Student number: sba22328

Programme: Diploma in Applied Software Development

Course: Web Development (M5)

Lecturer: Hugh McGovern

Deadline: 22/3/2023

This report will outline the project to develop a movie booking website, the decisions and rationale for the development of the site, and problems that were encountered and how they were resolved, where possible. It will be structured across four main sections. At the start, my approach to the problem will be discussed, and then the site map will be covered briefly. After that, the layout of the site will be explained, including any issues that arose in this. Lastly, the report will outline how long- and short-term data storage is managed, as well as also discussing how site maintenance would be approached.

Firstly, tooling will briefly be covered. Whilst the lecturer typically used *Notepad++* in class, I eschewed this for an IDE that I was already familiar with, *IntelliJ IDEA* (Community edition) by Jetbrains. This IDE had the dual benefit of a number of plugins I already used for linting and error-checking, as well as being multi-language, capable of coding simultaneously in HTML, CSS, JavaScript, and PHP, which suited my needs. The SQL server was mediated via a GUI on the web server solution, *XAMPP*. The SQL query section of this SQL server became useful later for testing out SQL scripts before implementing them in PHP to insert bookings into the database. Lastly, I also utilised a dual-monitor setup, with *IntelliJ* on one screen and Chrome on the other displaying the website.

Looking at the problem of the website, I initially decided on the types of functionalities the website needed. Aesthetics were decided upon after that. As far as I could determine, there were a few key areas of functionality that the website required:

1. The ability to populate a table with movie information extracted from an XML file;
2. The ability to create a dynamic page which would be populated by information from specific movies, and which could be cycled back and forward through;
3. The ability to book tickets using a booking form, as well as provide some basic validation for the information entered;
4. Bookings to be stored on an SQL database for permanency.

A later assessment would also require an extension on some of this functionality, but that will not be discussed in this report.

For the first two issues, JavaScript would be used. There was an option to use jQuery to read in the XML file, but in the end vanilla JavaScript could carry out the operation. jQuery was, however, used for some other functionality the site had. To read in the data from the XML file, I used a function that ran when index.html loaded, and then retrieved the XML data using AJAX. From there, it parsed each <movie> entry in the XML file and extracted the information for the movie name, cast, year, and genre, and then built table rows for each of these. The movie names in the table were also to act as URLs to bring the user to a listing page for that movie if they clicked on it; this was to be done using CDATA tags in the XML file. This is one obstacle, it seems, that comes with reading in XML files; the industry standard now seems to be JSON, however, which is more flexible in its ability to store URLs.

The second requirement of dynamically populating a webpage from information in the XML was a lot more challenging for me. In principle, I understood what needed to be done; create a dynamic page, listing.html, that takes information from each entry in the XML file and then populates the webpage with this information for each movie. However, I needed to use some workarounds for this as my JavaScript knowledge was not good enough to carry out this task literally. I could make a movie page and create buttons to move backwards and forwards, cycling through the movies, but did not have the skill to create dynamic links on index.html to do this. With a little more time and learning in JavaScript, this might have been possible, though.

The functionality of the web form to take in ticket bookings was elementary html, with each section of the form given a unique ID which could later be used in the PHP file for inserting into the database. I opted for a drop-down list for the movies as a workaround to the above issue of not being able to dynamically create webpages. The field for the movie date is also set to a date structure with a calendar. It was important to note that in SQL both strings and dates require parentheses when being inserted into a table, unlike numbers, something which I was not aware of before. It was through working with the SQL query GUI on XAMPP (and also additionally *MySQL Workbench*) that I discovered this and was therefore able to correct the problems I experienced in using the PHP file to write the entries to the database (chiefly, syntax errors).

With regards to a general outline of the site, below is an implementation of a basic site plan for how the site should look (with functionality explained above). As can be seen, the overall site is contingent on two to three main pages, with some information shared across them. A header is displayed across the top of all pages, including an ‘About’ button which will bring up some basic information about the project, a ‘Contact’ link to open a new email to the student’s email address with the word ‘Feedback’ to be inserted into the subject line, and a first button which alternates between moving to the listing.html page or to return to the index.html page, where applicable.

A picture containing timeline

Description automatically generated

Figure 1: Site map

Moving on to the third section of the report, the discussion of the site layout, when the website is loaded visitors are brought to the index.html page, which, as above, is divided into two *divs* using *Flexbox*. The left *div* has the table with headings for movie name, genre, cast, and year on top. As before, the table is then populated from an XML file in the same folder, movies.xml. Below this, too, are two search fields to search the listings dynamically using a JavaScript function. By way of making the website more attractive, the right *div* runs an automated slideshow of posters from the movies, using a JavaScript function, and alternates the pictures every 3 seconds. I felt that this made for a more attractive splash page where users could see both the names and details of the movies, but also the visual posters. All the three webpages had the same header at the top and the same background wallpaper for consistency, too, but for the table and some of the other options, backgrounds were used with some opacity settings so that the writing did not blend into the background or disappear. CSS functionality was also implemented to make the website responsive to window resizing or use on other devices.

The booking form page simply had the booking form in the middle of the page in a large style, with some of the fields having placeholder details and some fields set to ‘required’ so they would need to be entered before the form would be complete. This is simple validation which again is to be improved upon in the next continuous assessment.

Moving on to the final section of the report, storage will now be discussed. As the website simply runs locally, with the SQL database on the same location, security of stored data is obviously an issue. To that end, my main computer’s power supply broke during a power surge a number of days ago and had to be repaired (resulting in about 4 days downtime), bringing this issue into sharp relief as there was a risk that the disk may have been corrupted, too. Whilst backing up data on portable media is an option, it is not really a solution. If the website was launched seriously, it would perhaps be a better solution to host the database (and, indeed, the site) on a cloud service such as AWS, Microsoft Azure, or Google Cloud Platform. Furthermore, SQL itself also includes a BACKUP DATABASE statement which could be used daily or hourly to backup the database to an alternative location.

In conclusion, a website like this could be produced in a number of different ways. At a more advanced stage, frameworks like Facebook’s *React* or Google’s *Angular* could be used to produce truly single-page web apps to carry out all the functionality of the site but with less code. However, the website and its requisite functionality are important representatives of basic website operations, including searches, ease of navigation, storage, and use of APIs to deal with external data.

Word count: 1,454