Web-app Report

Tatiana Huskova
40207956@live.napier.ac.uk
Edinburgh Napier University - Advanced Web Technologies (SET09103)

Abstract

The purpose of this paper is to reflect on the alternative social network web-app project. This project introduces an alternative social network for posting messages and sharing them with the rest of the community. The app development is conducted by using Python Flask micro-framework and additional libraries. The development process has been successful and a working prototype has been developed. It is most likely possible to build an alternative social network in a Python Flask micro-framework and a prototype may serve as a base for further development.

Keywords – advanced web technologies, napier, python, flask, login, message, sqlite

1 Introduction

This web-app is a prototype of an alternative social network, where people can share their posts with the rest of the community anonymously, using just their username. The app consists of three URLs - homepage, log in page and a main content page.

2 Design

Homepage The homepage located at *localhost:5000* has a simple layout and its main purpose is to display the information about the nature of the web-app. A 'Log in' button located in the centre of the page redirects the user to a log in page, which offers more interaction.

Log in The Log in page contains a log in form where users are required to insert their username and password. In the main python file the form is connected to a database of users. Only users whose names are in that database are allowed to see the further content of the web-app. This has been achieved by adding a *@login_required* decorator into the main page route (see appendix A.1). In case someone inserts incorrect data, a flashing message appears under the form. This informs them that the data is invalid and therefore the content cannot be displayed to them.

Content display In case all the data submitted are correct, the user is then redirected into the Welcome page. This contains messages posted by other users, as well as write their own posts and share them via submit form. All the posts are retrieved from the database using a *Models* class (see appendix A.1) and new posts are added there. Once the user is done with reading and posting messages, they can log out. This redirects them back to the homepage where a flashing message telling them that they have been logged out appears.

Database The two databases used in this project are relational databases written using SQLite3, one of the most used libraries, especially in Python[1].

The first database contains data about users - username and password. To ensure that logging into the page is safe for users, a password encryption method has been added (see appendix A.1). This means that the password is not stored in the database as a plain text, but it is hashed instead.

The second database stores posts shared by users. The posts from this table are displayed on a main page once the user logs in. Also, new posts written by users are stored there.

Static files and Templates The web-app uses a template inheritance system, where one main template is being shared within a few other templates - child templates[2]. Each child template inherits all elements from the mother template. However, each of them is adjusted to meet the needs of the content of the specific URL (see appendix A.1).

The web-app uses a simple bootstrap CSS file, which makes it 'look appealing'. It is simple to use and saves a lot of time[3]. The original bootstrap code offers a very simple look, which makes it easier to adapt to a different website content. Therefore the code had to be slightly modified, to display the content in the centre and a colourful background has been added (see appendix A.1).

3 Enhancements

Even though the web-app works well, looking back at the final project there are some things that could be improved.

Responsiveness Making the website responsive would increase the target audience. This particular web-app is aimed for desktop computers, therefore the interaction with it on a smaller screen may be more complicated.

Sign up Adding a 'Sign up' page and connect it to a user database would allow new users to register. Despite this web-app is inspired by Napier's system, where the users cannot register directly, this element would not only add more user interface into the project, but also increase the final grade for this project.

Log in Current Log in page is very simple. Even though unauthorized access is denied, there is no limitation for username or password. Which means that same usernames can repeat, as well as passwords can be as short as one letter only. Adding a minimum length to the password and/or special characters would help to improve the security.

Images Except a background picture, there have been no other pictures used, since the nature of the web-app does not necessarily require them. However, this is an alternative social network, so allowing users to add their profile picture would be a good additional element.

4 Evaluation

4.1 Critical Evaluation

The part of the code, which may be the most interesting, is the Welcome page, where users can see and share posts from other users. It took a while to make the communication with the database work properly. However, the user experience could be improved by shifting new stories to the top, instead of showing them last.

In general, making the communication with both databases and forms was probably the most challenging part of the entire project. Having it work properly is a success.

There are some issues with Log in page, which have been mentioned earlier, such as a length of a password or repeating usernames. On the other hand, the development of encryption has been successful, therefore passwords are be safe from being stolen by hackers.

As mentioned in Enhancements, creating a Sign up page would add more interaction to the project and would create a more realistic social network environment. However, more time would be needed to add this to the project.

On my opinion, the code as such is well-written, despite some parts that could be re-written in a more logical way. Yet, the code works as desired an that is what matters.

4.2 Personal Evaluation

Same as previous project, the task was quite challenging to do. But with the correct resources and documentation, everything is possible. The two parts I was struggling with most, were password hashing and a database connection. Probably the majority of the time spent working on the project was working on a database connection itself, since all the similar projects used a different type of database (see appendix C). On the other hand, this created a lot of opportunities for me to learn a lot about Flask and its extensions in depth and the ways how to use them properly.

To sum it up, there is still a lot of space for improvement. However I can see a significant progress in the quality of the project since the last assignment. In addition, my knowledge of Flask is much better that I would ever expected to learn in just three months.

References

[1] SQLite-Consortium, "Sqlite." https://www.sqlite.org/docs.html.

[2] A. Ronacher, "Template inhneritance." http://flask.pocoo.org/docs/0.12/patterns/templateinheritance/.

[3] G. Bootstrap, "Bootstrap." https://getbootstrap.com/.

A Appendix A

A.1 Code examples

@login_required A code to create a login extension with a wrap decorator.

Listing 1: Login required code

```
def login_required(f):
  @wraps(f)
  def wrap(*args,**kwargs):
    if 'logged_in' in session:
      return f(*args, **kwargs)
    else:
      flash('Log in to see the content.')
      return redirect(url_for('login'))
  return wrap

```

Models A Models class to get the posts from the database.

Listing 2: Models

```
def getPost():
  con = sql.connect("texts.db")
  cur = con.cursor()
  cur.execute("SELECT auth, stat FROM posts")
  posts = cur.fetchall()
  con.close()
  return posts

```

Encryption An example of password encryption for one user using Bcrypt.

Listing 3: Hashed password storage

```
hashed = bcrypt.hashpw("jackie", bcrypt.gensalt(12))
c.execute('INSERT INTO user VALUES("Jack","' +hashed+ '")')

```

Template inheritance An example of a child template inheriting a mother template.

Listing 4: Child template

```
{% extends "index.html" %}
{% block content %}

  <h1>Welcome</h1>
  <br>
  <p>Welcome to the alternative social network. Share your thoughts, memorable moments from you everyday life and opinions with the rest of the community without the fear of being judged!</p>
  <br>
  <button class="btn"><a href="/login">Log in!</a></button>
  <br>

{% endblock %}

```

Bootstrap adjustments An example of an adjustment made in bootstrap to display the content in the centre.

Listing 5: Bootstrap

```
h1{
  text-align: center;
  }
h2{
  text-align: center;
  }
h3{
  text-align: center;
  }
h4{
  text-align: center;
  }
div{
  max-width:990px;
  border: 2px solid #D2B4DE;
  margin: auto;
  min-height: 830px;
  text-align: center;
  background-color: rgba(215, 189, 226, 0.5);
  }

```

A Appendix B

A.1 Special requirements

Since not everything that has been used in this project is included in a basic Levinux and Python Flask package, there are some additional libraries that need to be installed first.

SQLite3 The database in this project was created using SQLite3 database.

Listing 6: Installing SQLite3

```
$ tce load -wi sqlite3-bin.tcz

```

Flask-Bcrypt Widely used extension for Flas, which helps to keep the sensitive data, such as passwords, safe from the possible attacks from the outside.

Listing 7: Installing Bcrypt

```
$ curl -L -o py_bcrypt-0.4-py2.7-linux-i686.egg https://www.dropbox.com/s/x2vs6e9trv1yuon/py_bcrypt-0.4-py2.7-linux-i686.egg?dl=1
$ sudo su
# easy_install py_bcrypt-0.4-py2.7-linux-i686.egg
# exit
$python -c "import bcrypt"

```

A.2 Additional material used

In this project and additional material by third parties was used.

Background picture The original of the picture can be fount at:

https://weheartit.com/entry/163419196

Bootstrap The original CSS file can be downloaded from:

https://startbootstrap.com/template-overviews/bare

A Appendix C

A.1 Tutorials and Similar projects

The code was built with the help of many different tutorials and similar projects available online. However, none of the codes has been copied and used as my own project. Everything has been used as a guide and inspiration.

Flask Documentation - Pocoo A webpage that offers basic Flask syntax, examples and its usage.

http://flask.pocoo.org/docs/0.12/tutorial/

User Authentication A tutorial by Michael Herman shows how to create the user authentication.

https://www.youtube.com/watch?v=$_pzMDIi5BuIlist = PLLjmbh6XPGK4ISY747FUHXEl9lBxre4mMindex = 21$

SET09103 Workbook A workbook by Dr Simon Wells used during tutorial classes. Helpful while building many different parts of the code.

http://doi.acm.org/10.1145/1273445.1273458

Messaging page using database A similar project - database based messaging system.

https://blog.pythonanywhere.com/121/

Twitter like site A similar project - sharing posts on a wall and user authentication.

http://peewee.readthedocs.io/en/latest/peewee/example.html

Password encryption A blog that explains in a very simple way how to use Bcrypt to hash passwords.

http://dustwell.com/how-to-handle-passwords-bcrypt.html