Cardiff's School of Computer Science & Informatics

CM1102 "Web Applications"

Flask 1

Dr Natasha Edwards

With thanks to Dr Ian Cooper

Quickstart

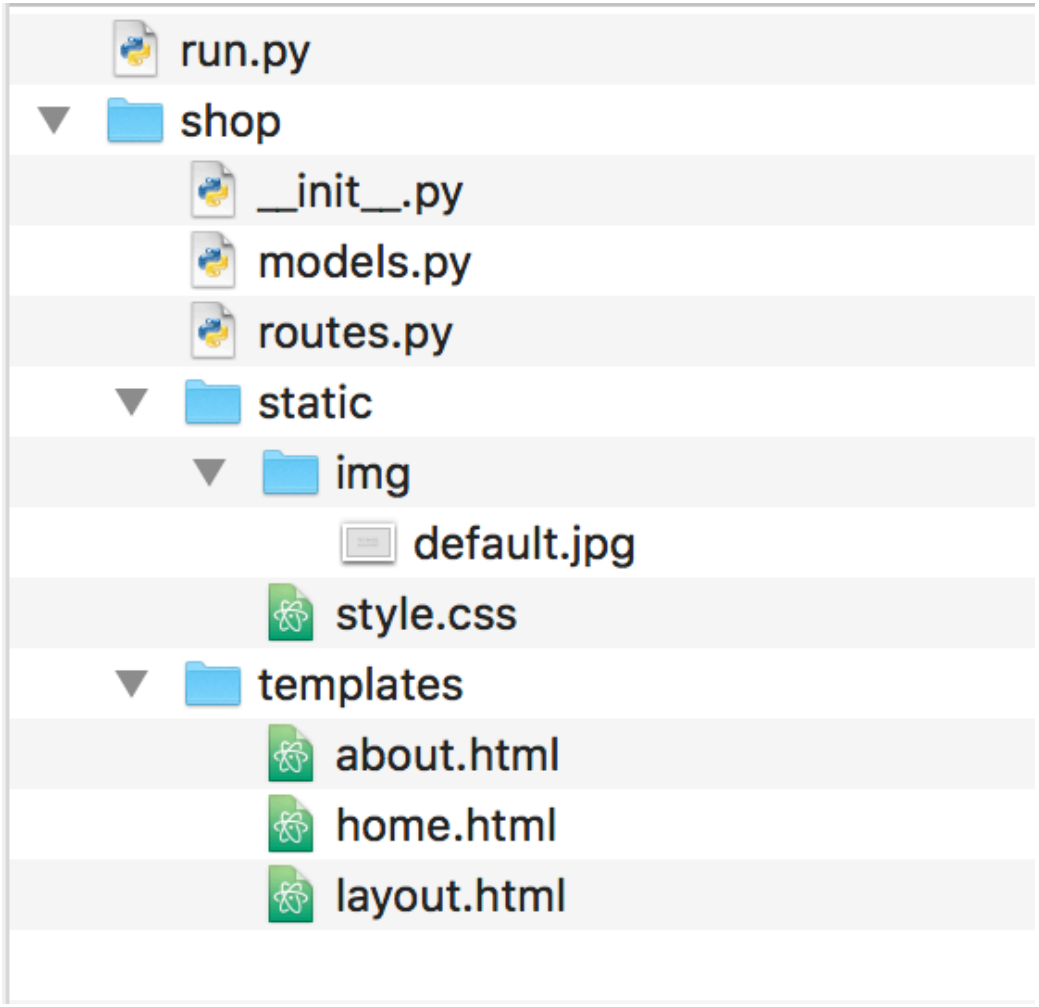
http://flask.pocoo.org/docs/1.0/quickstart/

Minimal application

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, World!'
```

Example structure of project



To run an application

On Windows

```
C:\path\to\app>set FLASK_APP=hello.py
```

On UNIX

```
$ export FLASK_APP=hello.py
$ flask run
 * Running on http://127.0.0.1:5000/
```

Routing

- Seen routes before in the browser urls
  - x.x.x.x:5000/Home.html
  - x.x.x.x:5000/IMC/Home.html
- Routes lead to execution of code
  - Go to the file system and retrieve page
- Routes could request data
  - or an action

use route() decorator

```
@app.route('/')
def index():
    return 'Index Page'

@app.route('/hello')
def hello():
    return 'Hello, World'
```

Redirect:

- Redirect a route to a static page

```
from flask import Flask, redirect

@app.route("/redirect")
def redirectToStatic():
    return redirect("/static/hello.html")
```

Templating

```
with Jinja
```

http://jinja.pocoo.org/docs/2.10/

What is templating?

- A tool that constructs html pages.
- Can get server side and client side templating.
- Jinja is Server side.
- It turns:

```
<ul>
 {%for value in
nums %}
    <li>{{value}}</li>
 {% endfor %}
</ul>
```

this into this

```
<ul>
   <li> one  </li>
   <li> two  </li>
   <li> three  </li>
   <li> four  </li>
   <li> five  </li>
   <li> six  </li>
   <li> seven  </li>
   <li> eight  </li>
   <li> nine  </li>
</ul>
```

What is templating: dynamic data

- The python list ‘nums’:
  - Can be created in the server and passed to the templating engine
  - This can be done when the page is requested.

```
<ul>
 {%for value in  nums %}
   <li>{{value}}</li>
 {% endfor %}
</ul>
```

Rendering a template

- Template files must be in the template directory.
- http://flask.pocoo.org/docs/1.0/quickstart/#rendering-templates

```
from flask import Flask, request, render_template
.....
@app.route("/Basic", methods=['GET'])
def returnFirst():
    if request.method == 'GET':
        return render_template('0_Basic.html',
data='Hello World')
```

Jinja Basics:

- {{ data }}
- {% command %}
  - If

```
...
<h1>
    {%if data%}
      {{ data }}
    {%endif%}
  </h1>
...
```

```
return render_template('0_Basic.html', data = 'Hello World')
```

Dictionaries

- We can access data from dictionaries

```
…
 <h1>
     {{ data.title }}
  </h1>
  <p> Hi this is a message from
{{data.name}}.</p>
  <p>{{data.message}}</p>
…
```

```
messages = {'title':'HelloWorld',
                    'name':'Ian',
                    'message':'more'}
return render_template('1_json.html', data =
messages)
```

Loops: lists

- Looping through a list

```
…
 <ul>
  {%for day in  days %}
    <li>{{day}}</li>
  {% endfor %}
</ul>
…
```

```
…
days = ['mon','tues','wed','thurs','fri']
return render_template('2_loops.html', days =
days)
…
```

Loops: Dictionaries

- data.items()
- Single var vs two vars

```
<ul>
 {%for key,value in  data.items() %}
    <li>{{value}},{{key}}</li>
  {% endfor %}
</ul>
```

```
data = {'title':'HelloWorld',
                'name':'Ian',
                'message':'hi!',
                'days': ['mon','tues','wed','thurs','fri']
                }
return render_template('3_dictionary.html', data =
data)
```

Inheritance

- A base file can be extended:

```
{%extends ‘base.html'%}
```

- Blocks can be overriden

```
{% block myBlock%}Bla Bla Bla
{%endblock%}
```

- Or Called…

```
{{ super() }}
```

http://flask.pocoo.org/docs/1.0/patterns/templateinheritance/#template-inheritance

More functionality and options

- E.g. filters

  http://jinja.pocoo.org/docs/2.10/

Models

- ORM - Object-relational mapping
  - converting data between different types of systems, which are not necessarily compatible with each other.
  - uses OO programming languages
  - Flask-SQLAlchemy
    - http://flask-sqlalchemy.pocoo.org/2.3/models/

Simple example

```
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), unique=True, nullable=False)
    email = db.Column(db.String(120), unique=True, nullable=False)

    def __repr__(self):
        return '<User %r>' % self.username
```

models.py

One-to-many relationship

```
class Person(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(50), nullable=False)
    addresses = db.relationship('Address', backref='person', lazy=True)

class Address(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    email = db.Column(db.String(120), nullable=False)
    person_id = db.Column(db.Integer, db.ForeignKey('person.id'),
        nullable=False)
```

Many-to-many relationship

need to define a helper table

```
tags = db.Table('tags',
    db.Column('tag_id', db.Integer, db.ForeignKey('tag.id'), primary_key=True),
    db.Column('page_id', db.Integer, db.ForeignKey('page.id'), primary_key=True)
)

class Page(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    tags = db.relationship('Tag', secondary=tags, lazy='subquery',
        backref=db.backref('pages', lazy=True))

class Tag(db.Model):
    id = db.Column(db.Integer, primary_key=True)
```

Define and access DB

- Set up DB connections
- Create tables
- Register with the application
- Initialise the DB File

```
> python
```

```
>>> from yourapplication import db
>>> db.create_all()
```

```
>>> db.drop_all()
```

```
>>> User.query.all()
[<User u'admin'>, <User u'guest'>]
>>> User.query.filter_by(username='admin').first()
<User u'admin'>
```

http://flask.pocoo.org/docs/1.0/tutorial/database/

http://flask-sqlalchemy.pocoo.org/2.3/quickstart/