Web Application Development

Coursework 2

Jacob Holland (sc15j3h)

Website Summary

This purpose of this website is to generate and store shortened URLs of other websites (in a similar manner to bit.ly), This implementation takes advantage of the flask web framework, and includes web forms, a many-to-many database relationship, the use of logon session and cookies, authentication, styling (via bootstrap and a custom stylesheet), unit testing.

Evaluation of Implementation

In terms of primary functionality, the website serves its purposes. It uses html5 form validation to make sure a URL is parsed before it is shortened. However, it doesn't check for invalid URLs – this could be circumvented by requesting the URL and observing a server's response code. If it returned an error such as 404, then we would refuse to shorten the link and notify the user. This would however be implemented in a later version.

Session handling is done via the utilization of the flask-login library that allow for the logging in of users. When a user has logged in, any URL they generate will automatically be viewed in the "My Profile" page. The user also has the option to modify some of their details, such as their first/last name, and their password (which requires entry of their old one for verification). I believe that I have provided basic functionality well, and have successfully utilised a many-to-many relationship (as a user can have more than one URL, and a URL can have more than one 'generator'). As this is primarily a link-shortening website, and not a social networking one, I don't believe that that much customisation is required for the user, although it could be later implemented.

To verify the authenticity of my database models, I created a few basic tests using flask's testing framework. I tested if I could successfully create a new account, generate a shortened URL and check if the database would append the entries into a new table. I made sure the code I wrote would past these tests, which allowed for this basic implementation of Test Driven Development to help find problems in my code that I could later fix.

Given more time, I would've liked to incorporate logging into my project. User errors are displayed in the form of flashed messages, but any underlying library issues (with SQLAlchemy or flask itself) could be stored in a log file, which could then be pulled by administration for bug fixing.

Architectural Analysis

The application architecture is broken down into 3 layers (as follows):

- Presentation – The presentation layer is handled by the HTML/CSS templates handled by Jinja2, this allows the information (such as forms and database rows) to be clearly displayed and intuitive to follow – These files are found in the templates/static folders of the project
- Logic – This is what handles the data to pass it between the layers. This is primarily done by the WTF library that deals with gathering data input from the forms, this data is then processed by SQLAlchemy and stored in the appropriate database models. The logic is primarily programmed into the views.py model, but the form handlers and schemas are stored into the forms.py and models.py files respectively
- Data – This is where all the relevant data generated by the website is store. SQLAlchemy deals with the input of information into a database file (powered by SQLite3). The data can be found in the app.db file. If there is a change to the schema, the database can be appropriately updated using db_migrate.py, if the database has not been created, db_create.py can handle this.

Application Deployment

To host this application on the web, I would first need to purchase rights to a domain name, I would then need to set up a server with a static IP that the domain name can redirect to successfully. The server would need to be running a copy of python3 with flask and all accompanying libraries installed. On the server would need to have the python files, the templates, stylesheets and the database files so that user information can be stored

Security

In the age of connected technologies, security is paramount for any application as it will typically be storing sensitive information submitted by users. If this information was acquired by an unauthorised party, other services that the customer might be using could be further compromised – This would lead to a loss of confidence in the service, causing a drop in user base, or even occur in legal action.

To start off, the user should only see the information that is meant for them. This is done in the form of sessions. This tells the website who the person in that specific instance and will appropriately load their data. This session is stored on a cookie on the user's local machine, so attackers would need to spoof the cookie if they wanted to gain information, though this can be difficult to do. Another vulnerability would be that an unauthorised user could try and access the profile page of another user and see their information, or even modify it. This is countered by the fact that view functions for profiles are wrapped around a "@login.required' function, which prevents a user from accessing the page without appropriate authorisation.

Another problem could be server intrusion, where a hacker could infiltrate the server file system directly and acquire the database file (containing user information). Intrusion is typically dealt with a third party, which would monitor network access and prevent any malicious access. Another thing that needs to be done would have to be file encryption. Passwords can be encrypted using a feature known as bcrypt. Unfortunately, due to time constraints, I could not implement this feature, making the website unfit for commercial deployment

References/Resources

The Flask Mega Tutorial (Miguel Grinberg)
https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world

Web Application Development (Samuel Wilson)
https://soc-dm.leeds.ac.uk/