HCI REPORT 2025/26

Filippo Guanti & Denis Budau

INDEX TABLE

1 Introduction
2 Technical Task 1 – Main Express Server configuration
- **2.1** Solution
- **2.2** Issues
- **2.3** Requirements
- **2.4** Limitations

3 Technical Task 2 – Second Express Server configuration
- **3.1** Solution
- **3.2** Issues
- **3.3** Requirements
- **3.4** Limitations

4 Technical Task 3 – Spring Boot Server configuration
- **4.1** Solution
- **4.2** Issues
- **4.3** Requirements
- **4.4** Limitations

5 Technical Task 4 – Website creation
- **5.1** Solution
- **5.2** Issues
- **5.3** Requirements
- **5.4** Limitations

6 Conclusions
7 Division of Work
8 Extra Information
9 Bibliography

1. Introduction

The aim of this project is to create a web-based system that allows fans and journalists to access and analyze a large dataset of anime.
The system provides structured information through different servers and a simple web interface. The focus of the project is on data access, communication between servers, and correct use of web technologies studied during the course.

2. Technical Task 1 – Main Express Server configuration

2.1 Solution

The logic of the main server is straightforward and designed to ensure fast responses: the browser sends requests, and the server handles two types of routes. API routes provide data in JSON format, while Page routes use res.render to return HTML pages, allowing the server to easily

distinguish between data requests and page requests while maintaining high performance. The API routes interface with two back-end servers: an Express satellite server with MongoDB and a Java Spring Boot server with PostgreSQL. The main API routes provide data on Anime, Character, Person, and User: the first three communicate with the Express server, while User data is retrieved from the Java server. There are also other routes that are currently unused and considered redundant. The architecture follows the MVC pattern, with controllers managing the asynchronous functions of the routes.

2.2 Issues

One of the main difficulties was understanding how to manage asynchronous requests correctly.
Another issue was handling errors when a server was not responding or returned an error.
These problems were solved by using try/catch blocks and by returning proper HTTP error codes to the client.

2.3 Requirements

The server correctly receives data from the client and sends it to the MongoDB server.
It is able to store and retrieve data from MongoDB through the NoSQL satellite server.
The communication between servers uses JSON and HTTP requests.
The server uses non-blocking asynchronous calls to handle multiple requests at the same time.
HTTP status codes are used to indicate whether a request was successful or not.

2.4 Limitations

The Main Server only forwards requests and does not perform complex operations.
Error handling is basic and could be improved in the future.
Security features such as user authentication were not implemented because they were not required for this assignment. Anche le funzioni

3. Technical Task 2 – Second Express Server configuration

3.1 Solution

The second Express server was implemented to manage dynamic data using MongoDB.
This server is responsible for storing and retrieving information such as ratings and user profiles.
MongoDB was chosen because it allows flexible data structures and is suitable for data that can change over time.
The database is called “anime” and contains the ratings and user profile collections as required. To interact with the databases, we used controllers that, through asynchronous functions, communicate with the datasets structured according to the schemas defined in the various models.
The server exposes REST endpoints that communicate using HTTP and JSON.

3.2 Issues

One of the main difficulties was understanding how to correctly model data in MongoDB without using a relational structure, as we’d never deal with it.
Another issue was managing the connection between the Express server and the MongoDB database.
These problems were solved by using Mongoose schemas and by testing the endpoints with simple requests.

3.3 Requirements

The solution correctly implements MongoDB and follows the required database and collection names.
The file system is organized by separating routes, controllers and models.
Data is stored and retrieved correctly from the MongoDB collections.
The server communicates correctly with the main Node.js server using JSON and HTTP requests.

3.4 Limitations

The MongoDB structure is simple and focuses only on the required data. Setting MongoDB up and uploading the largest dataset took a while.

4. Technical Task 3 – Spring Boot Server configuration

4.1 Solution

The Spring Boot server was implemented as a satellite server to manage static anime data stored in PostgreSQL. In Java Spring, we decided to manage the **Character**, **Anime**, and **Person** datasets being mostly static separately from the **User** data, which is dynamic and subject to frequent updates.
It exposes REST endpoints that allow the client and the main server to retrieve anime, person and character information.
The server uses Spring Boot together with JPA to map Java entities to database tables.
Data is loaded at startup using a CSV importer implemented with CommandLineRunner.
This approach allows importing and cleaning the dataset directly inside the application without using external database tools.

4.2 Issues

One of the main challenges at the beginning was importing CSV files, which we solved using dedicated scripts for this purpose. Apart from that, the Java part was very straightforward, as the repositories extend JPA and provided us with many pre-implemented methods, making the work quite easy.

4.3 Requirements

The Spring Boot server correctly receives requests and communicates with the client using HTTP and JSON.
It connects correctly to PostgreSQL and stores and retrieves data using JPA repositories.
HTTP status codes are used to indicate successful or failed requests.
The project follows the required structure, separating entities, repositories, services and controllers.
JPA is used correctly to manage persistence and database access.

4.4 Limitations

The CSV importers are specific to each dataset and not fully generic.
Error handling is basic and could be improved.
The database schema is simple and focuses only on the required fields.
Overall, the solution satisfies the requirements and produces correct results.

5. Technical Task 4 – Website creation

5.1 Solution
The website was developed using Express and Handlebars after the backend APIs were partially implemented and tested using Swagger and JSON responses.
This allowed us to verify that the servers were working correctly before integrating them into the web interface. In the views section, the structure is organized with a fixed navbar and footer, while the body changes depending on the page. There is a single search page, used to look up anime, characters, and people, and a top 50 page, which displays the top 50 anime, characters, people, and users according to rank, favorites, and completed anime, respectively. We could have used a single page for everything, but having all content in one HTML and JavaScript file would have been somewhat confusing, so we decided to split them into two separate pages. Both pages are populated by their respective JavaScript files, which make API calls using Axios to retrieve the data.

5.2 Issues
One of the main issues was understanding how to update the page content dynamically without reloading the entire page.
Another difficulty was correctly handling asynchronous requests and displaying the results in the correct section of the page.
Some layout adjustments were also needed to make the interface clearer and more readable.

5.3 Requirements
The website communicates correctly with the Node.js main server using HTTP requests.
All data exchange between the client and the server uses JSON.
Asynchronous JavaScript communication is used systematically through Axios.
The interface uses clear elements and views to avoid confusion for the user.
The results shown on the page correctly reflect the data returned by the backend servers.

5.4 Limitations
The graphical design of the website is simple and focused mainly on functionality.
The interface could be improved with additional styling and animations.
Client-side error handling is basic and could be extended to provide more detailed feedback to the user.

6. Conclusions
Overall, the project meets the objectives of the assignment and shows a correct use of the technologies studied in the course.
The system is well structured and the different components communicate correctly with each other.
The work helped to better understand distributed architectures, asynchronous communication, and REST APIs.
Even if some aspects could be improved, the final result is functional, clear, and consistent with the project requirements.

7. Division of Work
The project was developed by both group members, Filippo and Denis, with the goal of dividing the work as evenly as possible.
Both members worked on the different servers and discussed design choices together before implementation.
In practice, the workload sometimes differed depending on the task and its complexity, with one member taking a more active role in specific parts.

The website was developed by both members, starting from a basic Express and Handlebars structure and later improving and refactoring the code together.
The report was written collaboratively, with Filippo taking care of the English writing and final formatting.
Both Filippo and Denis contributed to testing, debugging, and integration between the different parts of the system.

8. Extra Information

Before running the project, the lecturer needs to run the provided Jupyter notebook to clean the datasets.
The cleaned datasets are then imported into MongoDB using MongoDB Compass, under the database name *anime*, with the collections *favs*, *ratings*, *recommendations*, and *user_profile*.
For PostgreSQL, the Spring Boot server automatically imports the cleaned CSV files at startup using three dedicated importers.
The same import process could also be performed manually using pgAdmin.
To run the Node.js servers, dependencies must be installed using the command npm install.
Java 17 must be installed and selected as the project SDK, since newer Java versions may cause compatibility issues with Spring Boot or Gradle.

9. Bibliography

No external books were used for this assignment.
The work was mainly based on the provided datasets and course material.
Course Lecture Slides – TWEB: Anime Data Aggregator, Academic Year 2025.
Professor F. Ciravegna, University of Torino – Lecture Notes.
W3Schools was used as a reference for JavaScript, Express, and basic web concepts.
Large Language Models were also used to support understanding and implementation, mainly ChatGPT 5.2 and Google Gemini 3 Pro.