### 

### **Component Description**

1. **Client Application (React)**:
   * **Role**: Interface for users to join quiz sessions, answer questions, and view the leaderboard.
   * **Technologies**: React.js for building interactive user interfaces, Axios for making HTTP requests to the backend.
2. **Frontend Container**:
   * **Role**: Runs the React application within a Docker container.
   * **Technologies**: Node.js for building the React app, serve for serving the production build.
3. **Backend Container**:
   * **Role**: Runs the FastAPI server to handle quiz sessions, user authentication, answer processing, and score updates.
   * **Technologies**: FastAPI, Uvicorn for ASGI server, SQLAlchemy and asyncpg for database interaction.
4. **PostgreSQL Container**:
   * **Role**: Stores user data, quiz questions, session information, scores, and leaderboard standings.
   * **Technologies**: PostgreSQL for relational data storage.

### **Data Flow**

1. **User Joins Quiz**:
   * User opens the client application and enters a unique quiz ID to join a session.
   * Client application sends a request to the Backend Container via HTTP.
   * FastAPI validates the quiz ID and adds the user to the session, storing the data in PostgreSQL.
   * Confirmation is sent back to the client application.
2. **Answer Submission and Score Calculation**:
   * User submits an answer through the client application.
   * Answer is sent to the Backend Container via HTTP or WebSocket.
   * FastAPI processes the answer, calculates the score, and updates the user’s score in PostgreSQL.
   * FastAPI sends the updated score back to the client application.
3. **Real-Time Leaderboard Update**:
   * FastAPI broadcasts the updated scores to all connected clients via WebSocket.
   * Client applications receive the updates and refresh the leaderboard in real-time.

### **Technologies and Tools**

1. **Client Application**:
   * **React.js**: To build the interactive UI.
   * **Axios**: For making HTTP requests to the FastAPI backend.
2. **Frontend Container**:
   * **Node.js (18-alpine)**: For building and serving the React application.
   * **serve**: To serve the built React application.
3. **Backend Container**:
   * **FastAPI**: For handling backend logic and APIs.
   * **Uvicorn**: As the ASGI server.
   * **SQLAlchemy and asyncpg**: For database ORM and async interaction.
4. **PostgreSQL Container**:
   * **PostgreSQL**: For storing application data.
5. **Docker Compose**:
   * **docker-compose.yml**: To manage and run multi-container Docker applications.

This architecture ensures that the real-time quiz feature is scalable, maintainable, and easy to deploy using Docker and Docker Compose. Each component is isolated in its container, making it easier to manage dependencies and scale the application.