Cardiff University
School of Computer Science & Informatics

Dr Natasha Edwards

CMT120 Fundamentals Of Programming

AUTUMN - PGT WK 9

Flask
Part 1: Essentials

Lab Objectives

Objectives of these exercises are to start developing websites using Flask, by

- implementing *'Hello, World!'* starter application;
- creating templates and specifying routing;
- creating an initial setup for 'Blogging website' application: project organisation, database connection, product catalogue;
- setting up connection to databases.

General Comments

This exercise is not assessed. If you do not manage to finish all the tasks during the live session, 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 lecture notes 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 teaching team is here to help - just ask for advice. It is also okay to discuss the solutions with your peers (these exercises 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 exercises 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 use the recommended resources (textbook, documentation, and other suggested resources) as well as 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 in this practical sessions (as well as other Flask sessions) are:
   - Python, Flask, and some libraries specified in the lab instruction documents
   - MySQL
   - Git
   - 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. The other version are untested and not guaranteed to work in our labs, so use them at your own risk. If unsure - just ask for advice!

   NB: You need to use a VPN connection 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'.

INTIAL 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 sever 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**:

   > `py -m venv venv`(1)

   activate it:

   > `venv\Scripts\activate`.(2)

   and then install flask: > `pip install flask` (3) (4)

   Note: if all the dependencies needed for the project are known and are specified in the `requirements.txt` file, you can install these recursively, using the following command:

   > `pip install -r requirements.txt`(5)

   NB: Make sure `requirement.txt` is in the current working directory. (Typically, this file is placed in the project's root directory.) See the lecture notes for details on how to export the project dependencies into `requirements.txt` file.

(1) If this command does not work, try: `python -m venv venv`
(2) On UNIX the command is: `source venv/bin/activate`. Use `deactivate` command for switching it off.
(3) Other libraries necessary for your project are installed using the same command, you just need to replace the package/library name with the one you need.
(4) **NB**: If you are getting an error message when you use `pip` to install a python package, you might need to use `--user` option, i.e. `pip install --user <PACKAGE>`.
(5) You might need prefix this command with '`py -m`', i.e the command will be: `py -m pip install <LIBRARY>`.

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) To enable the debug mode, either use:

   ```
   > 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
   ```

   If that doesn't work:

   (a) Create a file called `wsgi.py` in the same directory as `hello.py`:

   ```
   from hello import app

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

   (b) 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.

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.

    (a) In `layout.html` input the following code:

    ```html
    <!DOCTYPE html>
    <html>
    <head>
      <meta charset="utf-8">
      <title>My Wonderful Site</title>
    </head>

    <body>
      <div id="header">
      </div>

      <div id="content">
        {% block content %}
        {% endblock %}
      </div>
    </body>
    </html>
    ```

    (b) And in `home.html`:

    ```
    {% extends "layout.html" %}
    {% block content %}
      <h1>Hello, World!</h1>
    {% endblock content %}
    ```

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.

    ```python
    from flask import Flask, render_template, url_for
    app = Flask(__name__)

    @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`.
    (a) Add the following to `hello.py`
    ```
    @app.route("/about")
    def about():
        return render_template('about.html', title='About Us')
    ```
    (b) Create a new file called `about.html` in the templates folder with the following contents:
    ```
    {% extends "layout.html" %}
    {% block content %}
        <h1>This is About Us page</h1>
    {% endblock content %}
    ```
14. You can 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 `<a href="..">`. 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') }}">Home</a> 
    <a href="{{ url_for('about') }}">About Us</a>
    ```

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') }}">
    ```

BLOGGING WEBSITE

Project Organisation

Let's now move to creation of our blogging site . 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! (6)

19. Create a new dir `flask-blog` in a location of your choice (e.g. `H:`), and within it create a sub-directory called `blog`. Copy or move `templates` and `static` dirs into it – from the project directory you created for the `hello` app.
20. Put it under `git` control, create a new repository in your GitLab account (7), push to the remote using the commands displayed in the "Push an existing folder" on the GitLab's repository page.
21. If you deactivated your virtual environment, you need to re-initialise and re-activate (see exercise 6 above).
22. Update `wsgi.py` at the root of your project (i.e. in `flask-blog`) to point to `blog` project:

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

23. Tell Flask where to find your new app:
    ```
    > set FLASK_APP=wsgi
    ```
24. If you haven't yet, create the sub-directory called `blog` within `flask-blog`.

25. In the `blog` dir:

    (a) Create `__init__.py`:

    ```
    from flask import Flask

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

    from blog import routes
    ```

    (b) Create `SECRET_KEY` (8):

    ```
    > python
    >>> import os
    >>> os.urandom(24).hex()
    ```

(6) 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.

(7) https://git.cardiff.ac.uk/

(8) Needed for 'sessions' later - see http://flask.pocoo.org/docs/1.0/quickstart/ for more information.

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

```python
from flask import render_template, url_for
from blog 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).

(d) Copy the `templates` and `static` directories from earlier in the lab into the `blog` sub-directory.

26. The structure of folders should look like this:

```
./flask-blog
|   run.py
|
+---blog
|   |   __init__.py
|   |   routes.py
|   |
|   +---static
|   |       style.css
|   |
|   \---templates
|           about.html
|           home.html
|           layout.html
|
\---venv
```

DATABASE (DB)

Initial Setup

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

28. Check you have **Flask-SQLAlchemy** and **PyMySQL** installed and working, and if necessary install in using `pip`: (9)

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

29. 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: typically 'app.config[..]' is one line.  Here, it is split into
# multiple lines in order to fit on this page.

# <USERNAME> is your Cardiff username,
# and <YOUR_DATABASE> is your database you want to use for these labs -
# this is typically prefixed with your Cardiff username (e.g.
↪  'c123456_flasklabs').

app.config['SQLALCHEMY_DATABASE_URI'] =
'mysql+pymysql://USERNAME:MYSQL_PASSWORD@csmysql.cs.cf.ac.uk:3306/YOUR_DATABASE'

db = SQLAlchemy(app)
...
```

Models

30. Create `models.py` in the `blog` dir, and make sure you understand the database schema.

```
from datetime import datetime
from blog import db

class Post(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    date = db.Column(db.DateTime, nullable=False,
    ↪  default=datetime.utcnow)
    title = db.Column(db.Text, nullable=False)
    content = db.Column(db.Text, nullable=False)
    image_file = db.Column(db.String(40), nullable=False,
    ↪  default='default.jpg')
    author_id = db.Column(db.Integer, db.ForeignKey('user.id'),
    ↪  nullable=False)

    def __repr__(self):
```

(9) **NB**: If you are getting an error message when you use `pip` to install a python package, you might need to use `--user` option, i.e. `pip install --user <PACKAGE>`.

```python
        return f"Post('{self.date}', '{self.title}', '{self.content}')"

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(15), unique=True, nullable=False)
    email = db.Column(db.String(120), unique=True, nullable=False)
    post = db.relationship('Post', backref='user', lazy=True)

    def __repr__(self):
        return f"User('{self.username}', '{self.email}')"
```

31. At the top of `routes.py` add an import for the models
    `from blog.models import User, Post`

32. To create DB from models, go to the python shell:

```
> python
>>> from blog import db
>>> db.create_all()
```

33. Check that your DB now has two tables: `user` and `post`, e.g. the commands[10] would be:

```
mysql> USE <YOUR_DATABASE>;
mysql> SHOW tables;
+--------------------+
| Tables_in_DATABASE |
+--------------------+
| post               |
| user               |
+--------------------+


mysql> DESCRIBE user;
+----------+--------------+------+-----+---------+----------------+
| Field    | Type         | Null | Key | Default | Extra          |
+----------+--------------+------+-----+---------+----------------+
| id       | int(11)      | NO   | PRI | NULL    | auto_increment |
| username | varchar(15)  | NO   | UNI | NULL    |                |
| email    | varchar(120) | NO   | UNI | NULL    |                |
+----------+--------------+------+-----+---------+----------------+
```

(10) See last week's exercises on MySQL. Note I use lower case for the commands as this is the only way to display them

```
mysql> DESCRIBE post;
+------------+-------------+------+-----+---------+----------------+
| Field      | Type        | Null | Key | Default | Extra          |
+------------+-------------+------+-----+---------+----------------+
| id         | int(11)     | NO   | PRI | NULL    | auto_increment |
| date       | datetime    | NO   |     | NULL    |                |
| title      | text        | NO   |     | NULL    |                |
| content    | text        | NO   |     | NULL    |                |
| image_file | varchar(40) | NO   |     | NULL    |                |
| author_id  | int(11)     | NO   | MUL | NULL    |                |
+------------+-------------+------+-----+---------+----------------+
```

Blog posts

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

34. In `routes.py` modify the routing for home():

```
...
def home():
  posts = Post.query.all()
  return render_template('home.html', posts=posts)
...
```

35. In `home.html` tell the server to display all the posts, titles and users – by using a `for` loop:

```
...
{% for post in posts %}
  <p>"{{ post.title }}"  
        by {{ post.user.username }}</p>
  <p>{{ post.content }}</p>
{% endfor %}
...
```

36. Populate your DB with few products, using either the command prompt or a GUI of your choice. e.g.:

```
mysql> USE <YOUR_DATABASE>;
mysql> INSERT INTO user (username,email)
    -> VALUES ('johnsmith','john@smith.com');
mysql> INSERT INTO post (date,title,content,image_file,author_id)
    -> VALUES (now(),'Test post','This is a test post','default.jpg',1);
```

or the equivalent in python:

```
> python
>>> from blog import db
>>> from blog.models import User, Post
>>> db.session.add(User(username='johnsmith',email='john@smith.com'))
>>> db.session.commit()
>>> db.session.add(Post(title='My first post',
```

```
            content='Hello world!  This is my first ever blog
            ↪  post!',author\_id=1))
>>> db.session.commit()
```

37. Check the records were successfully inserted into your database. You should get an output exactly like this:

```
mysql> SELECT * FROM user;
+----+-----------+----------------+
| id | username  | email          |
+----+-----------+----------------+
|  1 | johnsmith | john@smith.com |
+----+-----------+----------------+

mysql> SELECT * FROM post;
+----+---------------------+-----------+---------------------+-------------
| id | date                | title     | content             | image_file
+----+---------------------+-----------+---------------------+-------------
|  1 | 2020-01-31 03:04:00 | Test post | This is a test post | default.jpg
+----+---------------------+-----------+---------------------+-------------
+-----------+
| author_id |
+-----------+
|         1 |
+-----------+
```

****

Note: If you run into problems and need to delete the tables from the database, do so in the following order:

```
mysql> USE <DATABASE>;
mysql> DROP TABLE comment; (ignore this query for now, we shall see this
↪  later)
mysql> DROP TABLE post;
mysql> DROP TABLE user;
```

****

38. Go to the home page and check everything is working as intended, i.e.:

Home About Us

"Test post" by johnsmith

This is a test post

39. You can now start working on further modification of your website to implement a '*look and feel*' you would like it to have. Try inserting another post into the database.

Useful resources	
COMSC database server: COMSC `phpMyAdmin`:	`csmysql.cs.cf.ac.uk` (MySQL) `https://www.cs.cf.ac.uk/phpMyAdmin`

Flask Website:
Flask Tutorial:
SQLAlchemy Quick Start:
	`http://flask.pocoo.org/docs/1.0/`
	`http://flask.pocoo.org/docs/1.0/tutorial/`
	`http://flask-sqlalchemy.pocoo.org/2.3/quickstart/`

Video tutorials you might find useful *(as usual, in no particular order!)*	`[1]`, `[2]` (these links are to the first lessons in a series).
A number of cheat sheets could be found on the web.	