Football Data Analysis System Report

Task 1: Data Storage Design

Solution: To store the football data efficiently, I created six tables in PostgreSQL for static data such as clubs, competitions, nations, player statistics, players, and users. This relational database is chosen for its robustness, support for complex queries, and data integrity features. For dynamic data, I used MongoDB (SoccerDB) to store collections like appearances, careers, club games, game events, game lineups, games, and rankings, leveraging its flexibility and efficiency in handling large volumes of unstructured data.

Issues: The primary challenge was to distinguish between dynamic and static data and choose appropriate storage methods for each. Dynamic data requires fast updates and retrieval, while static data benefits from the relational structure of PostgreSQL.

Requirements: The design meets the assignment requirements by using PostgreSQL for static data, ensuring data integrity and supporting complex queries, and MongoDB for dynamic data, providing flexibility and efficient handling of frequently changing information.

Limitations: To address potential limitations and ensure extensibility:

1. **Identifying Data Types:** User data, for example, is static, created once and stored as needed. Similar assessments were made for other data types.

2. **Data Organisation:** Original data from Transfermarkt and additional datasets like nations, player statistics, and user information were organised based on usage needs.

3. **Adding Tables and Collections:** For MongoDB, creating additional collections like rankings simplified data retrieval, given the non-real-time update requirement.

4. **Data Cleaning:** Using pandas and Python, incomplete or damaged data was cleaned, retaining only the most complete rows.

Overall, the design aims to balance efficiency, scalability, and ease of use, addressing key challenges and limitations proactively.

Task 2: Server Design

Solution: The JFT Soccer system uses three servers: the main Express server, the secondary Express server, and the Spring Boot server. Each server has a distinct role for efficiency, scalability, and maintainability.

Main Express Server: Handles front-end operations by serving static assets (HTML, CSS, JavaScript) and acts as a proxy to backend servers. Designed for simplicity, speed, and efficiency.

Key Features:

- Serves static files.
- Proxies API requests to backend servers (MongoDB, PostgreSQL).
- Uses Bootstrap, jQuery, and Axios for responsive design and data handling.

Advantages:	**Disadvantages:**
• Isolates front-end operations for independent scaling.	• Requires coordination between multiple servers.

• Lightweight setup ensures quick response times.	• Potential bottlenecks with increased user interactions.
• Modular code for better maintainability.	

Secondary Express Server: Handles dynamic data with MongoDB using the MVC architecture for clear separation of concerns and modularity.

Key Features:

- Uses MVC architecture.
- Efficiently manages dynamic, unstructured data.
- Provides CRUD API endpoints.

Advantages:	**Disadvantages:**
• Easy maintenance and scalability.	• Complexity in writing MongoDB queries.
• Efficient for dynamic data.	• Additional overhead in managing multiple servers.
• Clear codebase for readability and debugging.	

Spring Boot Server: Manages static data in PostgreSQL with a package structure based on data entities (clubs, competitions, nations), ensuring clear separation of concerns.

Key Features:

- Organized by data entities.
- Uses Spring Data JPA for data interactions.
- Provides RESTful APIs.

Advantages:	**Disadvantages:**
• Separates business logic, data access, and presentation layers.	• Learning curve for Spring Boot and JPA.
• Streamlines development and performance.	• Additional overhead in managing services.
• Clear codebase for debugging.	

Issues: The challenge is integrating these servers to work seamlessly together, ensuring efficient front-end operations, dynamic data management, and static data handling.

Requirements:

- Efficiently deliver static assets.

- Facilitate backend communication.
- Manage dynamic data (MongoDB).
- Handle static data (PostgreSQL).

Limitations:

- Scalability may require optimisation.
- Ensuring data consistency across services.
- Robust error handling is necessary.

Overall, the server design ensures efficient front-end operations, dynamic data management, and static data handling for the JFT Soccer platform.

Task 5: Chat System Implementation

Solution: The chat system, implemented using socket.io, facilitates real-time communication among fans and pundits. Socket.io's bidirectional communication capability ensures a seamless and interactive chat experience.

Key Features:

- **Real-Time Communication:** Enables instant messaging between clients and server.
- **Room-Based Chats:** Users join specific chat rooms for focused discussions.
- **User-Friendly Interface:** Intuitive design for easy message exchange.

Advantages:	**Disadvantages:**
• **Efficiency:** Minimal latency in message delivery.	• **Complexity:** Real-time communication management adds complexity.
• **Scalability:** Handles multiple chat rooms and many users.	• **Learning Curve:** socket.io integration can be challenging.
• **Interactivity:** Enhances user engagement with dynamic communication.	

Issues: Implementing a chat system using socket.io to handle real-time communication, multiple chat rooms, and a smooth user experience.

Requirements: The design meets the assignment requirements by providing robust real-time communication.

- **Client-Side Implementation:** Initialises the chat interface, connects to the server, handles user interactions.
- **Server-Side Implementation:** Sets up socket.io, manages user connections, chat rooms, and message broadcasting.

Limitations:

- **Scalability:** May face challenges with high user volume, requiring optimisation.
- **Network Reliability:** Real-time performance depends on network stability.

- **Security:** Ensuring secure communication and access control is crucial.

Overall, the chat system using socket.io offers a robust, scalable solution for real-time communication, enhancing interactivity and engagement on the JFT Soccer platform.

Task 6: Web Interface for Data Query

Solution: The web interface, built with Bootstrap, jQuery, and Axios, enables users to efficiently access and analyse football data. The main Express server serves static assets and acts as an intermediary between the client and backend servers, ensuring a seamless user experience.

Key Features:

- **Responsive Design:** Bootstrap ensures the interface looks good on all devices.

- **DOM Manipulation:** jQuery simplifies event handling and AJAX requests.

- **HTTP Requests:** Axios manages data fetching from backend servers.

- **Express Server:** Handles API requests and forwards them to backend servers.

Advantages:	**Disadvantages:**
• **Responsive:** Accessible on desktop and mobile.	• **Complexity**: Managing multiple libraries can add complexity.
• **Dynamic Data Handling: F**etches and displays data without page reloads.	• **Load Time:** Initial loading of static assets may take time.
• **Scalable:** Separation of concerns allows easy scaling and maintenance.	

Issues: Enable users to query the database via a web interface that provides easy access to football data.

Requirements: The design meets requirements by offering a user-friendly interface for querying data. Users can:

- View and search for clubs, competitions, and player statistics.

- Filter data by various criteria.

- Display results in a user-friendly format with detailed views.

Limitations:

- **Performance:** May degrade with large data volumes, requiring optimisation.

- **Input Handling:** Ensure user inputs are sanitised to prevent security issues.

- **Data Consistency:** Ensure data displayed is up to date.

Experience and Refactoring: The main challenge was structuring the JavaScript for the web interface. I refactored the code three times to improve maintainability.

1. **Initial Approach:** Created a large, hard-to-maintain function for all interactions.

2. **First Refactor:** Divided the function into smaller, macro area-specific functions, which still had issues with data formatting.

3. **Second Refactor:** Adopted an object-oriented approach, making the code more modular and maintainable.

Interactivity and Usability: The JFT Soccer web page is designed to be highly interactive, with multiple components that are clickable and interconnected. Users can navigate the data via the search bar or by clicking on various elements like nations, competitions, or clubs, providing multiple pathways to the same information and enhancing the user experience.

Task 7: Jupyter Notebooks for Data Analysis

Solution: Jupyter Notebooks were used for their interactive capabilities, allowing for combined code, text, and visualisations. Pandas handled data manipulation, while matplotlib and seaborn were used for visualisations.

Key Features:

- **Interactive Analysis:** Combines code, text, and visualisations.

- **Data Manipulation:** Utilises pandas for data processing.

- **Visualisation:** Uses matplotlib and seaborn for charts and graphs.

- **Documentation:** Inline markdown cells for clear documentation.

Advantages:	Disadvantages:
• **Ease of Use:** Immediate feedback on code execution.	• **Performance:** Slows with large datasets or complex computations.
• **Rich Visualizations:** Effective data insights with matplotlib and seaborn.	• **Version Control:** Challenging due to cell-based structure.
• **Documentation:** Clear and reproducible analysis process.	

Issues: Creating Jupyter Notebooks for football data analysis, including data cleaning, EDA, and visualisations.

Requirements: The notebooks comply by providing:

- **Data Cleaning:** Handling missing values and preprocessing data.

- **EDA:** Summary statistics and initial visualisations.

- **Advanced Analysis:** In-depth trend analysis and visualisations.

- **Documentation:** Clear step-by-step methodology.

Limitations:

- **Performance:** Slow with large datasets; may need optimisation.

- **Complex Visualizations:** Might require advanced techniques or additional libraries.

- **Collaboration:** Difficult due to merge conflicts in version control.