXJrsGBgj7gImL0MMu3ae'
```

Then navigate to the Open-Assembly/ folder and connect/push to dotcloud.

```
dotcloud connect appname
dotcloud push
```

That's it! You deployed your own verstion of OA live and at the end of output there should be a url. If the push fails for some reason try again. If the push times out, go to dashboard.dotcloud.com and check on the status of your OA install live. If you want to make your OA deployment scalable and reliable you will need to access the billing details from Dotcloud and your app to Live, but sandbox apps will work for small groups that don't mind using the dotcloud URL.

2.2.2 Other Hosts

Open Assembly is configured to use dotcloud but you can use your own host fairly easily with the pip requirements file, you'll need to change the settings.py file in the project to reflect your own Redis/MongoDB/Node/Celery Servers. If anyone has success deploying to a different host we would appreciate feedback on your experience.

CHAPTER

THREE

FOR DESIGNERS

3.1 Decoupling Design and Development

Open Assembly deviates from the traditional approach found in Django concerning Views. Instead of developing a views.py function for different types of content, we have a single view that loads template files, where template tags take the place of the logic traditionally found in views.py in a Django project.

3.1.1 So what does a template tag function look like?

Template tags are similar to Django template tags if you are familiar with those. They can be easily added to html files. These templates are rendered before the page loads, so they can also be used to procedurally generate javascript code. Combining template tags and javascript can be quite powerful.

This function calls `pp_consensus_get` from the `consensustags` module and displays the interest attribute of the `pirate_consensus.models.Consensus` object. There are a few things happening here within the `pp_consensus_get` function.

- Loads the parameter `object` into the local context of the Python/Djano function
- Performs the logic of the function
- Loads the results of the function into the `pp_consensus` context, which the HTML designer can access through the template

You can easily filter the data from the template tag's context using other templatetags found in Django libraries or elsewhere.

Example

```
<body>
        {% pp_consensus_get object=object.pk %}
                <div>
                    {% if pp_consensus.consensus.interest > 1000 %}

                        <a href="{{object.get_absolute_url}}">{{ pp_consensus.consensus.interest|floa

                    {% else %}

                        <a href="{{object.get_absolute_url}}">{{ pp_consensus.consensus.interest|floa

                    {% endif %}
                </div>
```

```
        {% endpp_consensus_get %}
</body>
```

Context Variables

As you can see the function relies on the `object` variable, which is loaded by the `oa_cache module`. For designers all you really need to know is that the following is available to you as Django template objects in any template you create. These are commonly used as parameters to template tag functions, or can be used to populate the template with contextual data you are presenting to the user.

object `django.db.models.Model` object

user `django.contrib.auth.User` object of logged in user

start start integer for pagination

end end integer for pagination

dimension dimension string for sorting or filtering, usually reserved for lists

Here's an example of how one might use these objects in a template.

```
<h2><a href="{{object.get_absolute_url}}">{{object.summary}}</a></h2>

{% if user == object %}

        Welcome home {{user.username}}.

        {% pp_get_messages start=start end=end user=user %}

                {% for message in pp_messages.all reversed %}

                        <div>

                                {{note.description}}

                        </div>

                {% endfor %}

        {% endpp_get_messages %}

{% endif %}
```

3.1.2 pp_url Links

Django allows you to drop links into your templates fairly easily. You need to use the *pp_url* template tag.

This block tag will produce a url that will link to the designated view or pattern name, and then will optionally populate the request passed to that view with either a specific ORM object, or a numerical range (start...end), as long as the pirate_core.url_middleware.UrlMiddleware is included in the projects' MIDDLEWARE_CLASSES. Any kwargs included in addition to "view", "object", "start" and "end" will be passed to redirect in order to produce the url for the designated view.

The default value for "view" is "pp-page", which expects that the kwarg "template" be included, passing in the name of the template being linked to.

For example:

```
{% pp_url object=object template="filename.html" %}

{% pp_url template="filename.html" start=0 end=30 dimension="n" %}

{% pp_url template="filename.html" %}
```

Try the following from the Django shell from `manage.py` in the openassembly directory.

```
python manage.py shell

>>> from django import template
>>> from pirate_topics.models import Topic
>>> topic = Topic(summary="A test topic.", shortname="test-topic", description="test", group_members=
>>> topic.save()
>>> load = "{% load pp_url %}"

>>> ts = "{% pp_url template='example.html' object=topic %}"
>>> template.Template(load + ts).render(template.Context({'topic':topic}))
u'/p/example/k-test-topic'

>>> ts = "{% pp_url template='example.html' object=topic start=0 end=30 %}"
>>> template.Template(load + ts).render(template.Context({'topic':topic}))
u'/p/example/k-test-topic/s-0/e-30'

>>> ts = "{% pp_url template='example.html' start=0 end=30 dimension='new' %}"
>>> template.Template(load + ts).render(template.Context({'topic':topic}))
u'/p/example/s-0/e-30/d-new'

>>> topic.delete()
```

3.2 Django Template Tags

3.2.1 cachetags

3.2.2 dashboardtags

3.2.3 locationtags

3.2.4 haystacktags

3.2.5 verificationtags

3.2.6 badgetags

3.2.7 commenttags

3.2.8 consensustags

3.2.9 pp_url

3.2.10 pp_combo_form

3.2.11 show_stars

3.2.12 tag_helpers

3.2.13 argumenttags

3.2.14 flagtags

3.2.15 blobtags

3.2.16 logintags

3.2.17 messagetags

3.2.18 notificationtags

3.2.19 cani

3.2.20 groups

3.2.21 profiletags

3.2.22 reputationtags

3.2.23 feedtags

3.2.24 subscriptiontags

3.2.25 voting_tags

3.2.26 sourcetags

CHAPTER

FOUR

DOCUMENTATION

4.1 openassembly Package

4.1.1 oa_cache Package

management Module

models Module

tasks Module

tests Module

views Module

Subpackages

cachetags

4.1.2 oa_dashboard Package

management Module

models Module

tasks Module

tests Module

views Module

Subpackages

dashboardtags

4.1.3 oa_location Package

models Module

search_indexes Module

views Module

Subpackages

base_fields = {'form_id': <django.forms.fields.CharField object at 0x3795310>, 'flag': <django.forms.fields.ChoiceFi

media

models Module

tests Module

views Module

Subpackages

flagtags

4.1.13 pirate_forum Package

forms Module

management Module

models Module

search_indexes Module

tasks Module

tests Module

views Module

Subpackages

blobtags

4.1.14 pirate_login Package

backends Module

models Module

tests Module

views Module

Subpackages

logintags

4.1.15 pirate_messages Package

management Module

models Module

tasks Module

tests Module

views Module

Subpackages

CHAPTER

FIVE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

O