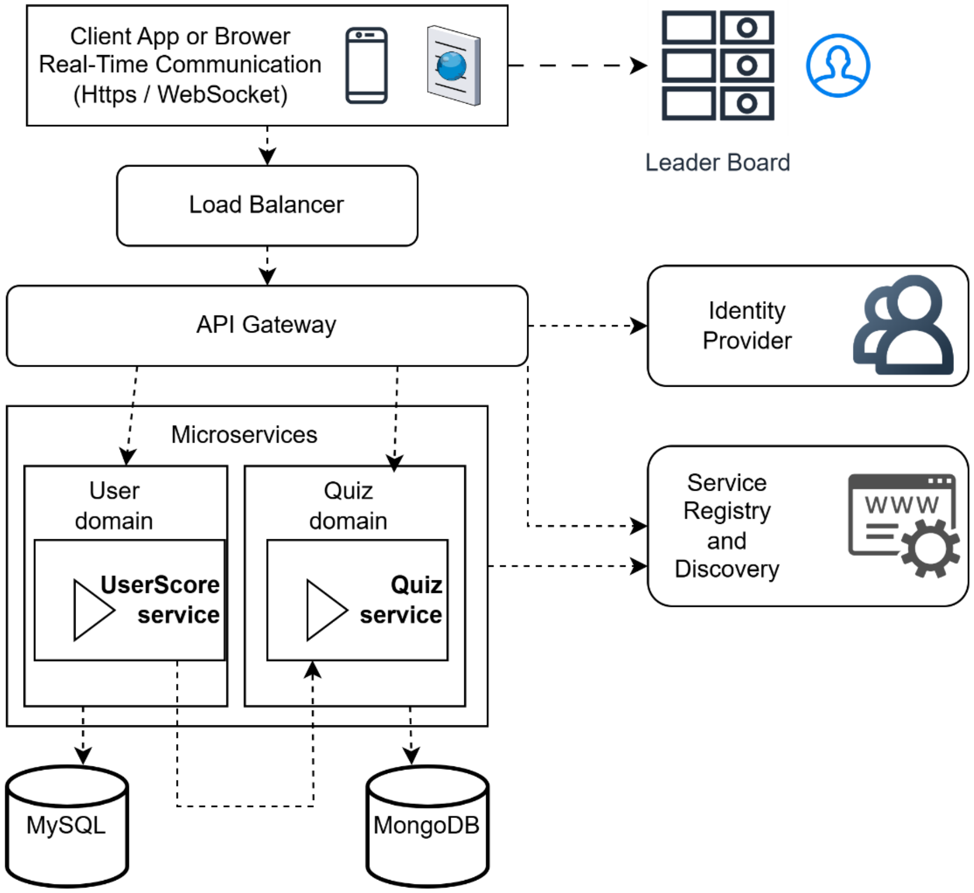
I - System Design



II - Component Description

The architecture supports **real-time scalability**, **loose coupling**, and **horizontal extensibility**—ideal for real-time quizzes with high concurrency and dynamic content.

Component	Description
Client App	- Interfaces through which users participate in quizzes. - Uses **HTTPS** for regular requests and **WebSocket** for real-time communication (e.g., score updates, live questions).
Load Balancer	Distributes incoming client requests across multiple backend instances to ensure high availability and scalability.
API Gateway	Request routing to microservices. Authentication/authorization (often via Identity Provider). Rate limiting, logging, and metrics.
Microservices Layer	Separated by **domain-driven design**: ✅ **User Domain: UserScore Service**

	• **Function**: ○ Manages user scores. ○ Tracks user participation and progress. ○ Provides real-time updates to leaderboard and clients. • **Data Storage**: **MySQL** – Relational data for users and scores. ✅ **Quiz Domain: Quiz Service** • **Function**: ○ Manages quizzes, questions, answer validation, and quiz sessions. ○ Coordinates quiz flow and tracks question timelines. • **Data Storage**: **MongoDB** – Flexible schema for quiz/question structures.
Databases	**MySQL**: Stores structured data like user information and score history. **MongoDB**: Handles unstructured/JSON-like data such as quiz content and configurations.
Identity Provider	Authenticates users and issues tokens (OAuth2, OpenID Connect). API Gateway and microservices to validate user sessions.
Service Registry & Discovery	Manages microservice instance registration and allows services to discover each other. **Common Tools**: Eureka, Consul, or Spring Cloud Discovery.

III - **Data Flow**

1. User Joins the Quiz

- **Action**: A user opens the app or browser and joins a quiz.
- **Flow**:
  - Via **HTTPS/WebSocket**, the client sends a **join request** to the **API Gateway**.
  - The **API Gateway** forwards the request to the **Identity Provider** for authentication.
  - Once authenticated, the request is routed to the **Quiz Service** (via service registry if needed).

2. Quiz Service Registers User Participation

- **Action**: Quiz Service logs the user’s participation in the session.
- **Flow**:
  - Updates its internal state (e.g., active participants, quiz session).
  - Stores relevant data in **MongoDB** (quiz sessions, question queues, user responses).
  - May notify other participants via **WebSocket** for real-time presence updates.

3. Quiz Execution

- **Action**: Quiz questions are served to participants.
- **Flow**:
  - **Quiz Service** sends questions to all participants via **WebSocket** for real-time delivery.
  - Participants submit answers via HTTPS/WebSocket to the **API Gateway**, which routes to the **Quiz Service**.
  - **Quiz Service** validates responses, computes score, and sends it to **UserScore Service**.

4. Score Calculation & Update

- **Action**: UserScore Service receives score update requests.
- **Flow**:
  - Updates user score in **MySQL**.
  - Publishes a message/event (or uses WebSocket callback) for **Leaderboard** component to consume (should be added to backlog as system improvement features).
  - Optionally sends acknowledgment back to the client (e.g., "Correct!", "+10 points").

---

5. Leaderboard Update

- **Action**: Leaderboard reflects the latest scores.
- **Flow**:
  - Subscribes to **score update events** from **UserScore Service** (via WebSocket push or event bus, should be added to backlog as system improvement features, using Redis Cache as an alternative solution).
  - Fetches top scores from **UserScore Service** or **MySQL**.
  - Updates and displays real-time leaderboard to clients.

---

6. Real-Time Feedback to Clients (Optional)

- **Action**: All connected clients receive real-time updates.
- **Flow**:
  - The server pushes leaderboard changes and score updates via **WebSocket**.
  - Clients render updated views without polling.

IV - **Technologies and Tools**

Frontend Stack: NodeJS, ReactJS, TypeScript, CSS

Backend Stack: Java, Spring Boot, Spring Data, Spring Security

Build Tools: Maven

Cloud Platforms applicable: AWS, Azure, GCP

CI/CD: Jenkins, Circle CI, etc.

Container Orchestration: Docker, Kubernetes

V - **AI Collaboration in Implementation**

AI-assisted tool: **ChatGPT**

+ Use GPT 4-5 model to prompt on possible system designs based on 2 approaches:

  + A general system design solution with only general problem overview prompted

+ In next prompt, asked ChatGPT to refine the provided solution to comply with 3 acceptance criterias.

=>Manually adding the missing parts of the AI-provided architecture to cover all features of "**Build For the Future**", such as: scalability, performance, reliability, maintainability, monitoring and observability.

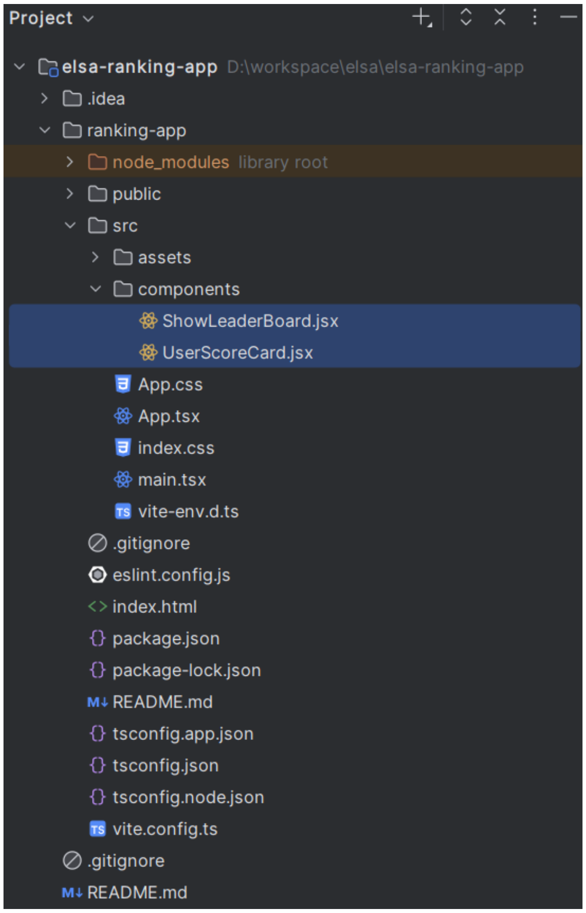
Deep-dive prompting with **ChatGPT**:

+ Microservices implementation:

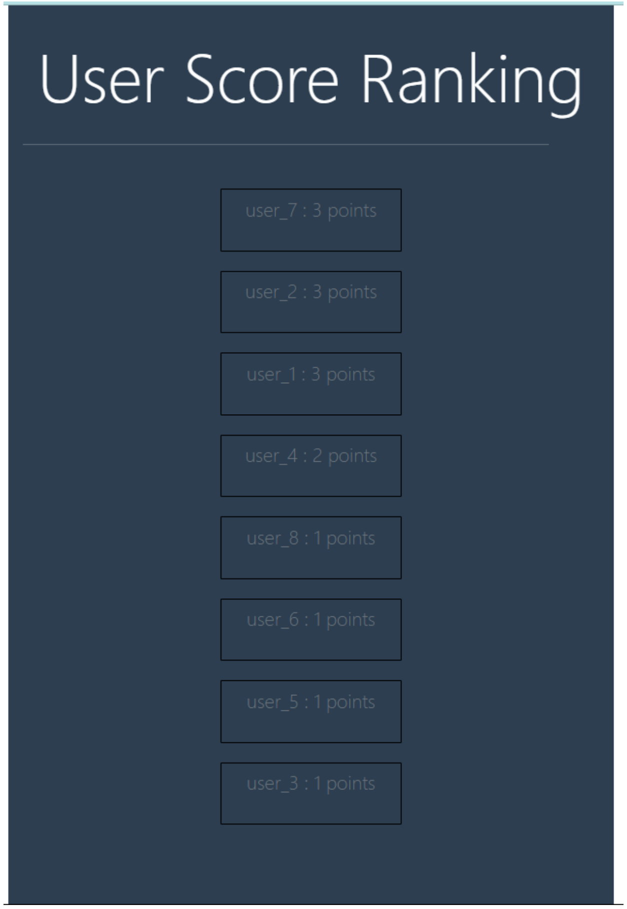
- Asked to generate controller classes (~70 % efficiency), spent effort to refactor ~30 % code.
- Asked to generate service classes (~50 % efficiency), refactor the rest.
- Asked to generate repository interfaces (~90 % efficiency).
- Asked to generate domain entities (~50 % efficiency).
- Manually coding for DTO, exceptions, controller advices, utility services, configuration, etc.
- Asked to generate test code (~50% efficiency, manually refactored).
- 100% manually created functional tests.
- Spent manual effort for others (Docker, Kubernetes configuration, etc)

+ Frontend application:

- Implemented elsa-ranking-app manually due to such a small micro-frontend application with 2 components required:



```
Project
elsa-ranking-app D:\workspace\elsa\elsa-ranking-app
  .idea
  ranking-app
    node_modules library root
    public
    src
      assets
      components
        ShowLeaderBoard.jsx
        UserScoreCard.jsx
      App.css
      App.tsx
      index.css
      main.tsx
      vite-env.d.ts
    .gitignore
    eslint.config.js
    index.html
    package.json
    package-lock.json
    README.md
    tsconfig.app.json
    tsconfig.json
    tsconfig.node.json
    vite.config.ts
  .gitignore
  README.md
```



User Score Ranking

user_7 : 3 points

user_2 : 3 points

user_1 : 3 points

user_4 : 2 points

user_8 : 1 points

user_6 : 1 points

user_5 : 1 points

user_3 : 1 points

VI – Source code repositories:

Microservices:

https://github.com/longleth/elsa-coding-challenges

https://github.com/longleth/elsa-quiz-service

Client Application:

https://github.com/longleth/elsa-ranking-app