**1. Architecture Diagram**

Here is the architecture diagram illustrating the interaction among system components:

┌──────────────────────┐

│ Client (Web) │

│ - Quiz UI │

│ - Leaderboard UI │

│ │

└───────┬──────────────┘

│ WebSocket / HTTP

│

┌───────▼──────────────┐

│ Node.js Server │

│ - Express │

│ - Socket.IO │

└───────┬──────────────┘

│

│ Redis Pub/Sub

│

┌───────▼──────────────┐

│ Redis Cache │

│ - Leaderboard Store │

│

└──────────────────────┘

**2. Component Description**

1. **Client (Web Application)**:
   * **Function**: Provides the interface for users to participate in quizzes, view and submit answers, and see their score on a real-time leaderboard.
   * **Components**:
     + *Quiz UI*: Displays the questions, options, and submit button.
     + *Leaderboard UI*: Shows real-time rankings of users participating in the quiz.
2. **Node.js Server**:
   * **Function**: Acts as the backend for handling client requests, managing quiz sessions, scoring, and the leaderboard.
   * **Subcomponents**:
     + *Express*: Provides the API enpoints and handles HTTP communication.
     + *Socket.IO*: Enables real-time, bidirectional communication between the server and clients, allowing live updates to the quiz and leaderboard.
3. **Redis Cache**:
   * **Function**: Stores leaderboard information in a highly performant and in-memory structure to enable quick retrieval and real-time updates.
   * **Subcomponents**:
     + *Leaderboard Store*: Stores leaderboard scores in a structure that’s sorted for easy rank retrieval.

**3. Data Flow**

**Data Flow Overview**

1. **User Joins Quiz**:
   * The user sends a join-quiz message to the server with a quizId.
   * The current leaderboard is sent to the user’s client.
2. **User Submits Answer**:
   * The user answers all quiz questions and clicks submit.
   * The answers and quizId are sent to the submit-answer message.
   * The server calculates the user’s score based on correct answers.
   * The updated score is stored in the Redis leaderboard.
   * A score update is broadcasted to all connected clients in the quiz, updating the leaderboard.
3. **Leaderboard Update**:
   * Every time a user’s score is updated, the server fetches the updated leaderboard from Redis.
   * The leaderboard is sorted and broadcasted to all clients via Socket.IO, providing real-time rank visibility.

**4. Technologies and Tools**

1. **Frontend (Web Application)**:
   * *React.js*: Provides a dynamic and interactive UI for quiz participation and leaderboard display.
   * *Socket.IO Client*: Enables real-time updates from the server for both quiz progress and leaderboard updates.
2. **Backend (Node.js Server)**:
   * *Node.js with Express*: Lightweight and performant, ideal for handling multiple quiz participants and concurrent requests.
   * *Socket.IO Server*: Facilitates real-time, bidirectional event-based communication between clients and server.
   * *Redis*:
     + *Data Store*: Redis is used for caching leaderboard data and session data, enabling quick access and real-time data updates.
     + *Pub/Sub Model*: Redis Pub/Sub allows easy propagation of messages across multiple components, especially useful for updating leaderboards across instances.
3. **Additional Libraries**:
   * *dotenv*: Loads environment variables securely from the .env file.
   * *uuid*: Generates unique IDs for quiz sessions.