

CM1102 Web Applications
SPRING LABS - WEEK 4

Flask

Part 1: Essentials

This exercise is not assessed. If you do not manage to finish all the tasks in the lab, please attempt to finish them in your own time. If you find the first few tasks too easy – skip to the harder ones.

Use the separate handout and suggested links to various resources to help you understand the code. Several snapshots of the state of the project are available on Learning Central. You can download these to check your progress (and save you typing *from scratch*). However, please make sure you understand each line of the code!

Remember, the lab tutors are here to help. If you get stuck – do not be shy, raise your hand and ask for advice. It is also okay to discuss the solutions with your peers (these labs are not assessed!), however, make sure you understand everything by yourself.


Abbreviations used in this document

`dir` – directory (folder)

`cmd` – command prompt - in these instructions prefixed with the symbol `>`

PRELIMINARIES

1. **IMPORTANT!!** The labs will give you some basic understanding of how to develop a website in Flask. To get more comprehensive understanding of how this 'stuff' works, it is strongly advised that you read the recommended book and complete few tutorials. The links to those are given at the end of the document. However these are just suggestions and the list is non-exhaustive! There are lots of various tutorials on the Web.
2. The technologies and tools we will be using for this lab (as well as other Flask labs) are:
 - Python, Flask, and some libraries specified in the lab instruction documents
 - MySQL
 - Git, GitLab
 - OpenShift
3. If you want to practice these exercises on your own machine, please make sure that the versions of the technologies and tools you have on your machine are compatible with our system setup. These will be specified in the `requirements.txt` file. The other version are untested and not guaranteed to work in our labs, so use them at your own risk.

Note: If you want to work on your own machine and you are not on Cardiff University network, to be able to connect to COMSC MySQL, OpenShift and some other applications, you need to set up a University VPN connection, see COMSC documentation: go to <https://docs.cs.cf.ac.uk/notes/>  and select 'VPN'.

INITIAL SETUP

4. Create the project directory `flask-lab1` in a suitable location, e.g. on **H:** drive.
5. Put the project directory under `git` control.
 - Create `.gitignore` file within your `.git` dir, and include the files and directories you do not want our deployment server to see, e.g. all byte code files (`*.pyc`), `env/`, cache files, etc.
 - Create `flask-lab1` repo on GitLab, and push your local repo to GitLab. Make sure you commit often and push it to the remote repo, and in particular, when you successfully complete each task.
6. Create the **virtual environment**:
 - > `python -m venv venv`
 - and activate it:
 - > `venv\Scripts\activate`.¹

STARTER APPLICATION: "Hello, World!"

Let us first create a simple 'Hello, World!' application.

7. Create `hello.py` file in the project directory with the following content:

```
from flask import Flask

app = Flask(__name__)

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

8. Go back to command prompt:

- (a) Enable the debug mode:
 - > `set FLASK_DEBUG="1"`

Note: whilst the debug mode is useful in development environment, it should be switched off when you deploy your website on the server – as it presents a major security risk!

- (b) Tell Flask where to find your app:

- > `set FLASK_APP=hello`

- (c) Start the development server:

- > `flask run`.

9. Go to: `127.0.0.1:5000` to check that the web page displays your 'Hello, World!' message.

¹On UNIX the command is: `venv/bin/activate`. Use `deactivate` command for switching it off.

Templating

At the moment, our page is just a static one. To fix this, we can use Flask's templating functionality. We also need to do various other changes to start separating the application logic from presentation (remember MVC!).

10. Create dir `templates`, and within it two files `home.html` and `layout.html`.
 - `layout.html` is a template used for presenting information on our website in a consistent manner. It will also be used to contain the website navigation (header, footer, side bar, etc.).
 - `home.html` inherits from `layout.html`, and is used for specifying what data we want to display on this web page.

Routes

11. We have already specified the routing for our home page (see 7). We now need to modify `hello.py` to tell the server to render the templates we are using.

```
from flask import Flask, render_template, url_for
app = Flask(__name__, static_folder="static")

@app.route("/")
@app.route("/home")
def home():
    return render_template('home.html', title='Home')
```

Note: we need to import `render_template` and `url_for` (line 1), as well as tell the app where to look for static files (line 2).

12. Modify `home.html` by formatting the 'Hello, World!' sentence as `<h1>`. Check the output.
13. We can also tell the server where to look for any other pages we want to add to the website – by adding routing information for another page, `about.html` to `hello.py`.

```
@app.route("/about")
def about():
    return render_template('about.html', title='About Us')
```

14. Add some content to `about.html` and check it is working by going to `127.0.0.1:5000/about`.

Navigation

15. Let's add some navigation – by modifying `layout.html`:
 - Add links to the two pages we created, using ``. It is advisable to use Flask's `url_for` to avoid having to change URLs throughout the application, e.g.

```
<a href="{{ url_for('home') }}"... >
```

Styling

Let's add some styling to our web pages.

16. Create dir `static`, which is used for all 'static' files (e.g. CSS, JavaScript, images). Within this dir create a file `style.css` - to be used for styling our web pages.
17. To get Flask serve the static files we need to tell it about the location of static files (i.e. `static` dir) to `hello.py`:

```
app = Flask(__name__, static_folder="static")
```
18. Modify `static/style.css` to change how you page looks like, e.g. change colour for the links, background colour for the page, etc. Check the result.

Hint: you will need to add `<link..>` to `layout.html` in the form of:

```
<link rel=stylesheet type=text/css
      href="{{ url_for('static', filename='style.css') }}">
```

ONLINE BOOKSTORE

Project Organisation

Let's now move to creation of our online bookstore. Before we start looking into implementation of essential functionality (database, etc.), we need to re-organise the project to allow for adding various packages – as well as making our project looking more professional! ²

19. Create a new dir `flask-shop` in a location of your choice (e.g. `H:`), and within it a sub-directory `shop`. Copy or move `templates` and `static` dirs to it - from the project directory you created for the `hello` app.
20. Put it under `git` control, and push to the remote, e.g. to a new repo on GitLab.
21. Initialise and activate the virtual environment (see 6).
22. Create `run.py` at the root of your project (i.e. in `flask-shop`), with the following content:

```
from shop import app
if __name__ == '__main__':
    app.run(debug=True)
```

23. Create `SECRET_KEY` ³:

```
> import os
> os.urandom(24)
> print(os.urandom(24).hex()) 4
```

² Unlike some other web development frameworks, Flask doesn't require you to have a specific directory structure for your project. It is possible to have your application in just one file. This can be appropriate for a small project. However, such flat structure of a project might not be desirable or indeed a good practice for a project that requires a substantial number of components, and in particular, if some components can be reused in different projects.

³ Needed for sessions later - see <http://flask.pocoo.org/docs/1.0/quickstart/>  for more information.

⁴ This gives us a hex string.

24. In the `shop` dir:

(a) Create `__init__.py`:

```
from flask import Flask

app = Flask(__name__)
app.config['SECRET_KEY'] = '<paste SECRET_KEY here>'

from shop import routes
```

(b) Create `routes.py` in the `shop` dir, and move the routing information from `hello.py` to this file, i.e. the two routes for `home` and `about` (see 11, 13).

```
from flask import render_template, url_for
from shop import app

@app.route("/")
@app.route("/home")
def home():
    return render_template('home.html', title='Home')

@app.route("/about")
def about():
    return render_template('about.html', title='About')
```

Note: we have now added some other imports (lines 1 and 3).

DATABASE (DB)

Initial Setup

25. Make sure you still have the virtual environment active.

26. Install Flask-SQLAlchemy and PyMySQL:

```
> pip install Flask-SQLAlchemy
> pip install PyMySQL
```

27. Open `__init__.py`, and add DB import and other configurations settings we need to be able to access MySQL database:

```
...
from flask_sqlalchemy import SQLAlchemy
...
# Note: type 'app.config[..]' as one line, it is split into
# multiple lines in this document to fit on the page

# {USERNAME} and {YOUR_DATABASE} are your Cardiff username
app.config['SQLALCHEMY_DATABASE_URI'] =
    'mysql+pymysql://USERNAME:MYSQL_PASSWORD@csmysql.cs.cf.ac.uk:3306/YOUR_DATABASE'

db = SQLAlchemy(app)

...
```

Models

28. Open `models.py` in the `shop` dir, and make sure you understand the database schema.
29. To create DB from models, go to the python shell:

```
>>> python
>>> from shop import db
>>> db.create_all()
```
30. Check that your DB now has two tables: `author` and `book`.

PRODUCT CATALOGUE

To be able to show all the products on our home page, we need to modify `routes.py` and `home.html`:

31. In `routes.py` add an import for the models:

```
...
from shop.models import Author, Book
...
```





and to the routing for home:

```
...
books = Book.query.all()
...
```

32. In `home.html` tell the server to display all the products's titles, authors and prices – by using for loop:

```
...
{% for book in books %}
  <p>{{ book.title }}" &nbsp;
    Author: {{ book.author.first_name }}
    &nbsp;{{ book.author.last_name }}</p>
  <p>Price: {{ book.price }}</p>
{% endfor %}
...
```

33. Populate your DB with few products, using either the command prompt or a GUI of your choice.
34. Go to the home page and check everything is working as intended.
35. You can now start working on further modification of your website to implement a *'look and feel'* you would like it to have.

Useful resources	
COMSC database server:	csmysql.cs.cf.ac.uk (MySQL)
COMSC phpMyAdmin:	https://www.cs.cf.ac.uk/phpMyAdmin 
Flask Website:	http://flask.pocoo.org/docs/1.0/ 
Flask Tutorial:	http://flask.pocoo.org/docs/1.0/tutorial/ 
SQLAlchemy Quick Start:	http://flask-sqlalchemy.pocoo.org/2.3/quickstart/ 
Video tutorials you might find useful (<i>as usual, in no particular order!</i>)	[1] , [2] (these links are to the first lessons in a series).
A number of cheat sheets could be found on the web.	