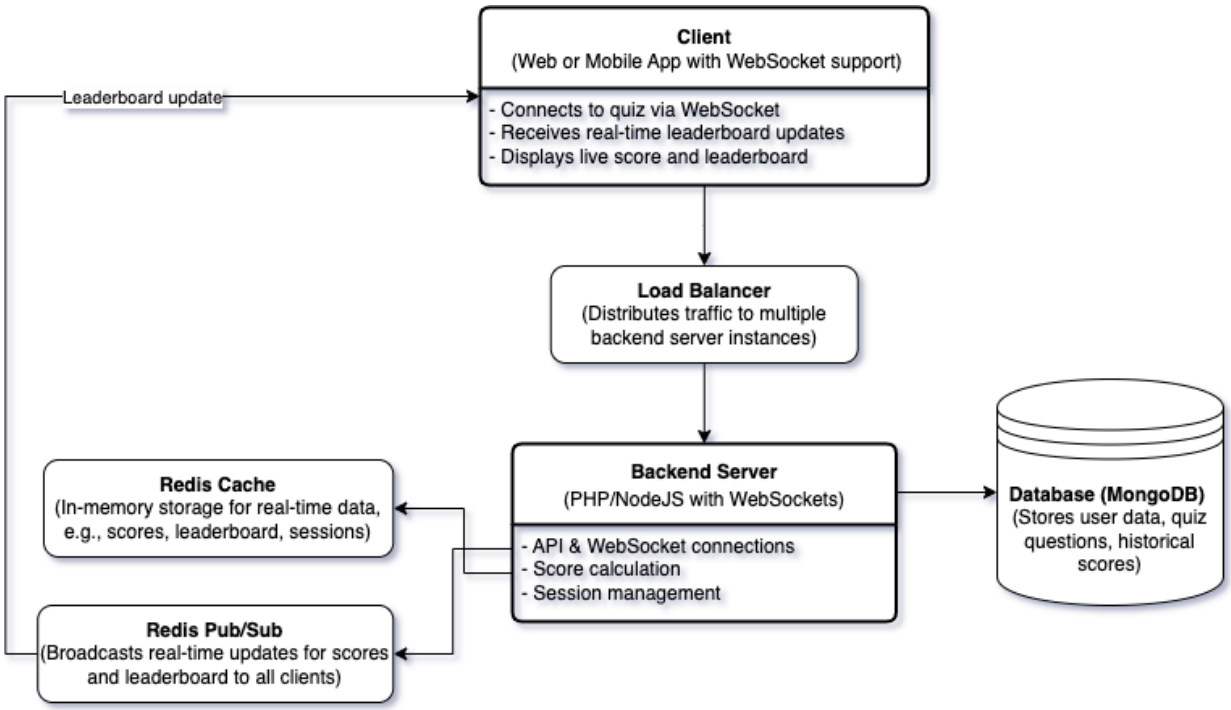
System Design Document: Real-Time Quiz Feature

1. Architecture Diagram

The architecture diagram below illustrates how different components interact to enable real-time quiz functionality with a live leaderboard.



---

2. Component Description

Frontend Client (Web/Mobile)

- **Role**: Provides the user interface for participants to join the quiz, answer questions, and view real-time leaderboard updates.
- **Details**:
  - Connects to the backend via WebSocket for real-time communication.
  - Submits answers, receives scores, and renders the leaderboard updates dynamically.

Backend Server

- **Role**: Manages quiz sessions, processes answers, updates scores, and facilitates real-time communication with clients.
- **Details**:
  - Hosts REST APIs for user login, quiz session creation, and answer submission.

    - Maintains WebSocket connections for real-time updates to clients.
    - Uses Redis for low-latency data retrieval and storage of active sessions, scores, and leaderboard information.

Redis Cache

- **Role**: Stores real-time, volatile data such as current scores and session information for fast access and low latency.
- **Details**:
    - Uses a sorted set to store and update scores, enabling efficient leaderboard ranking.
    - Caches active quiz data, minimizing database load.

Redis Pub/Sub

- **Role**: Broadcasts real-time updates on score changes and leaderboard status to all clients in a given quiz session.
- **Details**:
    - Provides a publish-subscribe mechanism for immediate notification of state changes across client connections.

Database (MongoDB)

- **Role**: Stores persistent data such as user profiles, quiz questions, historical scores, and completed quiz results.
- **Details**:
    - Stores long-term data not needed in real-time, supporting data analysis and quiz result storage.

---

3. Data Flow

User Joins Quiz

1. User provides a unique quiz ID through the client app.
2. Backend server validates the quiz ID and establishes a WebSocket connection.
3. Backend uses Redis to track active sessions and the user's session data is stored in Redis for quick access.

Answer Submission

1. User submits an answer via the WebSocket connection.

2. Backend server validates the answer and calculates the score.
3. The score is updated in the Redis sorted set for the session, ensuring fast and ordered leaderboard updates.

Leaderboard Update

1. After updating the score, the backend triggers a Redis Pub/Sub notification with the updated leaderboard data.
2. All clients connected to the quiz session receive the leaderboard update in real-time.

---

4. Technologies and Tools

Component	Technology	Justification
Frontend (Web/Mobile)	React / Swift / Kotlin	Popular frameworks with WebSocket support, efficient for dynamic UIs.
Backend Server	Node.js / PHP	Asynchronous and real-time capabilities, ideal for WebSocket connections.
Real-Time Communication	WebSocket	Necessary for low-latency, bidirectional communication for real-time updates.
In-Memory Store	Redis	Low-latency in-memory data store, efficient for caching and sorted sets.
Pub/Sub	Redis Pub/Sub	Fast and scalable pub/sub mechanism for broadcasting leaderboard updates.
Database	MongoDB	Schema flexibility, fast read/write for quiz data, and efficient for JSON-like documents.