TWEB
Year 2023/2024

# REPORT

University of Turin
Dario Bonfiglio **984337**, Fornaroli Inyan **966505**,
Gullì Alessandra **962507**
dario.bonfiglio@edu.unito.it - inyan.fornaroli@edu.unito.it -
alessandra.gulli@edu.unito.it

# Contents

## 1 Introduction

Our goal was to develop a project related to the world of football, providing both fans and industry experts with comprehensive information about the sport, ranging from statistics to match results.
We divided the data between two databases to optimize performance:

- **MongoDB (dynamic)**: This database has a higher access speed, making it better for constantly updating data. Here we maintain: appearances, clubGames, gameEvents, gameLineups, games,

- **PostgreSQL (static)**: This database contains tables that are modified less frequently, including: players, clubs, competitions, playerValuations

As per the assignment requirements, we developed a Main Server using JavaScript with Express.js to interact with the other two servers. The first server functions as the main server, connected to MongoDB, while the second server, developed in Java using the SpringBoot framework, is connected to the PostgreSQL database.

# 2 TASK: Login and SignUp

## 2.1 Solution

The solution involves creating a login and sign-up system using a combination of HTML for the user interface, JavaScript for form validation and interaction, and backend logic implemented with Node.js and MongoDB. The system provides two main functionalities:

1. **Sign Up**: Users can register by providing a username, email, and password. The password is securely hashed using bcrypt before being stored in the MongoDB database. Upon successful registration, a JWT token is generated and returned to the user for session management.

2. **Login**: Users can log in using their email and password. The entered password is compared with the hashed password stored in the database using bcrypt. If the credentials are valid, a JWT token is generated and returned to the user.

## 2.2 Issues

Several potential issues are identified in this implementation:

- **Validation of User Input**: While there is client-side validation, the system could be vulnerable to SQL injection or XSS attacks if proper server-side validation and sanitization are not implemented.

- **Password Security**: Passwords are hashed with bcrypt, but there is no mention of salting, which could enhance security against brute-force attacks.

- **Error Handling**: While error messages are provided in the frontend, the server could be made more robust by differentiating between specific types of errors, such as user not found versus incorrect password.

- **Session Management**: JWT tokens are stored in sessionStorage on the client-side, which might be vulnerable to XSS attacks if not properly protected.

## 2.3 Requirements

- **Frontend**:
  - HTML, CSS, and JavaScript (ES6) are used to build the login and signup forms. The UI is designed to be responsive and accessible, with error messages displayed for incorrect input.
  - The login and signup forms include validation for email format and password length.

- - **Backend**:
  - A Node.js server using Express handles API requests for login and signup.
  - MongoDB is used to store user data, including hashed passwords.
  - JWT (JSON Web Token) is used for session management, allowing the user to stay authenticated after login.

- **Libraries and Frameworks**:
  - ‘bcryptjs‘ is used for password hashing and verification.
  - ‘jsonwebtoken‘ is used for generating and verifying JWT tokens.
  - ‘mongoose‘ is used for database operations with MongoDB.

## 2.4 Limitations

- **Security Concerns**: While bcrypt is used for password hashing, other security measures like account lockout after repeated failed login attempts are not present. This could expose the system to brute-force attacks.
- **Client-Side Token Storage**: Storing JWT tokens in ‘sessionStorage‘ could be insecure. If an attacker manages to inject malicious code into the page, they could potentially access the token and impersonate the user.
- **Form Validation**: Although the frontend provides form validation, there could be further improvements, such as adding more robust server-side validation and checking for existing email addresses during registration to prevent duplicates.
- **Scalability**: As the user base grows, using JWT tokens might become inefficient if token revocation or expiration is not handled properly.

# 3 TASK: Chat System

## 3.1 Solution

The chat system is designed to provide real-time communication between users in a web application. It uses a combination of HTML, JavaScript, and socket.io for seamless interaction between the client and server. The key components are:

1. **Frontend (HTML + JavaScript)**:
   - The chat interface consists of input fields for username and messages, a button to join the chat, and a container to display messages.
   - The user can join a general chat room or a specific team's chat room. The frontend handles user input, joining a chat room, sending messages, and displaying them in real-time.
2. **Backend (Socket.io)**:
   - The backend is built using socket.io, which enables real-time, bi-directional communication between the server and clients.
   - When a user joins the chat, a unique user ID is generated, and they are connected to a specific chat room.
   - The server listens for chat messages and broadcasts them to all users in the same room.
3. **Message Handling**:
   - Messages are sent through socket.io events. When a user sends a message, it is broadcasted to all users in the room. The system also handles user disconnection and room switching, ensuring that messages are displayed to the correct users.

## 3.2 Issues

Several issues and potential improvements have been identified in the current implementation:

- **Security**:
  - User authentication is not enforced before joining a chat, which might allow unauthorized users to participate in conversations. Implementing authentication with JWT (as seen in the login system) would enhance security.
  - Messages are broadcasted without any sanitization. This leaves the system vulnerable to cross-site scripting (XSS) attacks if malicious users send harmful scripts as messages.
- **Scalability**:
  - The current system supports rooms and message broadcasting, but it does not handle scaling well for larger numbers of users or rooms. Implementing namespaces or rooms in socket.io would help manage multiple users more efficiently.

- **Error Handling**:
  - There is limited error handling in both the frontend and backend. For instance, if a user disconnects unexpectedly or the server crashes, the frontend does not provide meaningful feedback to the user.

## 3.3 Requirements

The chat system has the following key requirements:

- **Frontend**:
  - HTML, CSS, and JavaScript are used to create a responsive user interface. The interface includes input fields for entering a username, sending messages, and selecting a chat room.
  - JavaScript handles sending messages and interacting with the server in real-time using socket.io.
- **Backend**:
  - A Node.js server running socket.io handles real-time communication. It manages user connections, room creation, message broadcasting, and disconnections.
  - Each user is assigned a unique ID upon joining, and their messages are broadcasted to the appropriate room based on their current chat session.
- **Libraries and Frameworks**:
  - 'socket.io' is used for real-time communication.
  - 'uuid' is used to generate unique user IDs.
  - 'axios' is used for HTTP requests to interact with other parts of the system (if needed).

## 3.4 Limitations

- **No Authentication**:
  - The system does not include any form of user authentication, which makes it difficult to prevent unauthorized access or provide personalized chat experiences.
- **Message Storage**:
  - Messages are not stored persistently. Once users leave the chat or refresh the page, the message history is lost. Implementing a database to store messages would enhance the user experience by providing chat history.
- **Room Management**:

  - The current room system is basic, and users can join rooms without any restrictions. There is no distinction between private and public rooms, and room management features (e.g., room creation, deletion) are missing.

- **Scalability**:
  - The system is designed for small to medium-sized chat environments. It does not yet support scaling to large numbers of concurrent users or rooms, and further improvements would be required for deployment in larger applications.

# 4 TASK: Searching Pages (Matches, Teams, Competitions, and Players)

## 4.1 Solution

For searching information within the site, we used:

- **Games**: where you can search using combinations of Seasons, Championships, and Dates.
- **Teams**: uses a search bar to find clubs by name or through an alphabetically ordered list.
- **Players**: similar to Teams, it uses a search bar to find a player by name or through an alphabetically ordered list.
- **Chat**: allows searching for a chat to interact with other people interested in the same topic.

## 4.2 Issues

It was challenging to solve the problem of quick data loading and the visualization of the returned data. Specifically, combining data from **PostgreSQL** with those from **MongoDB** for match searches was a key difficulty. Additionally, the player list visualization is not ideal as we decided to load it every time the section is accessed, making it somewhat slow.

## 4.3 Requirements

- **Efficient Database Queries**: Both **MongoDB** and **PostgreSQL** need to support efficient data retrieval to ensure quick loading of search results.
- **User-Friendly Interface**: The search bars and the alphabetical lists must be intuitive and easy to use for both teams and players.
- **Real-Time Data Handling**: For chat-related searches, real-time data streaming is required, with the ability to retrieve relevant conversations quickly.

## 4.4 Limitations

- **Loading Time**: Due to the decision to reload the player list each time the page is accessed, it introduces a noticeable delay in loading the content.
- **Complex Query Processing**: Handling searches that involve combining data from multiple databases (MongoDB and PostgreSQL) is complex and requires optimization.
- **Limited Caching**: There is no caching mechanism in place for frequently accessed data, which could improve performance and reduce server load.

# 5 TASK: Data Analysis

## 5.1 Solution

The data analysis task involves extracting, processing, and analyzing data related to appearances, club games, competitions, game events, and player valuations. The analysis is performed using Python in Jupyter notebooks, utilizing libraries such as **pandas**, **numpy**, **matplotlib**, and **scikit-learn** for data manipulation, visualization, and machine learning.

- **Appearances Analysis**: Analysis of player appearances, their performance in different games, and correlations with other factors such as team success or competition type.
- **Club Games**: Detailed insights into club performance across multiple games, including win/loss ratios, goal differences, and key moments in a season.
- **Competitions**: Analysis of various competitions, extracting key statistics such as top-performing teams, most competitive seasons, and comparison between different leagues.
- **Game Events**: Processing and analysis of in-game events like goals, fouls, and substitutions to understand key moments that affect game outcomes.
- **Player Valuations**: Valuation models based on player performance data, including factors such as goals, assists, and appearances, to estimate a player's market value.

## 5.2 Issues

Some key challenges faced during the data analysis task were:

- **Data Consistency**: Merging datasets from different sources (games, player appearances, valuations) presented issues in data consistency, requiring heavy preprocessing to handle missing or conflicting data.
- **Scalability**: The size of some datasets (e.g., match events) caused performance bottlenecks, requiring optimization techniques to speed up queries and data processing.
- **Missing Data**: Certain datasets lacked complete information, particularly in player valuations and match event details, which led to the need for imputation techniques.

## 5.3 Requirements

To successfully perform the data analysis, the following requirements were essential:

- **Python Libraries**: Extensive use of **pandas** for data manipulation, **matplotlib** for visualization, and **scikit-learn** for applying machine learning algorithms in player valuations.

- **Clean Datasets**: The datasets required cleaning, preprocessing, and normalization to handle missing values, inconsistencies, and scaling issues.
- **Computational Resources**: Handling large datasets required sufficient computational resources, especially for aggregating large-scale player or match data.

## 5.4 Limitations

The data analysis task had the following limitations:

- **Data Gaps**: Not all competitions or games had complete data, limiting the scope of the analysis. Some player performance metrics were estimated based on available data.
- **Performance Bottlenecks**: Due to the size of the datasets, some analyses were slow to execute, particularly those involving complex joins between multiple datasets.
- **Limited Machine Learning Models**: The player valuation models were limited to linear regression and simple algorithms due to the available data features. A more comprehensive model would require additional factors like team performance or market trends.

## 6 Conclusion

In this project, we successfully developed and implemented multiple systems, including a login and signup system, a real-time chat platform, and comprehensive search functionality for games, teams, competitions, and players. Each component was carefully designed to meet the project's requirements, leveraging modern technologies such as **Node.js**, **MongoDB**, **PostgreSQL**, and **Socket.io** to deliver efficient and scalable solutions.

The data analysis provided valuable insights into the performance of players, teams, and competitions by processing large datasets, although certain challenges such as data consistency, scalability, and incomplete datasets required careful consideration. Despite these challenges, we managed to extract meaningful results that can inform future developments, such as enhancing the valuation models for players or improving the efficiency of data queries.

While the overall systems achieved their goals, the project also highlighted several areas for potential improvement, particularly in terms of security, data handling, and performance optimization. With the current foundations in place, further enhancements—such as advanced machine learning models for player valuation or optimized query processing for large datasets—can be explored to extend the system's capabilities.

## 7 Division of Work

- **Bonfiglio Dario**: 75% Main Server, 75% Express, 20% Data Analysis, Swagger, Home Page, Players, Games, Teams.
- **Fornaroli Inyan**: 25% Main Server, 25% Express, 75% SpringBoot, JavaDoc, 50% Chat.
- **Gullì Alessandra**:80% Data Analysis, 25% SpringBoot, Game Page, Player Page, Team Page, 50% Chat.

## 8 Bibliography

- https://chatgpt.com: Used to help us with any code issues