psephology Documentation

Release

Rich Wareham

Aug 24, 2017

Contents

CHAPTER 1

Getting started

This section will get you up and running with Psephology. It is assumed that you have a working docker installed. It's preferable to have docker-machine installed as well because it's awesome.

Docker hub

Psephology can be got up and running without needing to clone the source repository since there is an image on Docker hub which is built automatically on each commit to master.

The image is configured to use the SQLite database internally and so should be considered an "ephemeral" install in that the database state is not persisted. In production, a further Dockerfile would be used which builds on this image and adds configuration for a persistent database.

Since docker will automatically pull images not available locally, we can fetch and run the Psephology server in one line:

```
$ docker run -it --rm --name psephology-server \
    -p 5000:5000 rjw57/psephology
```

Although the container is now running, trying to visit the site will result in an error. This is because we've not yet migrated the database to the latest version. Database migration is the process of updating the schema to match the latest version. It's good practice to have the migration be under the control of a separate command so that potentially database changing operations are explicit.

The database migration takes one command. In a separate terminal,

```
$ docker exec psephology-server flask db upgrade
```

Now you should be able to navigate to http://localhost:5000/ and see the app in all of its glory. If you're using `docker-machine`, you'll need to use the appropriate IP for the virtual machine:

```
$ xdg-open http://$(docker-machine ip):5000 # Linux-y machines
$ open http://$(docker-machine ip):5000     # Mac OS X
```

Installation from source

Firstly, clone the Psephology application from GitHub:

```
$ git clone https://github.com/rjw57/psephology
$ cd psephology
```

One can now build the Docker container directly from the repo:

```
$ docker build -t rjw57/psephology .
```

As part of the container build, the test suite is run to ensure that the current version is runnable inside the container environment. Once the container is built, you can run the server using `docker run` as outlined above.

Alternatively, you can opt for a local install via `pip`. It is good practice to setup a virtualenv first:

```
$ virtualenv -p $(which python3) venv
$ . ./venv/bin/activate
$ pip install -e .
```

Once installed, the test suite can be run via `setup.py`:

```
$ python setup.py test
```

Code coverage can be calculated using the `coverage` utility:

```
$ pip install coverage
$ coverage run setup.py test && coverage report
```

Migrate the initial database and start the server:

```
$ export PSEPHOLOGY_CONFIG=$PWD/config.py
$ export FLASK_APP=psephology.autoapp
$ export FLASK_DEBUG=1
$ flask db upgrade
$ flask run
```

When installed from source, the server is configured in "debug" mode with the Flask debug toolbar inserted into the UI. You should be able to navigate to http://localhost:5000/ and use the webapp.

Getting data in

We can look around the site but at the moment there isn't much to see since there's no data in the database. The Psephology repo comes with the results of the General Election 2017 in the correct results format. You can use the excellent httpie tool to post the results:

```
$ pip install httpie    # if you don't have it
$ cat test-data/ge2017_results.txt | \
    http POST http://$(docker-machine ip):5000/api/import
{
  "diagnostics": [],
  "line_count": 650
}
```

Note the `diagnostics` field which is returned. If we add some bad results lines then human-readable errors are returned:

```
$ cat test-data/noisy_results.txt | \
    http POST http://$(docker-machine ip):5000/api/import
{
  "diagnostics": [
    {
      "line": "Strangford, 507, X, 607, G",
      "line_number": 1,
      "message": "Party code \"X\" is unknown"
    },
    {
      "line": "",
      "line_number": 5,
      "message": "Constituency name cannot be empty"
    },
    {
      "line": "Oxford East, 11834, C, 35118, L, 4904, LD, 1785, G, 10, LD",
      "line_number": 6,
      "message": "Multiple results for one party"
    }
  ],
  "line_count": 7
}
```

We can use the API to get a table listing how many seats each party currently has:

```
$ http http://$(docker-machine ip):5000/api/party_totals
{
  "party_totals": {
    "C": {
      "constituency_count": 321,
      "name": "Conservative Party"
    },
    "G": {
      "constituency_count": 8,
      "name": "Green Party"
    },
    "L": {
      "constituency_count": 263,
      "name": "Labour Party"
    },
    "LD": {
      "constituency_count": 13,
      "name": "Liberal Democrats"
    },
    "SNP": {
      "constituency_count": 35,
      "name": "SNP"
    }
  }
}
```

Similarly we can retrieve the winners of each constituency via the API. Results are returned for each constituency even when there is currently no winner. (For example if a blank results line has been given.)

```
$ http http://$(docker-machine ip):5000/api/constituencies
{
  "constituencies": [
    {
      "maximum_votes": 22662,
      "name": "Aberavon",
      "party": {
        "id": "L",
        "name": "Labour Party"
      },
      "share_percentage": 74.28459042187039,
      "total_votes": 30507
    },

    // ....

    {
      "maximum_votes": null,
      "name": "Belfast West",
      "party": null,
      "share_percentage": null,
      "total_votes": null
    },

    // ....

    {
      "maximum_votes": 34594,
      "name": "York Central",
      "party": {
        "id": "L",
        "name": "Labour Party"
      },
      "share_percentage": 65.16350210970464,
      "total_votes": 53088
    },
    {
      "maximum_votes": 29356,
      "name": "York Outer",
      "party": {
        "id": "C",
        "name": "Conservative Party"
      },
      "share_percentage": 51.118811708778104,
      "total_votes": 57427
    }
  ]
}
```

It is also possible to update a constituency result via the API. For example, let's allow the Liberal Democrats to win Cambridge:

```
$ echo Cambridge, 10, C, 10, L, 1000, LD |
    http POST http://$(docker-machine ip):5000/api/import
{
  "diagnostics": [],
  "line_count": 1
}
```

Looking at the party totals, we see that Labour have lost one seat and the Liberal Democrats have gained one:

```
$ http http://$(docker-machine ip):5000/api/party_totals
{
  "party_totals": {
    "C": {
      "constituency_count": 321,
      "name": "Conservative Party"
    },
    "G": {
      "constituency_count": 8,
      "name": "Green Party"
    },
    "L": {
      "constituency_count": 262,
      "name": "Labour Party"
    },
    "LD": {
      "constituency_count": 14,
      "name": "Liberal Democrats"
    },
    "SNP": {
      "constituency_count": 35,
      "name": "SNP"
    }
  }
}
```

Building the documentation

The documentation is built with the `sphinx` tool and has additional requirements. You can install the requirements and build the documentation via:

```
$ pip install -r doc/requirements.txt
$ make -C doc singlehtml
$ xdg-open doc/_build/singlehtml/index.html     # Linux-y
$ open doc/_build/singlehtml/index.html         # OS X
```

CHAPTER 2

Web UI

The web UI is available at http://localhost:5000/ (or at the appropriate IP if using `docker-machine`). The following pages are available:

- A summary giving total number of seats for each party
- A list of winners for each constituency
- An event log showing any errors/warnings from importing results files
- A page which lets the user upload a new results file
- A page which provides the current results as a plain text file in the result line format

CHAPTER 3

Command line

The Psephology application has a command-line interface which can be run via the `flask` tool. Perhaps the most useful command is `flask run` which will launch a server hosting the application.

Database migration

Psephology uses Flask-Migrate and alembic to manage the database migrations. On first run or when upgrading the software, remember to run `flask db upgrade` to migrate the database to the newest version.

Importing results

Results may be imported from the command line via `flask psephology importresults`. See `flask psephology importresults --help` for more information.

CHAPTER 4

Programmers' reference

Application object creation

The *app* module provides support for creating Flask Application objects.

psephology.app.**create_app**(*config_filename=None*, *config_object=None*)

Create a new application object. The database and CLI are automatically wired up. If `config_filename` or `config_object` are not `None` they are passed to `app.config.from_pyfile()` and `app.config.from_object()` respectively.

Returns the newly created application object.

The *autoapp* module may be imported to automatically create an application from the configuration specified in the `PSEPHOLOGY_CONFIG` environment variable. This is useful, for example, when using the gunicorn web server where one can launch the application via:

```
$ gunicorn psephology.autoapp:app
```

Importing/exporting data

The *io* module provides functions which can be used to parse external data formats used by Psephology.

psephology.io.**parse_result_line**(*line*)

Take a line consisting of a constituency name and vote count, party id pairs all separated by commas and return the constituency name and a list of results as a pair. The results list consists of vote-count party name pairs.

To handle constituencies whose names include a comma, the parse considers count, party pairs *from the right* and stops when it reaches a vote count which is not an integer.

Data model

The *model* module defines the basic data model used by Psephology along with some utility functions which can be used to mutate it. The *query* module contains some potted queries for this data model which provide useful summaries.

None of the functions in *model* will run `session.commit()`. If you mutate the database inside a UI/API implementation, you'll need to remember to commit the result. This is to guard against partial updates to the DB is a UI/API method fails.

class psephology.model.**Party**(***kwargs*)
A political party. Each party is primarily keyed by its abbreviation. In addition, the name of a political party should be unique.

id
String primary key. This is the party "code" such as "C" or "LD".

name
Human-readable "long" name for the party.

votings
Sequence of *Voting* instances associated with this party.

class psephology.model.**Constituency**(***kwargs*)
A constituency. Essentially this is a mapping between a numeric id and a human-friendly name.

id
Integer primary key.

name
Human-readable name. Must be unique.

votings
Sequence of *Voting* instances associated with this constituency.

class psephology.model.**Voting**(***kwargs*)
A record of a number of votes cast for a particular party within a constituency.

id
Integer primary key.

count
Number of votes cast.

constituency_id
Integer primary key id of associated constituency.

constituency
Constituency instance for associated constituency.

party_id
String primary key id of associated party.

party
Party instance for associated party.

class psephology.model.**LogEntry**(***kwargs*)
A record of some log-worthy text.

id
Integer primary key.

created_at
Date and time at which this entry was created in UTC. When creating an instance, this defaults to the current date and time.

message
Textual content of log.

psephology.model.**log**(*message*)
Convenience function to log a message to the database.

psephology.model.**add_constituency_result_line**(*line*, *valid_codes=None*, *session=None*)
Add in a result from a constituency. Any previous result is removed. If there is an error, ValueError is raised with an informative message.

Session is the database session to use. If None, the global db.session is used.

If valid_codes is non-None, it is a set containing the party codes which are allowed in this database. If None, this set is queried from the database.

The session is not commit()-ed.

class psephology.model.**Diagnostic**(*line*, *message*, *line_number*)
A diagnostic from parsing a file. Records the original line, a human-readable message and a 1-based line number.

psephology.model.**import_results**(*results_file*, *valid_codes=None*, *session=None*)
Take a iterable which yields result lines and add them to the database. If session is None, the global db.session is used.

If valid_codes is non-None, it is a set containing the party codes which are allowed in this database. If None, this set is queried from the database.

Note: This can take a relatively long time when adding several hundred results. Should this become a bottleneck, there are some optimisation opportunities.

The *query* module provides some ready-to-use queries which can be run against the database.

psephology.query.**constituency_winners**()
A query which returns the Constituency, Voting, winning vote count and total vote count for each constituency. If there was no winner in the constituency, then the Voting, winning vote count and total vote count is None.

The maximum vote count for a constituency is labelled 'max_vote_count' and the total vote count is labelled 'total_vote_count'.

If you intend to get related objects from the Voting, make sure to add an appropriate joinedload() to the options.

E.g.:

```
q = constituency_winners().options(
    joinedload(Voting.party)
)
```

psephology.query.**party_totals**()
A query which returns a Party and a constituency count, labelled 'constituency_count' which gives the number of constituencies that party has won.

Python Module Index

p

Index