Progetto TWEB

Francesco Bassignana

January 2024

Contents

1 Introduction

This is the report of the IUM TWEB assignment 2023/2024

1.1 What should you expect from this report?

In this report I'll try to resume all the motivations and the choices made behind the scenes by me in order to realize the solution

1.2 Language Premise

Despite my very bad English level, the project is fully developed in English (distraction-typo excluded), so I decided to made also this report in this language for coherence. Only commits are made in italian language because in order to write decent english i need to have minumum brain energy, and sometimes commits are made at the end of tiring session of coding.

2 Requirements

2.1 Functional requirements

1. Web Interface: HTML+Javascript+CSS page for data querying and exploration.
2. Servers:
   - IUM-TWEB (6 and 12 credits): Central server in Express.
   - IUM (6 credits) and IUM-TWEB (12 credits): Python Flask, NodeJS/Express, or Java SpringBoot.
3. Data Storage: Dynamic data in MongoDB; static data in PostGres (IUM-TWEB) or MongoDB/SQL database (IUM).
4. Chat System (IUM-TWEB): socket.io based chat for fans and pundits.
5. Data Analytics: Jupyter Notebooks with Web interface for querying data subsets.

2.2 Non-functional requirements

1. Performance: Fast central server for handling thousands of users.
2. Scalability and Extensibility: Adaptable for increasing loads and future requirements.
3. Usability: User-friendly interface for fans and pundits.
4. Reliability and Availability: Continuous system access.

3 DB data splitting

3.1 Objective

To outline the strategic allocation of data across MongoDB and PostgreSQL for a Football Statistics application, leveraging the strengths of each database management system.

3.2 Database Splitting Strategy

3.2.1 PostgreSQL (RDBMS) for Stable Structured Data:

- **Tables Stored:** Clubs, Competitions.
- **Reasoning:** Ideal for stable, structured data with consistent schemas, ensuring data integrity and efficient complex queries.

3.2.2 MongoDB (NoSQL) for Dynamic, Changing Data:

- **Collections Stored:** Apparences, Game Events, Game Lineups, Player Valuations, Games and Players.
- **Rationale:** Accommodates varying structures and frequent schema changes, ideal for dynamic data.

3.2.3 Decision Justification Based on Change Rate:

- **Data Volatility:** PostgreSQL for stable data with lower change rates.
- **Schema Evolution:** MongoDB for dynamic data with higher change rates and evolving schemas.
- **Adaptability to Change:** MongoDB's document-based storage for evolving schema adaptability.

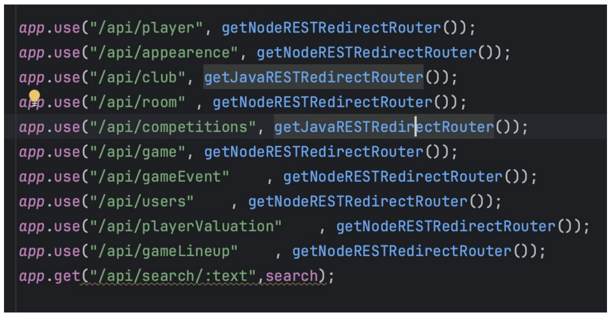
3.3 Conclusion

This strategy optimizes data storage, retrieval, and manipulation, ensuring both flexibility and relational integrity in the Football Statistics application. Technologies

- **Main Server:** NodeJS (JavaScript)
- **Java Server:** Java Spring Boot Server (Java)
- **Express Server:** NodeJS (JavaScript)
- **Web App:** React (TypeScript)

4 Main server

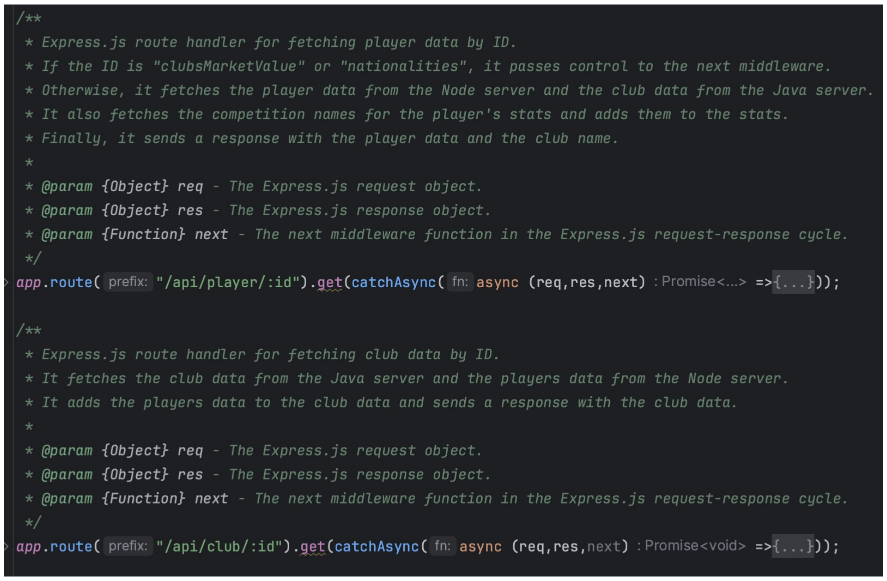
The principal purpose of the main server is to hold the chat system and to act as an intermediate between frontend and the servers (Java springboot and Express JS). It's a very light server because the only things its middlewares are doing is to redirect incoming requests to other servers like that:



```
app.use("/api/player", getNodeRESTRedirectRouter());
app.use("/api/appearence", getNodeRESTRedirectRouter());
app.use("/api/club", getJavaRESTRedirectRouter());
app.use("/api/room" , getNodeRESTRedirectRouter());
app.use("/api/competitions", getJavaRESTRedirectRouter());
app.use("/api/game", getNodeRESTRedirectRouter());
app.use("/api/gameEvent"    , getNodeRESTRedirectRouter());
app.use("/api/users"    , getNodeRESTRedirectRouter());
app.use("/api/playerValuation"    , getNodeRESTRedirectRouter());
app.use("/api/gameLineup"    , getNodeRESTRedirectRouter());
app.get("/api/search/:text",search);
```

I suggest to look at the code provided in option.js file in order to understand what's going on.

I also provided a good code docs. There are only some special routes that call specific api from other 2 servers non-automatically for more specific purposes



```
/**
 * Express.js route handler for fetching player data by ID.
 * If the ID is "clubsMarketValue" or "nationalities", it passes control to the next middleware.
 * Otherwise, it fetches the player data from the Node server and the club data from the Java server.
 * It also fetches the competition names for the player's stats and adds them to the stats.
 * Finally, it sends a response with the player data and the club name.
 *
 * @param {Object} req - The Express.js request object.
 * @param {Object} res - The Express.js response object.
 * @param {Function} next - The next middleware function in the Express.js request-response cycle.
 */
app.route( prefix: "/api/player/:id").get(catchAsync( fn: async (req,res,next) :Promise<...> =>{...}));

/**
 * Express.js route handler for fetching club data by ID.
 * It fetches the club data from the Java server and the players data from the Node server.
 * It adds the players data to the club data and sends a response with the club data.
 *
 * @param {Object} req - The Express.js request object.
 * @param {Object} res - The Express.js response object.
 * @param {Function} next - The next middleware function in the Express.js request-response cycle.
 */
app.route( prefix: "/api/club/:id").get(catchAsync( fn: async (req,res,next) :Promise<void> =>{...}));
```

This server also hold api-docs made with SwaggerUI

5 Express Server

This server hold the main operations on players, player valuations, game events, game, game lineups, appearances, room and user authentication. For each of theese entities i made a mongoose **model**, a js **router** and a **controller**. Following **MRC** Architecture:

5.1 Models

The models hold the structure of a single document that will be stored in Mongodb database + main operations between saving and retreiving operations

5.2 Routes

In an MRC architecture, routes are the main components responsible for handling incoming HTTP requests and directing them to the appropriate controller methods. Each route defines an endpoint and specifies which controller method should be executed for that endpoint. Routes are responsible for handling the routing logic and parsing request data

5.3 Controllers

Controllers contain the logic that handles specific actions related to the application's business logic. They receive requests from the routes, interact with models (if needed), process data, and send responses back to the client. In the MRC architecture, controllers are responsible for most of the application's functionality. In my case I gave as buisness-logic-work as I could to mongoose middle-ware to maintain logic separations of concepts and more efficient work distribution

6 Java server

In java server I decided to hold only clubs and competitions data. This server is made by JPA architecture and respond to basic retrieving data requests for theese two entities. I will not tell so much about this server because it's functioning is very standard

7 React app

I choose to use react as the frontend-part of my tech stack because many reasons. I think doing frontend with react-like framework is much developer-friendly (unlike Vanilla Javascript and HTML) for a lot of reasons:

- Component-Based Architecture: React promotes a component-based architecture, where you break down your UI into reusable and self-contained components. This modular approach makes it easier to manage complex UIs and encourages code reusability.
- Virtual DOM: React uses a virtual DOM to optimize DOM updates. Instead of directly manipulating the actual DOM, React creates a virtual representation of it. When changes occur, React calculates the minimal number of updates needed and applies them efficiently. This results in faster rendering and better performance.
- Declarative Syntax: React's declarative syntax makes it easier to describe what your UI should look like at any given time. You define the desired state, and React takes care of updating the DOM to match that state. This can lead to more readable and maintainable code compared to imperative DOM manipulation in vanilla JavaScript.
- Ecosystem and Community: React has a vibrant and active community, which means you can find a wealth of open-source libraries, tools, and resources to help with various aspects of frontend development. This ecosystem can save you time and effort in building and maintaining your application.
- State Management: React provides a straightforward way to manage the state of your application through its built-in state management system. For more complex applications, you can integrate libraries like Redux or Mobx for advanced state management.
- Component Reusability: React encourages the creation of reusable components, which can be shared across different parts of your application or even between projects. This can lead to a more efficient development process and maintainable codebase.
- React Hooks: React introduced hooks, which allow you to use state and other React features in functional components. This has simplified component logic and made it easier to work with state in functional programming style.
- Strong Community Support: With a large and active community, you can find solutions to common problems, access extensive documentation, and benefit from frequent updates and improvements to React.

7.1 UI Framework/libraries

For help me to write css code I used TailwindCSS and NextUI.

- TailwindCSS Helped me a lot on focusing on components developing and the application logic
- NextUI Helped me giving its beautiful fully customizable components with nice default UI, NextUI is based on (and also customizable using) Tailwind

8 ”External” Helpers

8.1 AI

Before I start to talk how I used AI as a helper to develop my project I would like to do a premise. My ”slogan” that I use when I'm trying to think wheter should I use AI or not is: ”I have to think faster than AI, AI have to write faster than me”. All the solutions you will see in the project are the result of my intuitions and reasoning for problem-solving. I basically used AI for doing boring stuffs like write some code for which I had already realized the solution's shape in my mind, write comments, basic components or functions and so on.

8.2 Previous ”home-made” functions

If take a look to the first commits you might see that functions like AppError, APIFeatures, CatchAsync, ecc... simply appear to my code. This is dued to my preparation to the project before the assignment was given (in fact it is general-purpose code).
The reason why I decided to put ”External” between quotes is because

- Despite using AI helped me a lot, I think in order to using it efficiently we have to be able to make very specific questions to it and, despite thinking otherwise, it’s not very common to have this ability.
- The functions which I just talked about are made by me and I’m of course ready to explain their behaviour