UNIVERSITY OF LEEDS

COMP2011 Web Application Development

Coursework 2 Written Report

Aodhan Gallagher,
201450577

University of Leeds

Statement of Purpose

This web application has been named “The Top 100 Movies Tracker” and the primary aim of the application is to provide end users with an interface that allows them to view the top 100 movies of all time (according to IMDB ratings) and track which movies they have seen.

Upon visiting the application, an unregistered user can view the top 100 movies as well as extra information about them such as: the year they were released, their description, the runtime and the genres they fall under. Upon creating an account and signing in, a user can then mark which movies they have watched and can then view which movies they have already seen as well as the movies they have yet to see.

Accessing the Deployed Website

The deployed website can be accessed at: sc20ag.pythonanywhere.com

Upon accessing the website you will be prompted to enter a username and password, which are shown below:

Username: sc20ag
Password: Unlock2022

Web Application Architecture

A tiered architecture is an architecture that uses a stack of tiers to meet its purpose, where each tier is independent of the implementation of the other tiers and only relies on the specification guaranteed by the contract.

In this web application, a 3-tiered architecture has been used. The three tiers present in this application are presentation, business logic and data access. This has been achieved through the use of Flask, which makes the separation of tiers clear and is the underlying structure of the web framework.

The presentational layer includes aspects of the web application that manage its presentation and includes the code that generates HTML responses, decodes HTTP requests and decodes form data from requests. This has been implemented in the web application by using FlaskWTF for the handling of forms and also the Jinja2 templating engine for handling render templates in the application.

The business logic layer includes aspects of the web application that are specific to the purpose of the application and can also include data validation. This has been included in the web application through the use of the Flask.route functions in the views.py module. Any time an action is taken in the application, from clicking buttons to going to a new page in the website, it is handled by a route which defines various functions that can be enacted on the page, from querying the database and outputting data to be displayed, to logging users out and even validating form data that has been submitted. Furthermore, the routes also handle redirection to the HTML pages on the website.

The data access layer is responsible for communicating with the database and other APIs. This has been implemented in the web application through the use models provided by SQLAlchemy. In the models.py module, a database schema is modelled that defines three tables for storing various data needed for the application to run, including a table for storing user data, a table for storing movie data and a table for storing if a user has watched a movie (association model).

Request Handling & HTTP Features

It is a common interaction on web pages that a page requests that the user submits some sort of data to be processed by the web server, such as login requests where the data sent to the web server is a user's username and password. To send information to the server, the POST HTTP verb is used and the information is then encoded and sent in the body of the request. When a POST request is received by the server, the body of the request is read and the encoded data is recovered and can then be used for processing. This is commonly used to submit information from forms, where a submit action on the form would trigger the encoding of the form and initiates a HTTP POST request to send data to the server.

Request handling for forms occurs frequently within the web application. Through the use of the Flask extension Flask-WTF, the web application has the facility to render and process data received from POST requests. Further configuration for Flask-WTF was added in my web application by including a (cryptographically secure) secret token (SECRET_KEY = 'a-very-secret-secret') in the form data that is submitted and the secret token is provided by the server when the page is delivered to the client. This is done to prevent Cross-site Request Forgery (CSRF). The impacts of CSRF and further details of how it was mitigated can be found in the "Security Issues & Mitigations" section of this report.

With Flask-WTF, whenever a user enters any data in the forms present in the application (such as the registration form or the login form) and presses the submit button, the browser makes a POST request with the POST request containing all the data from the form encoded in the message body. Flask-WTF is then used to easily retrieve this encoded data as the data is automatically decoded and placed in the data attributes of the form class (specified in various routes in views.py). The form data is then used for various purposes such as being submitted to the database or for querying the database.

Other HTTP verbs exist such as GET, PUT, PATCH and DELETE, each with its own purpose and function, however none of these are required in the web application in order for it to run.

Analysis of Web Application

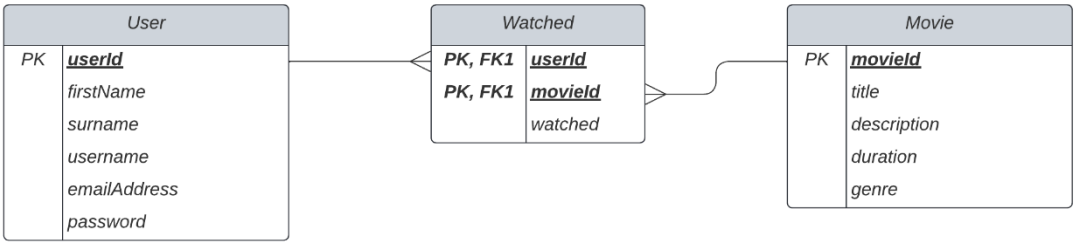
The web application consists of a number of different tools, extensions and frameworks in order to provide users with a fully functional application to successfully track which of the top 100 movies they have seen. An overview of the main implementations of the web application is outlined below.

The application makes use of web forms through the Flask-WTF extension. Two forms were created (in forms.py) to be used in the web application: A registration form and a login form. The registration form contains fields used to store information about a user such as their name, username, email and password. The data entered into the registration form would then be submitted to the database (into the user table).
The login form contains two fields for a user to enter their username and password. The data from this form is then used in a query to find if those details exist in the user table of the database in order to log an existing user into their account. The fields all include different types of validation from restricting the length of user input to ensuring that data is entered by the user for some fields.

SQLAlchemy was used to create a database to be used in the web application. The database consists of three models, including two standard models (“User” & “Movie”) as well as a single association model (“Watched”). When implementing the database, it was decided that two tables would be required. A user table to store all the relevant data about users, which could then be used to create a registration and login system, and a movie table to store all the information about the top 100 movies. These two tables exist in a many-to-many relationship, as many users can watch many movies. To further highlight this relationship, we can say that a single user can watch many movies and a single movie can be watched by many users. Many to many relationships are not supported by most database systems and thus this relationship had to be broken down into two one-to-many relationships. This is where the “Watched” association model was introduced to link the two models through one-to-many relationships and also provide an extra field (“watched”) of the boolean type to indicate whether a user has seen a certain movie or not. This is shown in the entity-relationship (ER) diagrams below.

Many-to-many relationship

Broken down into two one-to-many relationships



Cookies are used in the application for the login system and sessions. To ensure that the application is GDPR compliant, users are informed that the site uses cookies and asks for the user's consent to cookies through a (JavaScript) popup when the user first accesses the web application. This has been implemented through the Cookie Consent extension by InSites.

Furthermore, Flask-login was used to implement user session management into the application and to aid the process of logging in, logging out and managing sessions. Through Flask-login, additional features were added to enhance the website, such as protecting the user sessions as well as restricting views depending if the user was logged in or not. It also aided in retrieving and handling the current logged in user's ID which could then be used in queries to retrieve data from the database, which helped to create the login system.

The majority of styling on the application was performed using bootstrap classes in order to create a standardised layout that is responsive and can be accessed on many different devices. A dark colour scheme was chosen for the website to give it a sleek, modernised look with the main information clearly highlighted on the page.

An advanced feature that was implemented into the application was the inclusion of a JQuery function (in ajaxFunctions.js) that would update the appearance of the "Mark as watched" buttons to provide a clear indication that a film was marked as watch and to enhance the user experience. The function triggers on click of the button and when clicked, the button colour changes from white to green and the text is updated from "Mark as watched" to "Seen".

The web application was successfully deployed using PythonAnywhere. A few minor changes needed to be made to the views.py file in order for the application to work correctly, such as specifying the absolute path instead of the relative path for reading in the movies csv file. Ultimately, the deployed application is fully functional and no errors are present.
Force HTTPS was also enabled to ensure that there were no issues with POST-ing forms, as the CSRF protection in the application would be at risk of not working if the site was not secure.

The application was tested using unit testing to ensure that it was functioning correctly. The unittest python module was used to make a unit testing framework (in unitTests.py) to mainly test the database tables used in the application. Unit tests were defined to set up and configure a test database, as well as to tear down the test database. Furthermore, a unit test was created which adds users to the user table in the test database and checks to see if those users exist, as well as verifying the existence of an unregistered user. Another unit test was defined which added the contents of the movies csv file to the movie table in the test database and checked to see if 100 records / movies existed within the table. This ensured that the csv file was successfully read and its contents were uploaded to the database with no faults.

Logging was also implemented in the web application through the use of the "logging" python module. The application mainly makes use of info logs to track whenever actions are taken on the application or whenever a page is loaded, however warning logs are also present for unsuccessful actions on the website such as failing to login or register a user. Debug logs have also been added to aid in setting up and debugging the application. All of the logs are written to a file called "TopMovies.log" in the root directory of the application and it is set to show the logs specific to the most recent instance of the web application being used.

Accessibility Considerations

The web application was made with simplicity in mind so that it can be easily navigated and used by people of varying technology literacy and by people of all ages.

The navigation bar used to traverse the website's main pages is clearly displayed at the top with text for each page of the website neatly arranged across it (on the left side of the nav bar). Two buttons to log-in and sign up respectively have been included at the right side of the nav bar. This was done to separate them from the rest of the tabs and to have brightly coloured buttons to more clearly highlight the important functions of registering and signing in.

The buttons used to mark an application as watched have a JQuery script associated that activates on click of the buttons. When the buttons are clicked, they update to change from a white button that says "Mark as watched" to a green button that displays "Seen". This was done to highlight to users that the movie was successfully marked as watched.

A dynamic title tag was also used to define the title of each html page and set the title in the browser toolbar. This provides the title for the web page when it is added to favourites and displays it in search engine results, so that the website can be easily found and accessed.

Moreover, information is clearly visible on the website through containers used to house the main information and a colour scheme that makes text and other information stand out and easily read. All the tables and main information displayed in the website at any given time is placed in containers and modals that have darker backgrounds of black and grey, with text that is white. This provides a dark theme that is less harsh on the users eyes but still allows the text and data to visibly stand out.

Furthermore, bootstrap was used to make a responsive layout for the application that maintains a clean user interface regardless of the size of the application window. This allows the application to be viewed on a number of different devices from mobile phones and tablets to desktops, without losing any information or cluttering any of the website's elements (eg by replacing tabs on the nav bar with a dropdown button on smaller devices), thus maintaining a simple design that end users can easily navigate.

Security Issues & Mitigations

Web applications are susceptible to a number of different attacks and security issues, with many web applications being targeted for a variety of different reasons, from accessing user data for identity theft and marketing, to stealing and holding data for ransom value.
To protect the web application, a number of security measures were implemented to mitigate these kinds of threats and make the application more secure. Some of the common vulnerabilities that have been protected against are listed below:

Cross-site Scripting (XSS):
This is where an attacker submits unsanitized JavaScript to a web server. To avoid this, we need to use a technique called escaping to not return unsanitized input to the user. This has been achieved in the web application through the use of Flask, which by default configures the templating engine to escape all input rendered on the page.

Session Management:
Attackers can hijack and exploit authenticated sessions if the authentication and session management has been poorly implemented. The best solution for this is to use a well-tested authentication system instead of implementing your own authentication or session management system. This advice has been followed, as flask-login (an extension of Flask) and Werkzeug-security have been imported and used for secure role management and authentication throughout the application.

Security Configurations:
Many modern web servers have advanced software systems for preventing and detecting intrusions, however these systems are only effective if they are up-to-date and correctly configured. It is wiser to delegate the responsibility of correct configurations to a trusted third party, with many service providers providing pre-configured servers which they manage and web applications can be run on. Because of this, PythonAnywhere was chosen to host the web application as they provide secure pre-configured servers for websites to run on.

Sensitive Data Exposure:
Data in transit or saved to a database should be stored securely, in particular sensitive data such as passwords, credit card information and anything else covered by the Data Protection Act. In the web application, all passwords are encrypted using SHA256 hashing from Werkzeug's generate_password_hash() function and safely stored this way in the database.

Cross-site Request Forgery (CSRF):
CSRF is a security vulnerability which allows a malicious agent to perform actions as if they were the web user. It allows a link to be placed on a website which performs an action on a different website which normally can only be performed when the user has authenticated. Assuming the user has authenticated on the different website, the action will be performed and could be successful. To protect against this on the web application, Flask is used to provide a secret token (`SECRET_KEY = 'a-very-secret-secret'`) in the form data that is submitted. This secret is provided by the server when a page is delivered to the client.

References

University of Leeds. 2022. *COMP2011 Web Application Development.* [Online]. [28 November 2022]. Available from: https://alt-6100e9398f586.blackboard.com/bbcswebdav/courses/202223_32870_COMP2011/site/