Building Web Applications with Flask Documentation

Release 1.0

Michael Twomey

May 08, 2017

Contents

A short presentation on building web applications with flask (http://flask.pocoo.org/).

- Latest docs: https://building-web-applications-with-flask.readthedocs.org/en/latest/
- Source: https://bitbucket.org/micktwomey/building-web-applications-with-flask
- PDF version: https://media.readthedocs.org/pdf/building-web-applications-with-flask/latest/building-web-applications-with-flask.pdf

CHAPTER 1

Colophon

- Code hosted on Bitbucket: https://bitbucket.org/
- Docs built using sphinx: http://sphinx-doc.org/ and http://pythonhosted.org/sphinxcontrib-httpdomain/
- Docs hosted on Read the Docs: https://readthedocs.org/
- Builds driven using make :)
- Tests using py.test: http://pytest.org/latest/
- Database portion uses SQLAlchemy: http://www.sqlalchemy.org/
- **Running code on Heroku: http://building-webapps-with-flask.herokuapp.com/**
  - Using hg-git to push to heroku, my .hg/hgrc:

```
[paths]
default = ssh://hg@bitbucket.org/micktwomey/building-web-applications-
↪with-flask
heroku = git+ssh://git@heroku.com:building-webapps-with-flask.git

[extensions]
hgext.bookmarks =
hggit =
```

  - **Postgresql added using:**
    - heroku addons:add –app building-webapps-with-flask heroku-postgresql:dev
    - heroku config –app building-webapps-with-flask | grep HEROKU_POSTGRESQL
    - heroku pg:promote –app building-webapps-with-flask HEROKU_POSTGRESQL_ORANGE_URL

CHAPTER 2

Contents

Basics

Creating basic apps is easy, you just decorate a function with a route and you're pretty much done.

pythonie.simple.**index**()

As a bonus flask comes configured with a few things:

- Built in debugger
- Static file serving
- Templates (Jinja2)

pythonie.simple.**broken**()
Show off the built in debugger

Tests

tests.test_pythonie.**app**()
Sets up and returns the app

tests.test_pythonie.**test_blueprints**(*app*)

tests.test_pythonie.**test_database**(*app*)

tests.test_pythonie.**test_index**(*app*)

tests.test_pythonie.**test_signals**(*app*)

tests.test_pythonie.**test_templates**(*app*)

URLs

Bonus: You can use the sphinxcontrib-httpdomain's sphinxcontrib.autohttp.flask extension to automagically generate docs. Note the free static file serving below.

GET /broken
Show off the built in debugger

GET /

GET /static/ (**path:** *filename*)
Function used internally to send static files from the static folder to the browser.

New in version 0.5.

More Advanced

Now to work on the next topic, blueprints. The skeletal app is somewhat similar.

pythonie.application.**index**()
To see this in action go to http://building-webapps-with-flask.herokuapp.com/

All the URLs

The complete app will have all the following URLs:

POST /database/add/
Add a new book via POST

To see in action go to http://building-webapps-with-flask.herokuapp.com/database/add/

Note the use of methods in the decorator to only accept POST.

Parameters

- **title** – The book's title
- **description** – The book's description

GET /signals/not.json
A simple demo of very specific error handling

To see in action go to http://building-webapps-with-flask.herokuapp.com/signals/not.json?password=sekret

GET /blueprints/
Yet another hello world, but this time inside a blueprint

To see in action go to http://building-webapps-with-flask.herokuapp.com/blueprints/

GET /templates/ (*message*)

GET /templates/
Renders a page using a template

Parameters

- **message** – Optional message to display

To see in action:

•http://building-webapps-with-flask.herokuapp.com/templates/

•http://building-webapps-with-flask.herokuapp.com/templates/mick

GET /database/
List all the books in JSON

To see in action go to http://building-webapps-with-flask.herokuapp.com/database/

GET /signals/
A simple demo of authentication

To see in action go to http://building-webapps-with-flask.herokuapp.com/signals/?password=sekret

GET /
To see this in action go to http://building-webapps-with-flask.herokuapp.com/

GET /static/ (**path:** *filename*)
Function used internally to send static files from the static folder to the browser.

New in version 0.5.

Blueprints

Even though many basic apps don't require them I recommend looking into using blueprints to structure your app.

The benefits inclue:

- Easier to follow code with related views kept together
- Code re-usability, blueprints are very self contained (e.g.g templates and behaviour such as authentication)

Creating a blueprint involves:

1. Using flask.Blueprint instead of flask.Flask for your blueprint
2. Registering it in your app using app.register_blueprint

Code

Blueprints let you compose your application from components

pythonie.blueprints.blueprints.**index**()
Yet another hello world, but this time inside a blueprint

To see in action go to http://building-webapps-with-flask.herokuapp.com/blueprints/

For reference here's the application index (and implied link back to the source). You'll notice some use of configuration in that code too.

pythonie.application.**index**()
To see this in action go to http://building-webapps-with-flask.herokuapp.com/

URLs

GET /blueprints/
Yet another hello world, but this time inside a blueprint

To see in action go to http://building-webapps-with-flask.herokuapp.com/blueprints/

GET /
To see this in action go to http://building-webapps-with-flask.herokuapp.com/

Templates

Flask comes with Jinja2 support out of the box. Even better it makes it really easy to use templates from within blueprints too.

Code

It's a little clearer if we look at the full source code too: https://bitbucket.org/micktwomey/building-web-applications-with-flask/src/tip/pythonie/blueprints/templates Examples of templates

`pythonie.blueprints.templates.`**`index`**(*message='from a template'*)

Renders a page using a template

Parameters **`message`** – Optional message to display

To see in action:

- http://building-webapps-with-flask.herokuapp.com/templates/
- http://building-webapps-with-flask.herokuapp.com/templates/mick

URLs

`GET /templates/` (*message*)

`GET /templates/`

Renders a page using a template

Parameters

- **`message`** – Optional message to display

To see in action:

- http://building-webapps-with-flask.herokuapp.com/templates/
- http://building-webapps-with-flask.herokuapp.com/templates/mick

`GET /`

To see this in action go to http://building-webapps-with-flask.herokuapp.com/

Signals

Flask allows you to act on events and customise behaviour using signals.

Signals require Blinker to be installed, though many app hooks don't use signals, just a list of callables.

Signals vs Hooks

Flask signals use Blinker and are usually informational (e.g. you want to watch for errors and log them).

Flask hooks (usually spotted by being methods on blueprints or apps) don't require Blinker and allow you to modify the request or response. These change the behaviour of the app (or blueprint).

Typically you want hooks for changing behaviour (e.g. authentication or error handling) and signals for recording events (e.g. logging).

Caveat

I got bitten by the difference between flask.request_finished and flask.got_request_exception, the former doesn't fire when there is an error (HTTP 500) as Flask doesn't hit that part of the code, while got_request_exception fires on all exceptions. I wound up putting two handlers in place.

Flask 0.9 Lifecycle

Flask 0.9 full_dispatch_request():

```
request_started.send(app) -> signal
rv = preprocess_request()
  rv = [fn() for fn in before_request_funcs (@before_request)]
(rv = dispatch_request() calls actual view)
except: rv = handle_user_exception(e)
  rv = [fn(e) for fn in error_handler_spec[e | status_code] (@errorhandler)]
(response = make_response(rv) uses response_class)
response = process_response(response)
  response = [fn(response) for fn in after_request_funcs (@after_request)]
request_finished.send(app, response=response) -> signal
```

Flask 0.9 hooks to modify content:

```
@before_request (can give its own response, e.g. auth denied)
@errorhandler(e) (can work off exception type or status code, can set its own␣
↪response)
@after_request(response) (can override the response)
```

Flask 0.9 signals:

```
request_started.send(app)
got_request_exception.send(app, exception=e)
request_finished.send(app, response=response)
request_tearing_down.send(app, exc=exc) (@teardown_request(exception) (always called␣
↪at the end, possibly passed an exception)
```

Code

Signals let you change the behaviour of your app or blueprint

pythonie.blueprints.signals.**authenticate**()
: Performs authentication based on HTTP params

  Looks for a password param.

pythonie.blueprints.signals.**handle_errors**(*e*)
: Ensure exceptions always return JSON errors

  Note how this is registered with either an exception type or a HTTP code.

pythonie.blueprints.signals.**index**()
: A simple demo of authentication

  To see in action go to http://building-webapps-with-flask.herokuapp.com/signals/?password=sekret

pythonie.blueprints.signals.**notjson**()
: A simple demo of very specific error handling

To see in action go to http://building-webapps-with-flask.herokuapp.com/signals/not.json?password=sekret

URLs

GET /signals/not.json
A simple demo of very specific error handling

To see in action go to http://building-webapps-with-flask.herokuapp.com/signals/not.json?password=sekret

GET /signals/
A simple demo of authentication

To see in action go to http://building-webapps-with-flask.herokuapp.com/signals/?password=sekret

GET /
To see this in action go to http://building-webapps-with-flask.herokuapp.com/

Database

This isn't really something flask comes with, but it's a good demonstration of using parts of flask to manage database connections.

This code is specific to the blueprint, you can potentially mix completely different databases and transaction semantics in one application.

Code

Example of using signals to manage a database connection

pythonie.blueprints.database.**add**()
Add a new book via POST

To see in action go to http://building-webapps-with-flask.herokuapp.com/database/add/

Note the use of methods in the decorator to only accept POST.

Parameters

- **title** – The book's title
- **description** – The book's description

pythonie.blueprints.database.**connect**()
Creates a per request connection and transaction

pythonie.blueprints.database.**disconnect**(*exception*)
Commits or rolls back the transaction and disconnects

pythonie.blueprints.database.**index**()
List all the books in JSON

To see in action go to http://building-webapps-with-flask.herokuapp.com/database/

pythonie.blueprints.database.**init_db**()
Creates the initial database connection

Fired before the first HTTP request (to any part of the site).

tests.test_pythonie.**test_database**(*app*)

URLs

POST /database/add/
Add a new book via POST

To see in action go to http://building-webapps-with-flask.herokuapp.com/database/add/

Note the use of methods in the decorator to only accept POST.

Parameters

- **title** – The book's title
- **description** – The book's description

GET /database/
List all the books in JSON

To see in action go to http://building-webapps-with-flask.herokuapp.com/database/

GET /
To see this in action go to http://building-webapps-with-flask.herokuapp.com/

HTTP Routing Table

/

GET /, 8

/blueprints

GET /blueprints/, 7

/broken

GET /broken, 6

/database

GET /database/, 11
POST /database/add/, 11

/signals

GET /signals/, 10
GET /signals/not.json, 10

/static

GET /static/(path:filename), 7

/templates

GET /templates/, 8
GET /templates/(message), 8

Python Module Index

p

t

Index