MOVIES KINGDOM

INDEX

IUM-TWEB PROJECT REPORT

Report by Cacciatore Stella and Faraca Niccolò.

INTRODUCTION

This report provides an honest perspective on our web project's development, highlighting both the valuable lessons and significant challenges encountered. We navigated diverse technologies, including an Express.js central server, an Express microservice, and a Java Spring Boot microservice, focusing on performance and scalability. This practical experience has been exceptionally rewarding, allowing us to build a tangible product usable by everyday users.

TASK 1: FRONT-END DEVELOPMENT

SOLUTION

Our design is based on **Handlebars** pages, which includes its own partials, **Bootstrap** elements, **CSS** styling, and dedicated **JavaScript** files for handling page-specific functionality. We chose not to overlook the importance of design, because a clean and intuitive interface greatly improves navigation and user experience, additionally having a stable and well-structured front-end layout allowed us to develop the back-end and database queries in an organized manner.

ISSUES

Initially, the main challenge was understanding how to use Handlebars to create dynamic pages. With the help of some lessons and explanations provided by ChatGPT, we managed to understand how to split our pages into components and how to pass parameters effectively.

REQUIREMENTS

- The layout, pages, and partials are developed in HTML and Handlebars.
- Styling is handled through a combination of a custom CSS file and Bootstrap classes.
- The user can manage page contents thanks to Javascript.
- Each page is divided into multiple sections that allow users to query the central server for specific data, which is then dynamically rendered and displayed on the page.

LIMITATIONS

Web pages could be improved by adopting screen size responsive techniques to make it viable through diverse devices.

TASK 2: MAIN SERVER DEVELOPMENT

SOLUTION

Our main server, implemented using **Express**, acts as the primary entry point for all clients requests.

When a client sends a request (e.g., "open a certain movie page" or "search for a movie"), the Express server receives it, identifies the necessary data, and then forwards the request to the appropriate backend API servers using Axios. This approach offers several significant advantages because it allows the main server to handle a large number of users and not overload with database access operations or heavy calculations.

ISSUES

Our main issue developing the main server has been around parameters coming from the clients, checking their validity, so that the right information could be used to retrieve data from backend servers.

REQUIREMENTS

Our central server design directly addresses and complies with all specified requirements:

- All client requests are handled by our Express.js central server.
- All data is requested to backhand servers using Http requests via **Asynchronous Axios** calls.
- The Express performs **no complex business logic** or heavy data processing, ensuring its speed and ability to handle a high volume of concurrent requests.

LIMITATIONS

The high number of data retrieved from multiple services would be a scalability problem if more features and multiple backend servers would be introduced.

TASK 3: MONGODB SERVER DEVELOPMENT

SOLUTION

This service is specifically responsible for managing and serving highly dynamic data, namely **reviews and releases datasets**. The server handles API endpoints to requests of reviews by movie title *(/api/reviews/getReviews*), "best movies ever" based on weighted review scores (*/api/reviews/bestMoviesEver*), and fetch latest theatrical movie releases (*/api/releases/getLatestMovies*).

The choice to implement these data domains with MongoDB was driven by their dynamic nature, mainly because the releases database holds every release from each country, usually one for each type (theatrical, digital, etc..) and reviews have a pretty elevated change rate.

ISSUES

Our main issue with the development of this task was creating queries to fetch “best movies of all time”. This was mainly due to the parallel pipelining that we had to use in order to retrieve scores and all calculations to produce a weighted score, so that a more accurate and realistic result would be obtained.

REQUIREMENTS

- It is an Express server.
- The server has API endpoints to serve dynamic data.
- Mongoose is used to create schema and models for MongoDB collections.

LIMITATIONS

A more structured approach on error handling and data retrieval errors could be adopted.

TASK 4: SPRINGBOOT DEVELOPMENT

SOLUTION

Our **Java Spring Boot server** is designed to handle the main, more *static*, movie data retrieval from PostgreSQL database. Clients can request data based on 4 main ways:

- **Single Movie General Data**
- **Summarized movies data by providing: titles** (*best movies of all time, latest movies*), **no parameters** (*top-rated movies, most awarded movies*), **movie id** (*similarity to another movie*)
- **Summarized movies based on category parameters:** *by genre, by theme, by studio, by name, by duration (tv series), by actor, by director*
- **Summarized movies based on category and filtering parameters:** *by year, by language, by genre*
- **Summarized movies based on category and sorting parameters:** *by most popular (default), by alphabetical order, by year (newest,older)*

Our solution also makes use of **5 Data Transfer Objects**, which helped a lot in responding with the correct data. We also used the *Page* class to handle pagination when displaying movies, allowing us to not send all the movies which respected certain parameters in the requests.

ISSUES

Our main limitation with this task was creating the right queries in SQL and to find the right keywords for JPQL methods, especially for requests with several parameters.

REQUIREMENTS

- This service is explicitly designed in Java Spring Boot, fulfilling the requirement.
- The Spring Boot server exposes a comprehensive **REST API** collection for central server requests.
- The detailed filtering and sorting capabilities provide extensive database data retrieval.

LIMITATIONS

A lack of movie id in Oscars table leads to possible wrong data aggregation and timing retrieval could be improved.

TASK 5: DATA ANALYSIS & VISUALIZATION

SOLUTION

Our data analysis solution comprises one cleaning Jupyter Notebook for each csv files and three more for data analysis and visualization, the latter containing plots to tell a specific story:

1. **data_visualization.ipynb**: It focuses on analyzing and visualizing the broad movie industry evolution in time and if audience and Oscar academy judgment align.
2. **the_power_of_actresses.ipynb**: This notebook delves into the representation and impact of actresses within the Academy Awards.
3. **marvel_studios.ipynb**: This project explores the evolution, influence, and global presence of Marvel Studios.

ISSUES

In this task we were faced with two main big issues: “writer’s block” and limitations with data. At the beginning, we put a lot of effort and time into finding a story which would reach our expectations, but none of the ideas felt complete, and every time we came up with one, there was not enough data to fulfill our analysis.

REQUIREMENTS

- Jupyter notebooks were used.
- Data cleaning was performed on all csv files.
- Libraries like pandas, numpy, plotly, matplotlib and seaborn were used.
- All three notebooks use visualization libraries to produce a variety of chart types..
- Our data visualization narrates **three different stories** of evolution in time.

LIMITATIONS

Our cleaned csv files could be cleaned in a better way, to ensure a more fluid and cohesive analysis/querying. With the data being this big, finding every flaw and error is hard.

CONCLUSIONS

We believe that we managed to respect most of the requirements for this project and, being the first time we have developed a project this big and complex, we are highly satisfied with it. Our website almost feels like a real one, and it has been fun to learn and create something that’s part of everyday life and to understand what we’ve been using all this time.

DIVISION OF WORK

Although we are trying to separate the work, as commit log history shows, we have worked equally in every section and functionality of the project.

	STELLA CACCIATORE	NICCOLÒ FARACA
FRONTEND DEVELOPMENT	mainly focused on partials, worked on chat.js, movieFilters.js	mainly focused on pages and styling, some functions in js
MAIN SERVER	***Routes***: /best-of-all-time, /tvseries, /genres ***Others***: Swagger documentation	***Routes***: /directors-page, /actors-page, /search, /themes-page, /studios ***Others***: some functions in index.js,, skeleton of project
MONGODB SERVER	Retrieving best movies of all time and latest releases + skeleton of project and connection to mongo	Retrieving reviews for a specific movies
SPRING BOOT SERVER	***Packages***: The classes in the packages were developed by both. ***DTOs***: MovieCategoryDTO, HomepageDTO ***Others***: *SwaggerDocumentation*	***Packages***: The classes in the packages were developed by both. ***DTOs***: ListWrapperDTO, DetailedMovieDTO, CarouselDTO ***Others***: *UtilityClassManager, ProjectExceptionHandler*
PYTHON DATA VISUALIZATION	***Main cleaning***: actors, genres, countries, releases, oscars, studios ***Visualization***: the_power_of_actresses	***Main cleaning***: crew, countries, languages, themes, reviews. ***Visualization***: marvel_studios

EXTRA INFORMATION

No extra information is needed.

BIBLIOGRAPHY

ChatGPT and Gemini provided us with quick, clear guidance and suggestions that saved us time and deepened our understanding when we were in trouble. Additionally, it helped us with the construction of the styling and skeleton of the website.