Tabs on Tallahassee Documentation

Release 1.0

Erin Braswell and James Turk

January 16, 2016

Contents

Contents:

CHAPTER 1

Developing

Tabs on Tallahassee (ToT) is a Django application powered by the Open Civic Data specification.

Development requires:

- Python 3.4 or 3.5 (w/ virtualenv/pip)
- PostgreSQL 9.4 w/ PostGIS
- git
- **a relatively recent version of poppler utils for `pdftotext`**
  - on OSX: `brew install poppler`
  - on Ubuntu: `apt-get install poppler-utils`

1.1 Getting Started

1. Create a new virtualenv:

```
$ pyvenv tot -p `which python3`
```

2. Within this virtualenv install the requirements:

```
(tot)$ pip install -r requirements.txt
```

3. Create a Postgres DB (default name is "opencivicdata" w/ a user named "pupa" w/ password "pupa", this is fine for development but be sure to change these settings for deployment.

4. Initialize the database:

```
(tot)$ pupa dbinit us                # loads OCD / pupa setup for United States jurisdictions
(tot)$ ./manage.py migrate           # run remainder of database migrations
(tot)$ ./manage.py loadshapefiles    # load shapefiles from shapefiles/ directory
(tot)$ ./manage.py loadmappings us   # create mappings from FL districts to shapefiles
```

5. At this point it should be possible to run the scraper:

```
(tot)$ pupa update fl people bills session=2016
```

You can also run other sessions too:

```
(tot)$ pupa update fl bills session=2015A
```

6. At this point you're good to go, you can run the Django dev server in the typical way:

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

1.2 A Functional Database

Setting up a full database w/ all of the data is time-consuming and in theory only needs to be done once.

A "clean" database has been created w/ the following steps:

```
(tot)$ pupa dbinit us
(tot)$ ./manage.py migrate
(tot)$ ./manage.py loadshapefiles
(tot)$ ./manage.py loadmappings us
(tot)$ ./manage.py loaddata fixtures/*.json
(tot)$ pupa update fl people
(tot)$ pupa update fl bills session=2015
(tot)$ pupa update fl bills session=2015A
(tot)$ pupa update fl bills session=2015B
(tot)$ pupa update fl bills session=2016

# TODO: 204, 2015C, 2016
```

1.3 Using Docker

It's also possible to use docker-machine to run a development server.

To run a dev environment w/ Docker:

```
$ docker-machine create --driver virtualbox --virtualbox-memory "2048" tot
$ eval $(docker-machine env tot)
$ docker-compose up
$ open http://$(docker-machine ip tot):8000
```

1.4 Directory Layout

The ToT source code consists of:

ansible/ Ansible deployment playbook

docs/ The sphinx source for these docs.

fixtures/ Django fixtures used to initialize an empty database.

shapefiles/ Florida district shapefiles from Census.gov, current as of 2015.

static/ Static assets (css, javascript)

templates/ All Django templates for the website.

fl/ A Pupa 0.5 compatible scraper for Florida's legislature.

api/ Django application powering the API.

bills/ Django application powering the bill list/detail views.

glossary/ Django application powering the glossary functionality.

legislators/ Django application powering the legislators list/detail views.

preferences/ Django application for ToT user preferences.

tot/ Project settings/wsgi app/etc.

1.5 Additional Notes

This project was developed against:

- opencivicdata-django 0.8.2
- opencivicdata-divisions-2015.4.21
- pupa 0.5.2
- represent-boundaries 0.7.4
- Django 1.9
- djangorestframework 3.3.0
- django-cors-headers 1.1.0
- whitenoise 2.0.4
- lxml 3.4.4
- Markdown 2.6.2
- django-registration @ f1a8c0
- rest_framework_json_api @ d217ba

Many of these libraries were under active development at the time of writing and significant changes may have occurred. Before upgrading any libraries be very careful to ensure that they don't introduce breaking changes.

CHAPTER 2

Deployment

A complete ansible deployment plan is provided in the `ansible/` directory.

It assumes a clean Ubuntu host (tested w/ 15.04) that'll be used exclusively for hosting ToT. That said it makes a best effort to be self-contained and not do anything unnecessary system-wide, but is **untested** in a shared hosting environment.

The basics of deployment are (see `tot/tasks/main.yml` for detail):

- installs necessary system-wide packages
- creates a new user `tot` w/ a homedir of `/home/tot`
- checks out latest source to `/home/tot/src/tot`
- builds a virtualenv in `/home/tot/virt/`
- installs tot entries for uwsgi and nginx
- writes a `/home/tot/run-scrapers.sh` script and installs a cron job that calls it at regular intervals

This means a homedir that looks something like:

```
~tot
   |
   +-- data          - directory containing uwsig sock files
   +-- logs          - uwsgi, nginx, and scraper logs
   +-- src/tot       - checkout of project
   +-- virt          - virtualenv
   +-- _data         - scraper data directory from last run
   +-- _cache        - scraper cache directory
```

2.1 EC2 Deployment

2.1.1 Configure SES

SES should be configured to send emails to registered users.

- Within the AWS Console select SES -> Identity Management -> Domains
- Add desired domain, console will give instructions on adding DNS entries
- After adding DNS entries domain should show up as verified, be sure to enable DKIM.

Despite verification at this point you can only send emails to verified email addresses.

While this will work for testing, it'll be necessary to use the console to make a support request to Amazon to remove this limitation.

2.1.2 Create RDS instance

tested with Postgres 9.4.4

2.1.3 Create EC2 instance

tested with ami-a85629c2

2.1.4 Set Security Groups

Suggested configuration is two groups:

- tot-web - for EC2 instance(s), open to world on port 443 for HTTPS and 22 for selected IPs
- tot-db - for DB instance(s), only open to tot-web

2.1.5 Create Ansible Config

Create an ec2/ directory with the following contents:

ec2/hosts:

```
tot ansible_ssh_host=<instance ip> ansible_ssh_user=ubuntu ansible_ssh_private_key_file=ec2/tot.pem
```

ec2/hosts/tot.yml:

```
---
django_environment:
    SECRET_KEY: <random string>
    DEBUG: false
    DATABASE_URL: postgis://<rds username>:<rds password>@<rds host>:5432/<rds db name>
    ADMINS: Name email@example.com, Name 2 email2@example.com
    EMAIL_HOST: email-smtp.us-east-1.amazonaws.com
    EMAIL_HOST_USER: <smtp-username>
    EMAIL_HOST_PASSWORD: <smtp-password>
    DEFAULT_FROM_EMAIL: noreply@example.com
server_name: ""
ssl_cert: "..."
ssl_key: "..."
```

2.1.6 Run Ansible Playbook

$ ansible-playbook tot.yml -i ec2/hosts

CHAPTER 3

Scraping

Scraping is an inheriently fragile process, and because it depends on an outside resource (in this case the Florida State Legislature's websites) it is the one most likely to break at some point.

This document assumes you can run the scraper locally, the process in developing will walk you through getting a basic environment set up to run the scraper locally.

3.1 pupa basics

`pupa update` is the command used to run the scrapers

It takes a module name and a list of scrapers to run, each of which can have it's own keyword arguments.

For our purposes it will always be invoked as `pupa update fl people` or `pupa update fl bills session=...`

Run `pupa update --help` for additional details.

3.2 Scraper Structure

The scrapers in `bills.py` and `people.py` are composed of Page objects that return either a single piece of information or a list of similar information using XPath.

The pattern is something the author refers to as 'Spatula' and there's a decent summary in `fl/base.py`.

Generally this makes it possible to swap out functionality when a page changes without affecting other parts of the scraper.

One other note about the general philosophy applied to the scrapers is that they use the tried & true "break early & break often" method. The more "intelligent" a scraper tries to be against page changes, the more bad data sneaks into the system. Given the relative importance of clean data for the purposes of trustworthiness, the scraper will more than likely bail if the page has changed substantially. Often these are small one-line fixes, but this method prevents bad data from being exposed publicly.

3.3 When Things Change

When things inevitably do change on the sites being scraped, the process looks something like this:

- isolate the pages that have changed (hopefully just one or two types of pages) and modify the appropriate page subclasses.
- locally, run the modified scraper and watch the pupa output to see if there are unexpected numbers of modified items. (Ideally you can test against stable data and ensure 0 modified items.)
- use the admin to verify that any changes are desired/acceptable
- merge the scraper changes into production & redeploy to the server

3.4 New Sessions

Updating the scraper for new sessions is a matter of looking at __init__.py and adding a new dictionary to `legislative_sessions` in the format of the others.

It is also necessary to modify the `session_number` dict in `HousePage.do_request` (found in `bills.py`) Look at the source of http://www.myfloridahouse.gov/Sections/Bills/bills.aspx to determine the appropriate value.

CHAPTER 4

Using the Admin

The Admin Site allows easier access to the database and also offers several useful utilities for ensuring data quality.

The admin allows superusers access to all objects in the system, but extreme caution should be used when modifying objects.

4.1 Administering Users

Clicking the ‘Users’ link on the admin page will allow viewing/modifying users.

- To grant admin access: check the ‘Staff status’ and ‘Superuser status’ boxes on a user’s page.
- To disable a user: uncheck the ‘Active’ box on a user’s page.
- To change a user’s password: click the link under the ‘Password’ information on a user’s page.

4.2 Browsing Legislative Data

All of the legislative data collected by the scraper is browsable under the ‘Open Civic Data’ heading. Most of these views are nearly 100% read-only as data should only be modified with extreme caution as the scraper will overwrite most changes. These admin views are instead designed to be useful for reviewing data.

Bills bills and related information (actions, sponsorships, etc.)

Divisions boundaries

Jurisdictions top-level object representing Florida’s state government

Organizations political parties and other relevant statewide organizations

People legislators and related information (contact details, etc.)

Posts definition of seats in the legislature

Vote Events votes taken within the legislature

4.3 Unresolved Legislators Tool

Legislators are quite commonly referred to in different ways across the official data sources. A common example is having legislators referred to by last name only when they’re bill sponsors or voting in committee.

While it is possible to automatically resolve legislators in many cases it is a common source of hard-to-diagnose data quality issues. To avoid this, the system in place here is to not make assumptions and favor a quick manual reconciliation step.

The tool takes the form of a list of entities referred to in the system (as sponsors or voters) that are currently unresolved to legislators. The list is prioritized by the number of times the name in question occurs.

Note: Due to the nature of legislative data it is not possible to resolve 100% of the entities. Occasionally non-legislator entities are listed as sponsors, and sometimes that upstream site (i.e. the FL legislature) does not provide enough information to disambiguate legislators. In this case it is unfortunately the case that the legislator cannot be properly resolved.

4.4 Merge Tool

From time to time a legislature will change the way they refer to a legislator. For example, adding a nickname or perhaps the legislator's legal name has changed.

For example let's say that a legislator named William Smith is now known as Bill Smith.

In this case the scraper will assume that Bill Smith is a new legislator and create a duplicate entitiy in the system. In this case, the solution is to navigate to the merge tool and select the two legislators.

After selecting the legislators a long list of the data that'll be modified will be shown. This should be reviewed carefully as the change is irreversible.

Once the data to be merged has been reviewed, the merge can be confirmed and the entities will be updated as shown in the merge plan.

4.5 Pupa Runlog

View a list of scraper runs.

Runs are either marked as successes or failures. If a run is a success it will have details such as how many objects it created/updated and if a run fails it will have an exception traceback showing what failed. In the case of repeated failures it is typically necessary to make modifications to the scraper as it is likely the site structure has changed.

4.6 Glossary

Edit terms that are highlighed on the site & listed on the glossary page.

4.7 Preferences

View & modify user preferences such as tracked information & location.