AshTag Documentation

Release 1.0

Adam Charnock & Steve Pike

May 27, 2015

Contents

CHAPTER 1

Contributing to AshTag

All are welcome to contribute to AshTag!

If you're a coder and you'd like to help out, then you're already reading the docs which is an excellent start. Continue reading to get going.

If you're not a coder, you can still help! Send your ideas to us by adding an 'issue' here: https://github.com/adapt/ashtag/issues

1.1 Code style

While we're very grateful for contributions, we also like clean code! If you'd like to contribute any python code, please make sure you have read PEP-8 and PEP-257 first.

Your editor should be using soft tabs, set to 4 spaces, and should remove any trailing whitespace when you save.

If you are using something like vim or emacs, you can install flake8 to ensure you stay on the straight and narrow ;)

If you are using any other editor, why not add to these docs so we all know how to stay on the straight and narrow!

CHAPTER 2

Project Structure

2.1 Layout

2.2 Settings & Environment

CHAPTER 3

Running Locally

3.1 Using Vagrant / VirtualBox (simple)

AshTag is packaged as a Vagrant project, which allows you to get up and running quickly using a virtual machine. You will need to download and install the following:

- VirtualBox
- Vagrant

Clone the AshTag repository into a directory of your choosing:

```
# Note use "git://github.com/adapt/ashtag.git" for read-only access
git clone git@github.com:adapt/ashtag.git
cd ashtag
```

Now setup the VM using vagrant:

```
# Start the VM. As the VM doesn't exist, this will download and
# configure it
vagrant up
```

Once complete, you can SSH into the sever and run the django development server:

```
# SSH into the VM
vagrant ssh
# You are now on the virtual machine, so you can run the django dev server
runserver
```

You should now be able to open http://127.0.0.1:8080/ in your web browser.

The files in your cloned repository will be kept in sync with those on the VM, so you can go ahead and start editing files locally.

Note: `runserver` is simply and alias for the somewhat longer command:

```
django-admin.py runserver 0.0.0.0:8080
```

3.1.1 Starting and Stopping the Virtual Machine

When you are finished working on the project, you can shut down the virtual machine to save on system resources:

```
# Shut down the VM
vagrant halt
```

When you come back to the project, you can simply start it using:

```
# Start the VM again (as the VM exists, this will be a relatively fast)
vagrant up
```

3.1.2 Reconfiguring the Virtual Machine

If the Vagrant config or build scripts change it will probably be necessary to rebuild the VM. To do this, simply destroy and recreate it:

```
vagrant destroy
vagrant up
```

3.2 Using virtualenvwrapper on Mac OS X (more involved)

This section of the guide assumes you already have `virtualenvwrapper` setup. Checkout the virtualenvwrapper documentation for details of how to do this.

First, you will need to checkout the code:

```
# Note use "git://github.com/adapt/ashtag.git" for read-only access
git clone git@github.com:adapt/ashtag.git
cd ashtag
```

3.2.1 Setting up the Environment

Install the dependencies for the local development environment (note you have to install django first, because django-registration has a bug):

```
pip install Django==1.5.1
pip install -r requirements/localdev.txt
```

Now edit your environment's `postactivate` hook to include the following:

```
# Add this to your postactive hook ($VIRTUAL_ENV/bin/postactivate)
export PYTHONPATH="$PROJECT_HOME/ashtag/src:$PROJECT_HOME/ashtag/lib"
export DJANGO_SETTINGS_MODULE=ashtag.settings.localdev
export DJANGO_SECRET_KEY="RANDOM STRING HERE"
```

Warning: Ensure you replace *RANDOM STRING HERE* with a random string, especially for deployment.

And now souce the file to load the new settings into your environment:

```
source $VIRTUAL_ENV/bin/postactivate
```

3.3 Setup Postgres / PostGIS

Outside your virtual environment:

```
sudo pip install numpy
brew update
brew install gdal
```

For those on Mac OS X, we recommend using **'Postgres.app'_**. In order to enable the spatial element, simply create a database (let's call it ashtag) and then enable the spatial element:

```
createdb -h localhost ashtag
psql -h localhost ashtag

ashtag=# CREATE EXTENSION postgis;
```

In theory, that's it...

Note that we do `-h localhost` because the Postgres.app is not using the normal sockets approach, rather it binds to 0.0.0.0 (or 127.0.0.1 by default I think) on port 5432. If you're using linux then probably you don't need that bit.

3.3.1 Start it up

Now sync the db and start the development server:

```
django-admin.py syncdb
django-admin.py migrate
django-admin.py runserver
```

CHAPTER 4

Deployment

To deploy to dotCloud, follow the instructions here: http://docs.dotcloud.com/firststeps/install/ (though, as we are using virtualenv, you won't need to install pip or use *sudo* as you'll be doing this inside the virtualenv.) You'll need to get the credentials for dotcloud from the maintainers of ashtag.

Once you have done that, add the following to your *$VIRTUAL_ENV/bin/postactivate* hook:

```
dotcloud connect --git --branch master ashtag
```

Now, run the following:

```
dotcloud push
```

That's it!

CHAPTER 5

Test status