Name: Sahil Salim Mukadam Student ID: 230832271

Email-id: ec24099@qmul.ac.uk Subject: ECS713P – Functional Programming

Project Report: Movie Ticket Booking

Project Overview

This project focusses on creating a multi-threaded movie ticket booking system that replicates the interaction of numerous clients with a server. In order to simulate real-world behavior, the server uses threads to process client requests to purchase tickets for various films concurrently. Request processing, logging, and the creation of interactive HTML reports with visualizations are all crucial features. A comprehensive dashboard that shows booking details and a bar chart for improved visualization and analysis is made possible by the logging tool, which records important data points.

What Does the App Do?

The app implements a basic movie booking system with the following functionalities:

Clients: Multiple clients generate booking requests for movies, specifying the number of seats and the desired movie.

Server: The server retrieves these requests, processes them, and generates a response.

Queues: A request queue holds the incoming booking requests, and a response queue holds the processed responses.

Logging: Detailed logs of each request and response are recorded for tracking purposes.

Steps to Run the Code

Install Haskell: Download and install Haskell from Haskell Platform.

Install Dependencies: Use stack to install necessary dependencies. For example, run stack install to install the required libraries.

Clone the Repository: Clone the project to your local machine using git clone <repository_url>.

Build and Run the Program: Navigate to the project directory and build the project with stack build. Then, execute the program using stack exec <executable_name>.

View the HTML Report: Once the program finishes, open the generated `BookingDashboardWithGraph.html` file in any web browser to view the dashboard with the graph.

Key Features

1. **Multi-threaded Client and Server Model**: At random intervals, a number of clients—each represented by a thread—submit requests for movie tickets. Even while managing several requests at once, the server maintains excellent efficiency by processing these requests concurrently.

2. **Logging Mechanism**: Every client request and server response are recorded by the logging system, which also records important information such the client ID, movie title, booking date, and order confirmation time. For future use, the information is logged in a text file and kept in memory.

3. **Dynamic Movie Selection**: Customers select films at random from a predetermined list. Users can choose any movie they want without any limitations thanks to this function, which simulates a real-world movie ticket booking system.

4. **Interactive HTML Dashboard**: An HTML report is produced after the client-server exchanges are finished. Included in this report is a sortable table that shows the specifics of customer requests and responses. The number of reservations for each film is displayed in a dynamic bar chart. The well-known Chart.js package is used to render the bar chart, which allows users to examine the booking trends visually.

Design Choices

The design of this project was centered around simplicity, scalability, and concurrency. Key design decisions include:

1. **Thread-based Concurrency**: The application uses threads to simulate simultaneous client-server interactions, making the system responsive and capable of handling multiple requests concurrently.

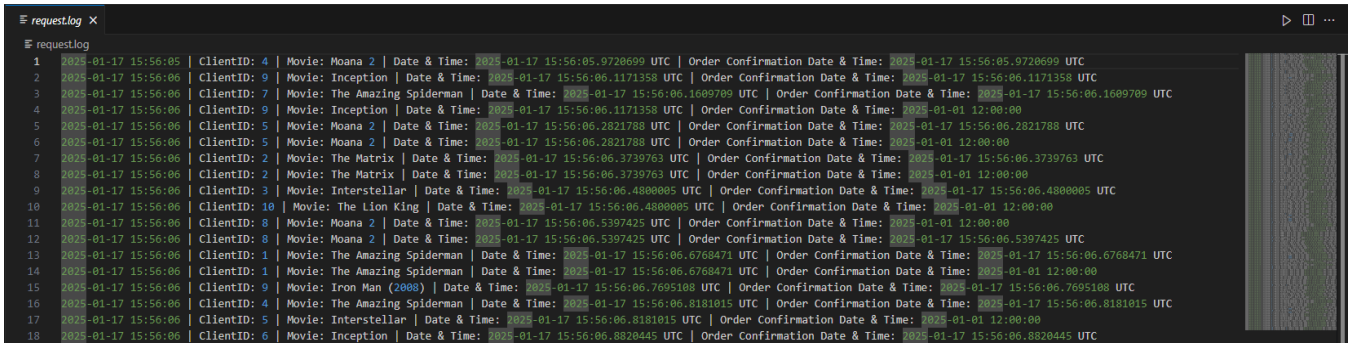
2. **Request Queue with MVars**: A shared request queue implemented using Haskell's MVar ensures thread-safe communication between clients and the server.

3. **Modular Architecture**: The code is divided into well-defined modules (Client, Server, RequestQueue, Booking, and Logging) to ensure maintainability and separation of concerns.

4. **Data Logging for Insights**: A logging mechanism captures request and response details, which are then used to generate HTML reports for post-interaction analysis.

5. **Interactive Reports**: The use of a bar chart and dashboard ensures that logged data is not only recorded but also presented in an intuitive and user-friendly manner, highlighting trends in bookings and server interactions.

Request.log



Leaderboard.html

Movie Ticket Booking Dashboard

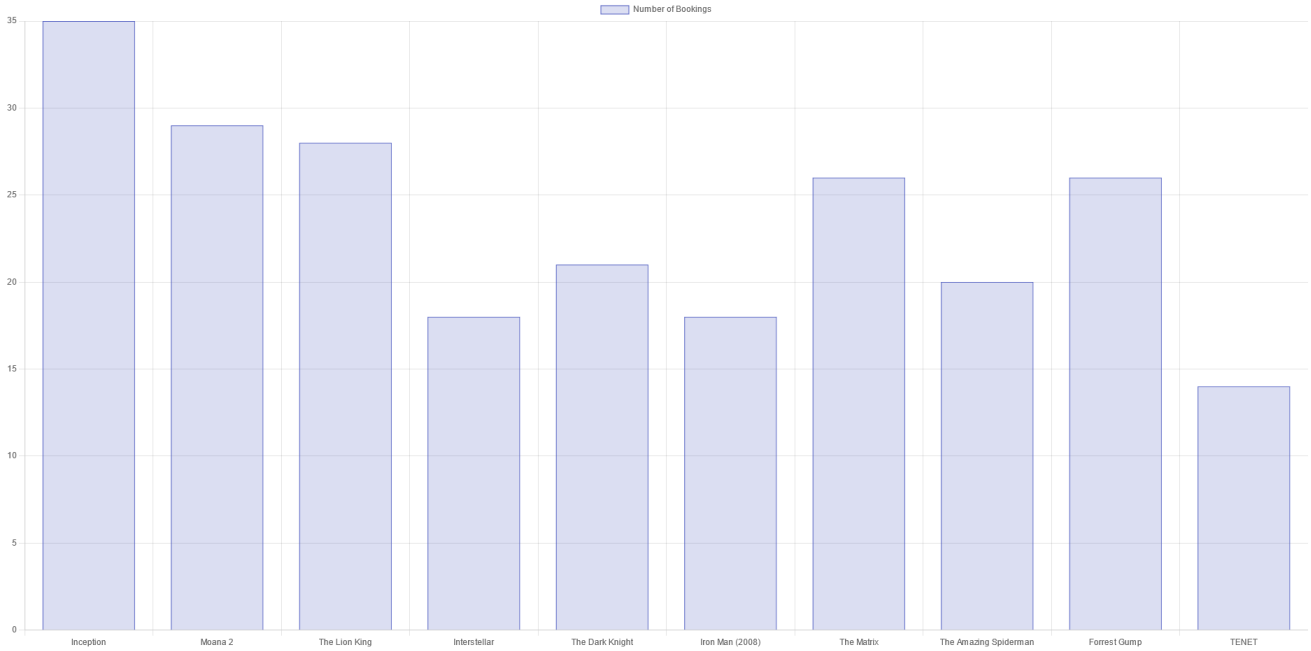


Table Representation

ClientID	Movie Name	Date & Time	Order Confirmation Date & Time
10	Inception	2025-01-17 15:56:13.9489212 UTC	2025-01-17 15:56:13.9489212 UTC
10	Moana 2	2025-01-17 15:56:13.3077829 UTC	2025-01-01 12:00:00
10	Moana 2	2025-01-17 15:56:13.3077829 UTC	2025-01-17 15:56:13.3077829 UTC
3	The Lion King	2025-01-17 15:56:13.1349219 UTC	2025-01-01 12:00:00
3	The Lion King	2025-01-17 15:56:13.1349219 UTC	2025-01-17 15:56:13.1349219 UTC
5	The Lion King	2025-01-17 15:56:12.7526994 UTC	2025-01-01 12:00:00
6	Interstellar	2025-01-17 15:56:12.7247323 UTC	2025-01-01 12:00:00
5	The Lion King	2025-01-17 15:56:12.7526994 UTC	2025-01-17 15:56:12.7526994 UTC
6	Interstellar	2025-01-17 15:56:12.7247323 UTC	2025-01-17 15:56:12.7247323 UTC
2	The Dark Knight	2025-01-17 15:56:12.648739 UTC	2025-01-01 12:00:00
2	The Dark Knight	2025-01-17 15:56:12.648739 UTC	2025-01-17 15:56:12.648739 UTC

Conclusion

This movie ticket booking system project effectively illustrates real-time logging, data visualization, and a multi-threaded client-server architecture. With the ability to generate an interactive HTML report that offers valuable insights into user behavior, the system replicates a real-world movie ticket buying platform. The system's general functioning is improved and a layer of complexity is added through the use of concurrent programming and data visualization techniques.