Cardiff University
School of Computer Science & Informatics

Flask 2:
Making Blogging Website Dynamic

Lab Objectives

Objectives of this lab are to carry on developing our **Blogging website**, by:

- setting up a **database**;
- adding **individual pages** for blog posts, which are accessed from the home page;

PRELIMINARIES

- As before, this exercise is not assessed, but you should complete all tasks.
- **NB:** It is advisable to do all the work with your virtual environment **activated**.
- Review the **RELIMINARIES** section in Flask 1 instructions - these are relevant to this lab too.

Useful resources	
Snapshots demonstrating code at various points of lab tasks completion:	https://git.cardiff.ac.uk/scmne/flask-labs
Flask Website:	https://flask.palletsprojects.com/en/2.2.x/
Flask Quickstart:	https://flask.palletsprojects.com/en/2.2.x/quickstart/
Flask Tutorial:	https://flask.palletsprojects.com/en/2.2.x/tutorial/
SQLAlchemy Quick Start:	https://flask-sqlalchemy.palletsprojects.com/en/2.x/quickstart/
Also see the 'Useful Resources' section in Flask 1 lab.	

Initial Setup

1. Make sure you have the virtual environment active.

DATABASE (DB)

To add dynamic content and to store our data, we need to set up a database. For this project, we will be using **SQLite** [1], and **Flask-SQLAlchemy** [2] to manage connection to the db [3].

Database Connection

2. Check you have **`Flask-SQLAlchemy`** installed and working, and if necessary install it using **`pip`**: [4]

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

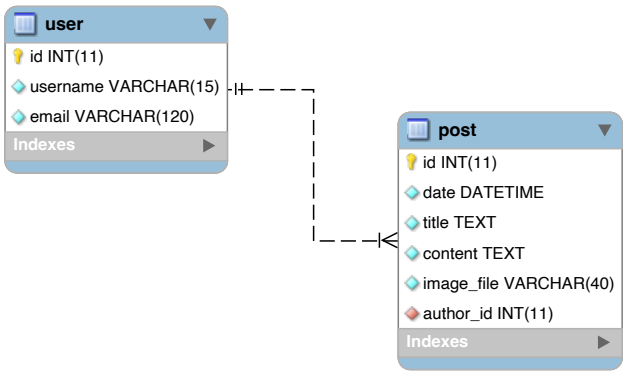
3. Open **`__init__.py`**, and add DB import and other configurations settings we need to be able to access SQLite database:

```
...
import os
from flask_sqlalchemy import SQLAlchemy
...

basedir = os.path.abspath(os.path.dirname(__file__))

app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///' +
↪  os.path.join(basedir, 'blog.db')

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

(1) https://www.sqlite.org/index.html
(2) https://flask-sqlalchemy.palletsprojects.com/en/2.x/
(3) Alternatively, we could use Python's built-in support for SQLite in the **`sqlite3`** module https://docs.python.org/3.9/library/sqlite3.html. However, this is beyond the scope for these labs, and presents an opportunity for independent learning.
(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>`**.

Models

The schema for the db we will be creating in this lab is:



4. Create `models.py` in the `blog` dir, and define two models for `Post` and `User` db tables. Make sure you understand what each line of the code means.

```
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):
        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}')"
```

5. Open `routes.py` and add an import for the models

```
...
from blog.models import User, Post
...
```

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

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

7. A db `blog.db` is now created in the app dir, i.e. `blog`. Check two tables `post` and `user` have been created. In `SQLite` shell:

   (a) Firstly, make sure you are in the `blog` dir,

   (b) and then use the following commands:

```
# to start the shell
$ sqlite3
 ...
# check you are in blog dir:
sqlite> .databases
main: PATH_TO_PROJECT_DIR/blog/blog.db r/w
 ...
# use the db:
sqlite> .open blog.db
 ...
# check the tables are present in the db by listing them all:
sqlite> .tables
post  user
  ...
 # describe the tables (db schema):
 sqlite> .schema
```

   which will return:

```
CREATE TABLE user (
        id INTEGER NOT NULL,
        username VARCHAR(15) NOT NULL,
        email VARCHAR(120) NOT NULL,
        PRIMARY KEY (id),
        UNIQUE (username),
        UNIQUE (email)
);
CREATE TABLE post (
        id INTEGER NOT NULL,
        date DATETIME NOT NULL,
        title TEXT NOT NULL,
        content TEXT NOT NULL,
        image_file VARCHAR(40) NOT NULL,
        author_id INTEGER NOT NULL,
        PRIMARY KEY (id),
        FOREIGN KEY(author_id) REFERENCES user (id)
);
```

NB: if you get am empty set (’`[]`’), i.e. no tables were created, check you completed Task 5.

DB Management

We can now start populating the db with data.

NB: for the tasks in this section, we will be using `sqlite3` command line interface (CLI). You may use other tools, e.g. an SQLite browser. However, using CLI is, arguably, more efficient without an overhead of leaning a new GUI.

8. Make sure you have switch to `blog` dir (where our db is located), and have connected to the db (see Item 7b)

9. Populate tables with some data:

   (a) Insert into `user` table:

```
sqlite> INSERT INTO user (username,email) VALUES ('johnsmith','john@smith.com');

sqlite> INSERT INTO user (username,email) VALUES ('janedoe','jane@doe.com');
```

   (b) and into `post` table:

```
sqlite> INSERT INTO post (date,title,content,image_file,author_id)
        ->  VALUES (datetime('now'),'Test post','This is a test post','default.jpg',1);

sqlite> INSERT INTO post (date,title,content,image_file,author_id)
        ->  VALUES (datetime('now'),'Second post','This is the second post','default.jpg',2);
```

   ****

   NB: Data can also be imported in bulk, by using:

```
sqlite> .mode csv
sqlite>.import <FILE> <TABLE>
  -- e.g. '.import c:/flask/project/app/users.csv user'
```

   ****

10. Check the records were successfully inserted into your db. You should get the following output:

```
sqlite> SELECT * FROM user;
1|johnsmith|john@smith.com
2|janedoe|jane@doe.com

sqlite> SELECT * FROM post;
1|2021-08-02 11:11:11|Test post|This is a test post|default.jpg|1
2|2021-08-02 15:15:15|Second post|This is the second post|default.jpg|2
```

****

NB: If you run into problems and need to delete data or tables, use the following commands:

```
-- to truncate table (i.e. delete all data from the table but not the table itself):
sqlite> DELETE FROM <TABLE>;
   -- e.g.
      sqlite> DELETE FROM post;

-- to delete a table (i.e. both, data and table):
sqlite> DROP TABLE <TABLE>;
```

Further comments:

- You can only drop one table at a time.
- The database can be deleted by removing `*.db` file. However, make sure you keep a backup if case you need it.

****

BLOG POSTS

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

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

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

12. In `home.html` replace

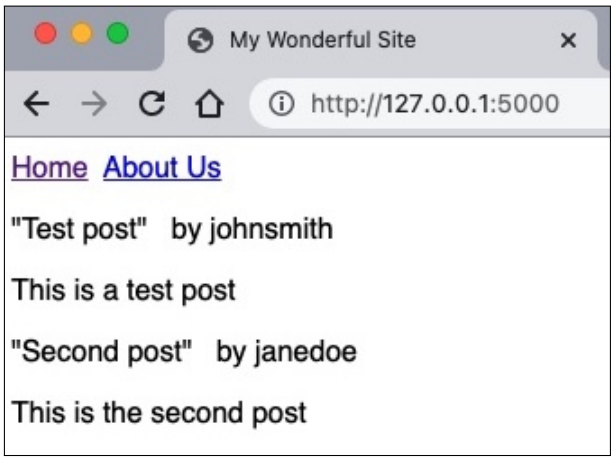
```
...
<h1>Hello, World!</h1>
...
```

with:

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

This tells the server to display all the posts, titles and users, using a `for` loop.

13. Go to the home page and check the home page now displays the data we added to the database, i.e.:



Note: This example does not have any styling - if you applied styling in the previous lab, the content on your page will be displayed differently.

NB: You might need to restart the server for the changes to take effect.

14. Insert few more records in your db and check everything works as intended.

INDIVIDUAL POST PAGES

In our online Blog, each individual post is accessed by using dynamic URLs in the form of `post/<post_id>`, e.g. for the first post in our database with the URL is `post/1`.

To enable our website visitors to access each post's page, we need to:

- create a new `post.html` template,
- and then update `home.html` and `routes.py`.

15. In `blog/templates` dir, create an empty `post.html`, make sure it inherits all the elements of our site's layout (i.e. navigation, etc.), i.e.:

    (a) In `{% block content %}` section of the page, specify that we want the page to display each post's image, content, titles and author (similar to what we did previously in `home.html` page).

```
...
<img src="{{ url_for('static', filename='img/' + post.image_file) }}">
<p>"{{ post.title }}"   Author: {{ post.user.username }}</p>
<p>{{ post.content }}</p>
...
```

16. In `home.html`, update the template in such a way that when the user clicks on a post's title that post's individual page is displayed. This is accomplished by using `a href`, e.g.:

```
...
<a href="{{ url_for('post', post_id=post.id) }}">
...
```

17. Now, we need to update `routes.py` to tell the server where to redirect to `post.html` when the user clicks on the post's title:

```
...
@app.route("/post/<int:post_id>")
def post(post_id):
    post = Post.query.get_or_404(post_id)
    return render_template('post.html', title=post.title, post=post)
```

18. Create `img` subdirectory in the `static` dir. Find a suitable image, name it '`default.jpg`' and put `img`, i.e.:

```
./PROJECT_DIR
|
+---blog
    |
    +---static
        |
        \---img
                default.jpg
```

19. Go to your website to test it works by clicking on a post's title to check that it redirects to that post's page. (*Hint: you might need to restart the server.*)

20. Update `home.html` to also enable the user to click on a post's image to redirect to that post's page.
21. Insert few more records in the db to check everything works as intended.

Further Styling

22. Continue working on further modification of your website to implement a '*look and feel*' you would like it to have, and improve usability, e.g. by adding the top navigation bar accessible from each page.
