TWEB Anime Explorer – Project Report

Begoña Arana Méndez de Vigo student id number: 1202141

Contenido

Web Technologies – University of Turin

1. Introduction

This report describes the design, development, and evaluation of the *TWEB Anime Explorer* project, carried out for the *Web Technologies* course at the University of Turin.
The main objective of the assignment is to apply the architectural principles, technologies, and methodologies presented during the course, with particular attention to multi-tier web architectures, server-to-server communication, and the integration of heterogeneous data sources.

The project is based on a dataset containing information about anime titles, including their release year and associated genres. The developed system allows users to explore this data through a web interface that supports search, filtering, sorting, and pagination.

The project was developed incrementally, starting from a basic server architecture and progressively adding functionality, data management, and presentation layers.

2. Architecture Overview

The system follows a **multi-tier client–server architecture**, clearly separating presentation, application logic, and data management.

The client is a standard web browser, which interacts with a **Main Server** implemented in Node.js using Express. The Main Server is responsible for handling HTTP requests from the user and rendering HTML pages using server-side templates.

The Main Server does not directly access any database. Instead, it communicates via RESTful HTTP APIs with two backend services:

- a **Data Server** implemented in Express and connected to a MongoDB database;
- a **Spring Boot Server** implemented in Java and connected to a PostgreSQL database.

Each backend server has a specific responsibility, following the principle of **separation of concerns**. This architectural choice improves modularity, maintainability, and clarity of the system design.

3. Data Server – Express and MongoDB

3.1 Design Choices and Motivation

The Data Server is implemented using Express and manages dynamic and semi-structured data stored in MongoDB. MongoDB was selected due to its flexible document-based schema, which is well suited for handling anime data where attributes such as genres may vary in number and structure.

The server exposes a REST API that provides access to the dataset in JSON format. The main endpoint, GET /animes, supports:

- search by title (case-insensitive),
- filtering by genre,
- sorting by title or year,

- pagination through page and limit parameters.

Additional endpoints such as GET /genres and GET /health support the user interface and provide a simple way to monitor server availability.

3.2 Issues Encountered

One of the main challenges was designing query parameters that could be combined flexibly without producing inconsistent results. In particular, pagination required careful handling to ensure that page counts and result sizes remained coherent when filters were applied.

These issues were addressed by explicitly validating parameters and by structuring MongoDB queries in a clear and incremental way.

3.3 Requirements Compliance and Limitations

This component satisfies the assignment requirement of using **Express together with MongoDB** to manage a dataset.
A limitation of the current implementation is the absence of caching or indexing strategies, which could improve performance for very large datasets.

4. Spring Boot Server – Java and PostgreSQL

4.1 Design Choices and Motivation

The Spring Boot server manages more **structured and relational data** using PostgreSQL. Spring Boot was chosen for its robustness, strong ecosystem, and native support for REST APIs and database access through Spring Data JPA.

The application defines a simple domain model mapped to a relational schema. The use of JPA annotations allows the database structure to be defined directly in Java code, reducing boilerplate and improving readability.

The server exposes REST endpoints such as:

- GET /health, to verify that the application is running correctly;
- additional API endpoints that demonstrate database connectivity and data access.

Demo data is automatically inserted at startup to simplify testing and demonstration.

4.2 Issues Encountered

Initial difficulties were encountered during database configuration, particularly in aligning application properties with PostgreSQL credentials and driver settings. These problems were resolved through incremental testing and careful inspection of error messages.

4.3 Requirements Compliance and Limitations

This component fulfills the requirement of implementing **at least one server using Java Spring Boot and PostgreSQL**.
The current implementation focuses on read-only access and does not include advanced relational queries or transactional operations.

5. Main Server – Express and Handlebars

5.1 Design Choices and Motivation

The Main Server acts as the **central coordination point** of the system. It is implemented using Express and uses Handlebars as a templating engine for server-side rendering.

When a user submits a search request, the Main Server:

1. receives the HTTP request from the browser;
2. forwards the request parameters to the appropriate backend server using Axios;
3. processes the JSON responses;
4. renders the final HTML page sent back to the client.

This approach allows the user interface to remain simple while keeping all data access logic on the server side.

5.2 Issues Encountered

Managing multiple asynchronous requests to different backend services required careful error handling. This was addressed by implementing fallback mechanisms that preserve basic functionality even when one of the services is temporarily unavailable.

5.3 Requirements Compliance and Limitations

The Main Server satisfies the requirement of a **central Express server** responsible for integrating multiple backend services.
The system is not designed for high concurrency and does not include load balancing mechanisms.

6. Use of Generative AI Tools

Generative AI tools were used during development as a **support resource**, not as a substitute for implementation. They were mainly used to:

- clarify theoretical concepts discussed during lectures,
- understand and interpret error messages,
- assist in resolving configuration and environment-related issues,
- improve the clarity and structure of documentation.

All architectural decisions, code implementation, testing, and validation were performed independently by the author.

7. Division of Work

The project was developed individually. All components, including backend servers, frontend interface, database configuration, testing, and documentation, were implemented by the author.

8. Execution Instructions

To run the project locally, both MongoDB and PostgreSQL must be running.
The Express servers are started using npm, while the Spring Boot server is started using Maven.

Conclusions

The *TWEB Anime Explorer* project successfully demonstrates the application of the main concepts covered in the Web Technologies course. By combining multiple servers, heterogeneous databases, and REST-based communication, the project reflects a realistic modern web architecture.

The project highlights the importance of modular design, separation of concerns, and clear communication between components. Despite its simplicity, the system provides a solid foundation that could be extended with additional features and optimizations