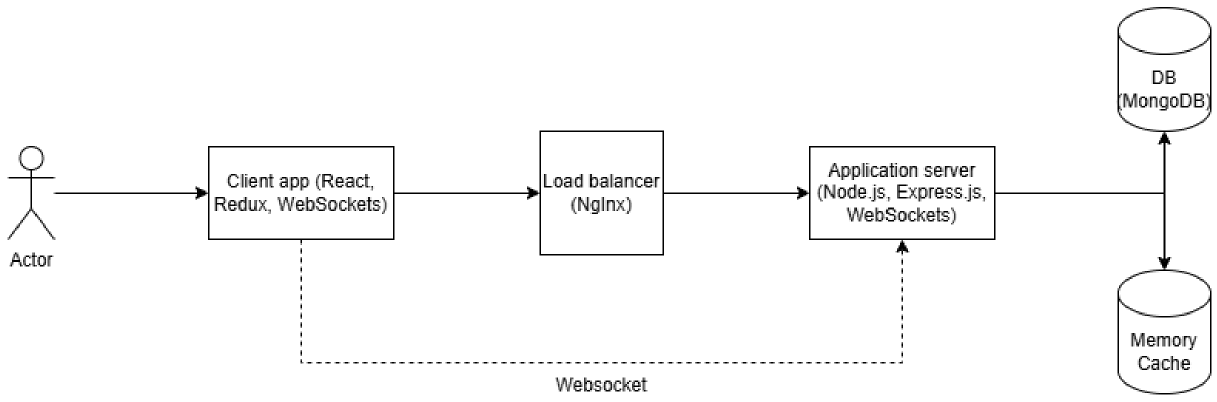
Part 1: System Design

Architecture Diagram



Actor
Client app (React, Redux, WebSockets)
Load balancer (Nginx)
Application server (Node.js, Express.js, WebSockets)
DB (MongoDB)
Memory Cache
Websocket

Component Description

- **Client App**: This app handles user interactions, sends requests to the server, and updates the leaderboard in real-time.
- **Load Balancer**: Distributes incoming traffic across multiple application servers.
- **Application Server** (Node.js): Manages quiz sessions, updates scores, and broadcasts leaderboard updates.
- **Database** (MongoDB): Stores quiz data, user information, and scores.
- **Memory Cache**: Stores and updates leaderboard data in real-time.
- **WebSocket**: Enables real-time communication between clients and the server.

Data Flow

- The user joins a quiz session using a unique quiz ID.
- The client app sends a request to the application server.
- The application server verifies the quiz ID and creates a new quiz session.
- The user submits answers, and the client app sends requests to the application server.
- The application server updates scores in the database and memory cache.
- The application server broadcasts leaderboard updates to connected clients using WebSocket.

Technologies and Tools

- Client App: React, Redux, and WebSocket.
- Application Server: Node.js, Express.js, and WebSocket.
- Database: MongoDB.
- Memory Cache: For leaderboard storage and updates.

- Load Balancer: NGINX.

Justification

- Scalability: Node.js and MongoDB enable horizontal scaling.
- Real-time updates: WebSocket ensure real-time communication.
- Performance: Memory cache optimises leaderboard updates.
- Reliability: MongoDB and Memory Cache provide data redundancy.

Demo

Real-time Quiz Challenge