Deliverable 2

An analysis and selection of software architecture, internal design and interface composition

Epic Group

Introduction

The purpose of this report is to clarify and justify the design components of CinemaScout, a web-based movie recommendation platform. The report will consist of the following components:

Part 1

1. **External Data Sources**: An enumeration over the third-party data sources CinemaScout will access.
2. **Software Components**: A list of the major software components that will be required for CinemaScout to function.
3. **Component Choices**: A list and justification of the choice of software components for CinemaScout.
4. **Machine Requirements**: A justification of the machine requirements of CinemaScout.
5. **Summary**: A summary of the entailments the aforementioned software architecture decisions provide.

Part 2

1. **User Stories**: An updated list of user stories relating to the previous report.
2. **Sequence Diagrams**: An interaction diagram for each use case of CinemaScout as described in the user stories.

1 Software Architecture

1.1 External Data Sources

CinemaScout is a movie aggregator which requires comprehensive movie related information to function. Foremost, we require an encompassing and compre-

hensive list of all movies to date. In addition, the user expects each movie to be listed alongisde its creation date, title and rating among other title specific information. This will require an API (application programming interface) which provides such data for all movies in the database of CinemaScout.

There exists no comprehensive API which singly provides all information CinemaScout requires. As a result, we are required to source data from multiple external databases:

- **IMDb**: The Interactive Movie Database provides archive snapshots of its comprehensive movie data sets. However, the data provided is limited in detail as it only contains metadata for each movie in the IMDb database. The dataset of IMDb is sufficient in providing a list of all movies, although specific details such as rating and plot are not provided.
- **OMDb**: The Open Movie Database is a RESTful web service which provides comprehensive movie information for a singular, specific title. OMDb completes our data requirements, compensating for the lack of detail provided by IMDb.

Most critically, the information provided by the database snapshots of IMDb are a requisite of making a query to OMDb. As a result, the two external data sources complement each other and complete our data sourcing requirements.

1.2 Software Components

In general, the components of CinemaScout may be split into two categories: server and client. The client category provides a medium for a user to invoke CinemaScout functionality, while the server category consists of functionality which fulfills solicited requests made by the client category. To this effect, the components of CinemaScout are interdependent and must be interoperable in order to function.

Client Components

- **Web Browser**: A web browser which must support HTML5 (HyperText Markup Language), CSS (Cascading Style Sheets) and Javascript interpretation.
- **Stylesheet**: A responsive and multi-platform CSS framework.

Server Components

- **Webserver**: A HTTP and REST web application framework.
- **Database Manager**: Software capable of sorting and parsing hundreds of thousands of database entries in real time as required by the web server.
- **Database Storage**: A software solution capable of saving hundreds of thousands of database entries into long term storage.

- **HTML Files**: A HTML-based UI allowing a user to perform actions with the webserver through a web browser.
- **Javascript Files**: Software to support the functionality of HTML files, extending the usability of the web interface.
- **Testing Framework**: Software capable of fulfilling the unit test requirements of the project.

1.3 Component Choices

Due to the large volume of data utilised by CinemaScout, the performance of each software component is critical in order to maintain a responsive user experience. Therefore, the use of the C++ language for the webserver component has been deemed as essential. Additionally, half of the development team is fluent with the language, further justifying C++ as a viable language option for server. This choice has influenced much of the other server components.

Client Component Choices

- **Web Browser**: Any browser which supports HTML5, CSS and Javascript to any reasonable degree is compatible with the server.
- **Stylesheet**: Bulma has been selected as a responsive, lightweight and cross-platform CSS framework, enabling the website to function on mobile and desktop devices without specific HTML implementation.

Server Component Choices

- **Webserver**: Pistache has been selected as it is a modern and elegant, asynchronous HTTP and REST framework for C++.
- **Database Manager**: The C++ standard library contains several data structures and algorithms to accelerate server performance and reduce code duplication. The database manager must be developed for CinemaScout to function.
- **Database Storage**: The C++ library RapidJSON has been selected for database storage, as JSON is a lightweight and human readable solution, aiding server performance and software complexity.
- **HTML Files**: The layout of the website administered by HTML files does not exist and must be developed from scratch.
- **Javascript Files**: The functionality of the website, dependent on the layout of the website, does not exist and must be developed from scratch.
- **Testing Framework**: The C++ testing framework GoogleTest has been selected as a modern solution for providing unit tests.

1.4 Machine Requirements

In order to maximise our target audience and minimise running costs, we have decided to enforce lax machine requirements for both the server and client portions of the project. It is necessary that devices meet all of the following criteria for CinemaScout to function correctly.

Client

- **Web Browser**: Any modern browser with Javascript and HTML5 support, along with the ability to store cookies.
- **Internet Connection**: At least 1 MB/s download for images to load timely.
- **Performance**: A device capable of meeting the previous criteria to a reasonable degree.

Server

- **Operating System**: Any recent Linux derivative.
- **Performance**: Dependent on website traffic. CPU speed and memory bandwidth and size should be maximised for faster runtime performance and user session persistence. Low disk speed requirements as it is rarely used aside from during initialisation.
- **Compiler**: A compiler with C++20 support. Capable of building and linking with Boost, libCurl, Pistache, GoogleTest while building with CMake.
- **Internet Connection**: At least 10 MB/s upload and download to anticipate rapid surges of website traffic.

1.5 Summary

The use of C++ as the programming language for the server alongside the native C++ libraries RapidJSON, Pistache, libCurl as well as the C++ standard library will allow the server to deliver accurate movie suggestions in real time for a responsive user experience. The use of JSON as a data storage format will prevent unnecessary database queries, decreasing server boot time and individual request response time.

Bulma as a CSS framework will enable the website to function responsively on both mobile and desktop user browsers. Finally, selecting both OMDb and IMDb as external data sources will enable the user to receive relevant and comprehensive movie information via the frontend.

2 Initial Software Design

2.1 User Stories

- **User Story**: As an avid film watcher, I want to receive tailored, specific recommendations relevant to me.
  Sequence Diagram: See Figure 1.

- **User Story**: As someone who enjoys multiple movie genres, I want to receive recommendations which are not based on my previous viewing sessions.
  Sequence Diagram: See Figure 1.

- **User Story**: As a power user, I wish to be able to receive many movie recommendations over a short period of time.
  Sequence Diagram: See Figure 1.

- **User Story**: As a first time website visitor, I wish to be able to perform a search without offering my credentials and other personal information to first generate an account.
  Sequence Diagram: See Figure 1.

- **User Story**: As a user of the service, I wish to be able to use the service in its full capability without malicious content from ever corrupting my movie suggestions.
  Sequence Diagram: See Figure 1.

- **User Story**: As a user of the service, I do not wish to receive recommendations based on corporate interests or monetary influence.
  Sequence Diagram: See Figure 1.

- **User Story**: As a returning user of the service, I wish to be able to view my recommendations and come back to my search results later.
  Sequence Diagram: See Figure 2.

- **User Story**: As a user of CinemaScout who likes to watch trailers before the film, I wish to be able to view trailers for my recommendations in the CinemaScout website.
  Sequence Diagram: See Figure 3.

- **User Story**: As a user of CinemaScout who wishes to view the results of the search offline, I want to be able to download the search results in a readable format for offline use.
  Sequence Diagram: See Figure 4.

- **User Story**: As a user of CinemaScout who wants to print out my results, I wish to be able to print out a formatted version of the results page tailored for readability on an A4 page.
  Sequence Diagram: See Figure 5.

2.2 Sequence Diagrams

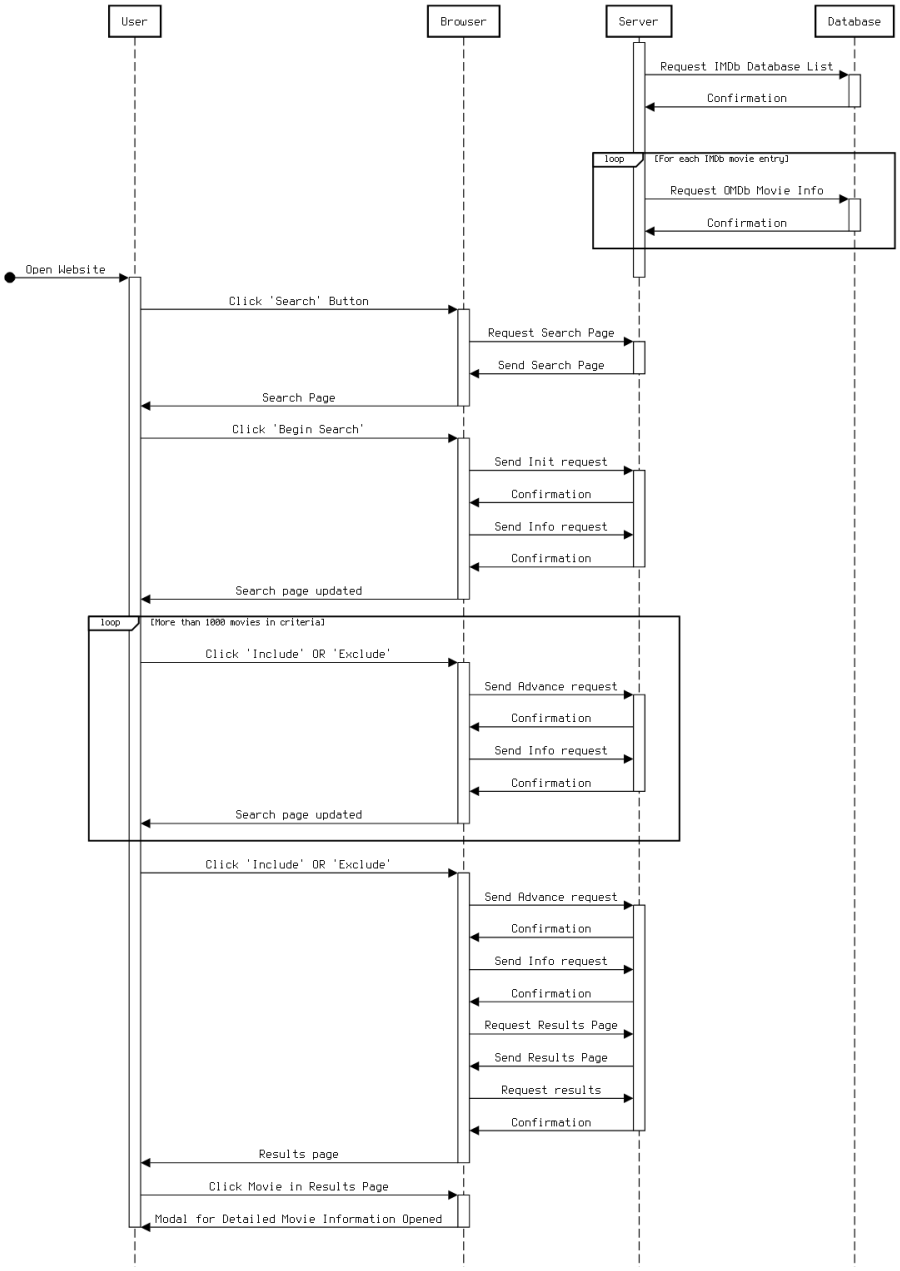

Figure 1: Typical CinemaScout Use Case

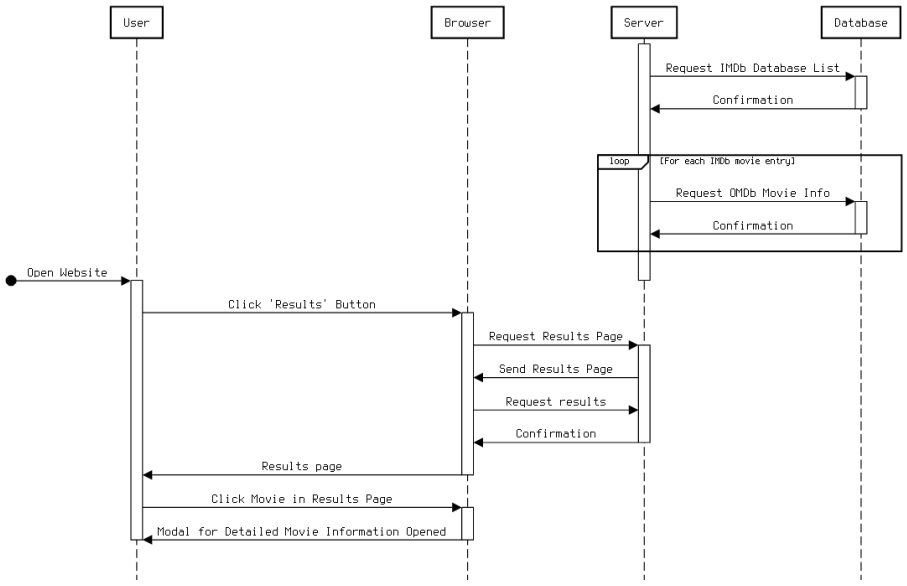

Figure 2: Previous Results Use Case

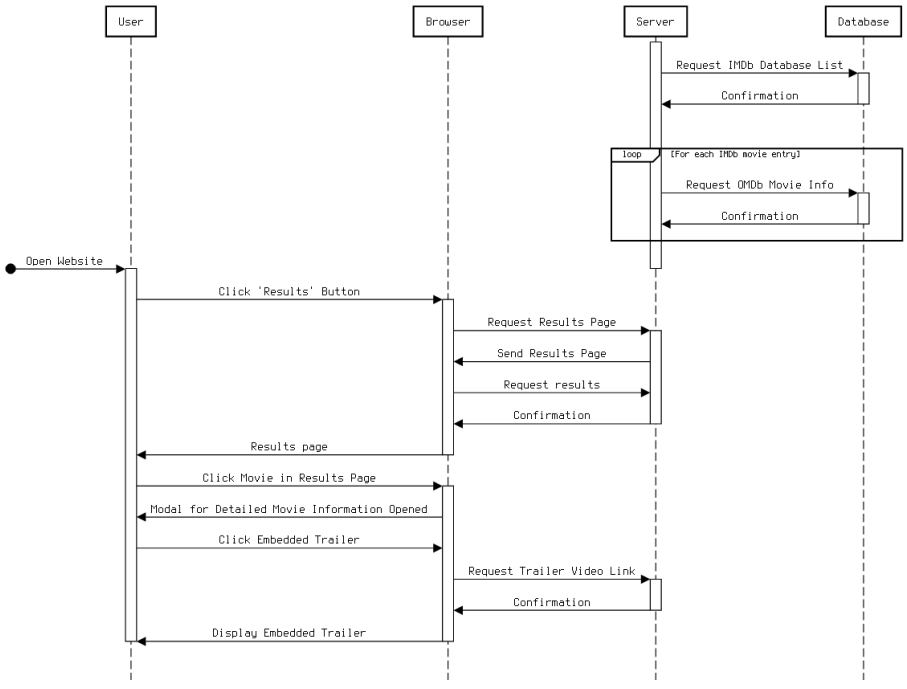

Figure 3: Embedded Trailer Use Case

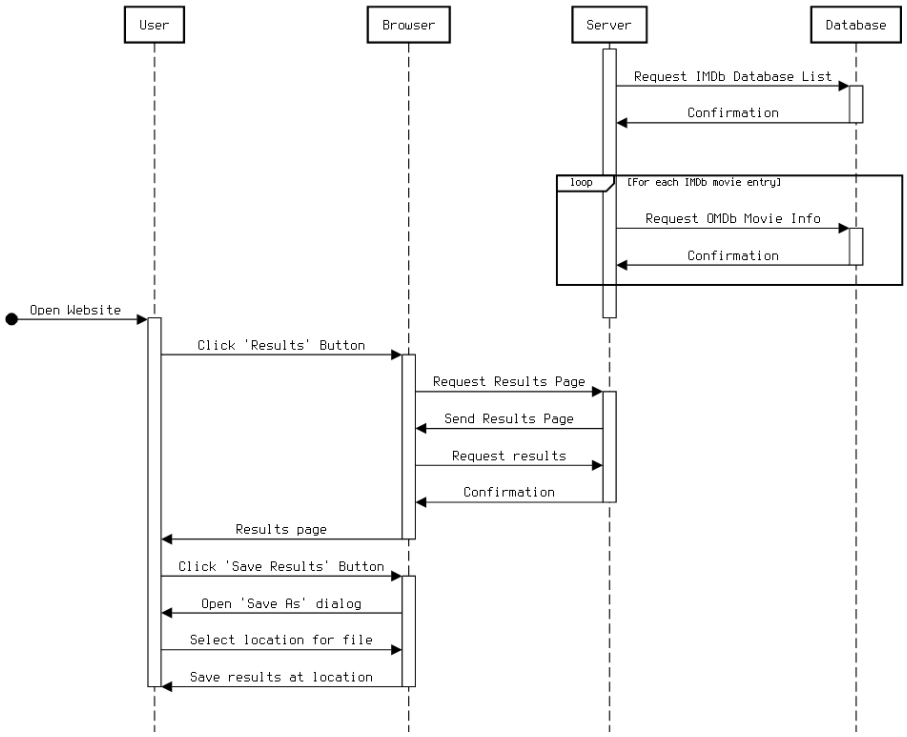

Figure 4: Save to File Use Case

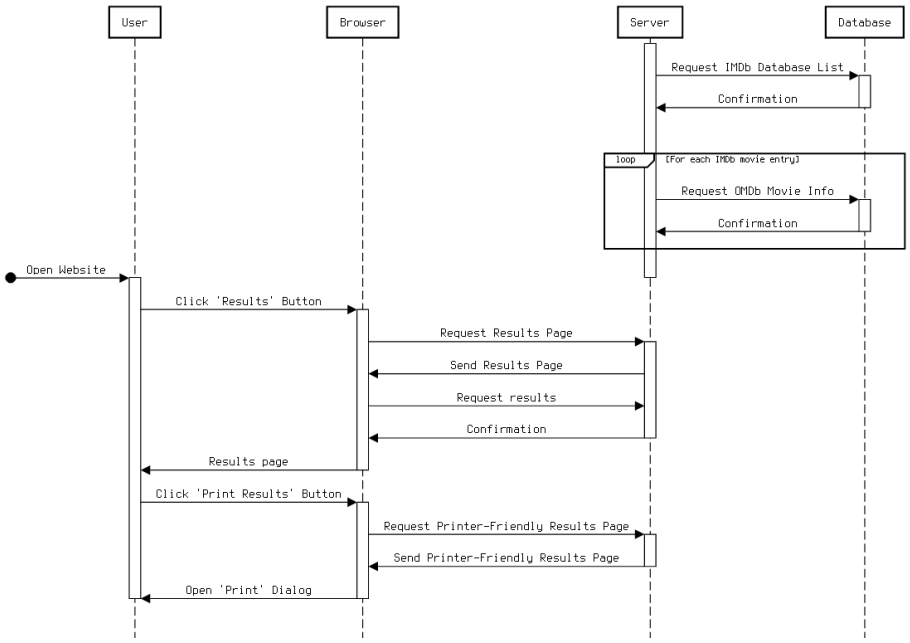

Figure 5: Print Results Use Case