Real-Time Vocabulary Quiz Challenge System Design

1. Architecture Diagram 2

Key Components: 2

2. Component Descriptions 3

a. Client Applications (Web and Mobile) 3

b. API Gateway / Backend Server 3

c. SignalR Hub 3

d. Database 3

e. Caching Layer (Redis) 3

3. Data Flow 3

Step 1: User Joins Quiz Session 3

Step 2: User Submits an Answer 3

Step 3: Real-Time Leaderboard Update 4

4. Technologies and Tools 4

Core Technologies 4

Database 4

Development Tools 4

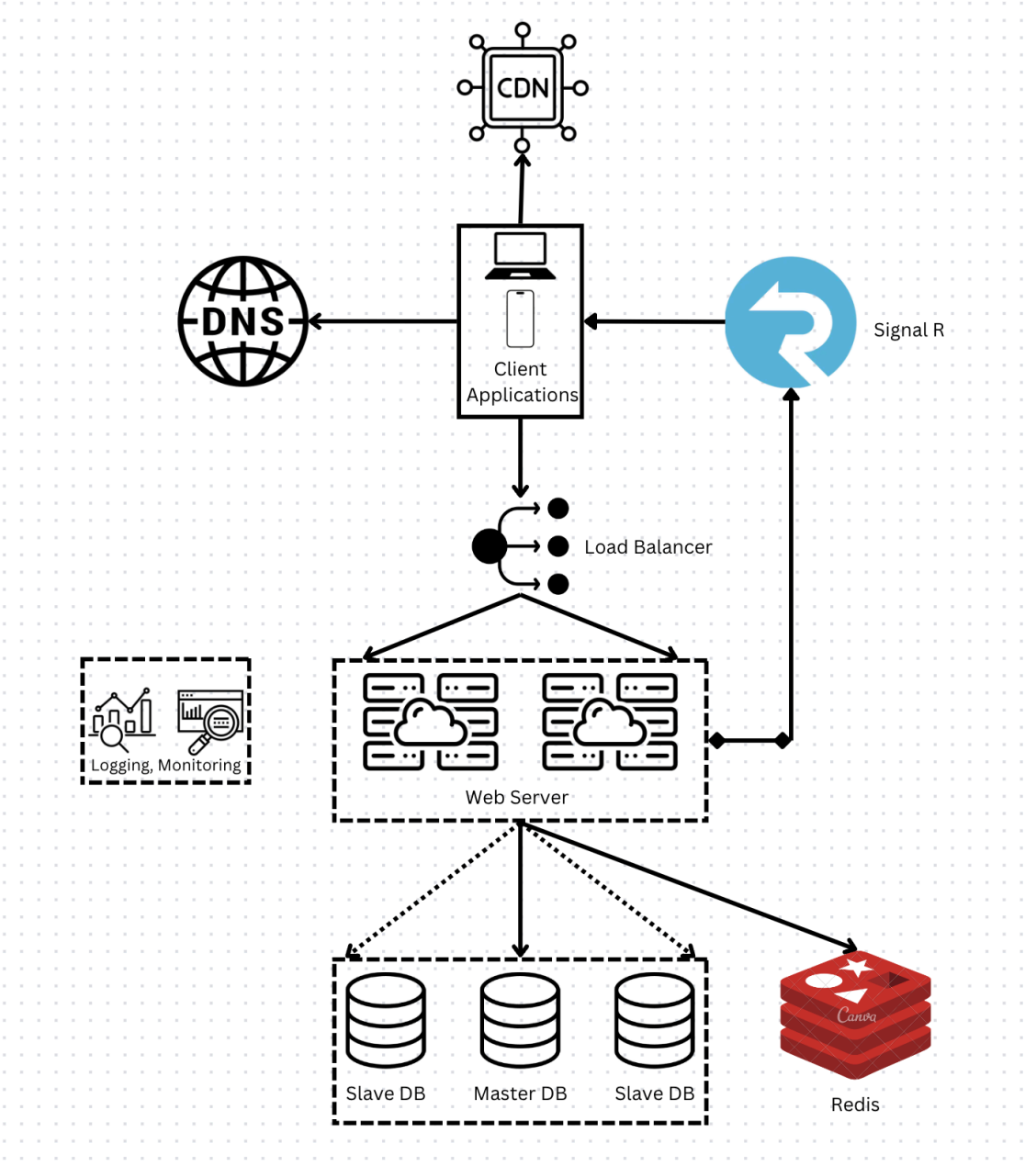
Justification of Technologies 4

1. Architecture Diagram

The architecture of the Real-Time Vocabulary Quiz feature follows a client-server model that prioritizes real-time communication. This design includes several components, each with a specific role in enabling seamless interaction between the client applications, backend, and database.

Key Components:

- **Client Applications** (e.g., Web App, Mobile App)
- **API Gateway/Backend Server**
- **SignalR Hub** (for real-time data communication)
- **Database** (for storing quiz data, user scores, and leaderboard information)
- **Caching Layer** (Redis or any cache system for session and leaderboard data)
- **External Services** (for future enhancement)



2. Component Descriptions

Each component contributes to the real-time quiz functionality as follows:

a. Client Applications (Web and Mobile)

- **Role**: Provides an interface for quiz participation, score tracking, and leaderboard viewing.
- **Real-Time Integration**: Connects to the SignalR Hub to receive real-time score and leaderboard updates.

b. API Gateway / Backend Server

- **Role**: Acts as the central access point for client interactions, managing quiz participation requests, answer submissions
- **Core Responsibilities**: Processes HTTP API requests, validates inputs, and interacts with the database to retrieve or store data as needed.

c. SignalR Hub

- **Role**: Facilitates real-time communication, allowing data to be broadcast across all connected clients during a quiz session via unique quiz ID
- **Functionality**:
  - Allows clients to join specific quiz groups based on quiz ID.
  - Receives backend updates (e.g., score changes) and broadcasts them to all users in the group.
- **Justification**: SignalR in .NET Core 8 is optimized for high-efficiency real-time updates, capable of handling thousands of concurrent connections.

d. Database

- **Role**: Maintains core data such as user profiles, quiz session details, answers, scores, and leaderboard records.
- **Justification**: Persistent data storage is essential for scoring, leaderboard management, and historical data access.

e. Caching Layer (Redis)

- **Role**: Ensures fast access to frequently updated data, such as active session details and leaderboard positions.
- **Functionality**: Stores session data in memory to facilitate quick retrieval and updates without overloading the main database.
- **Justification**: Redis offers high performance for real-time applications with in-memory data storage, enhancing response times in session management.

3. Data Flow

The following steps outline how data flows through the system, from when a user joins a quiz to when leaderboard updates are displayed:

Step 1: User Joins Quiz Session

- **Client Request**: The client app sends a `JoinQuiz` request to the API Gateway with a unique quiz ID.
- **API Gateway**: Validates the request and retrieves relevant session information.
- **SignalR Hub**: Adds the user's connection to the SignalR group linked to the quiz ID.
- **Database**: Updates the session data to indicate the new participant.

Step 2: User Submits an Answer

- **Client Request**: The answer submission is sent from the client app to the API Gateway.
- **Backend Processing**:
  - The server verifies the answer and calculates the score.

    - Updates the score in both the database and potentially in Redis for quick access. (Using the in-memory cache for storing the score)
- **SignalR Broadcast**: The server uses SignalR to broadcast the updated score to all users in the session.

Step 3: Real-Time Leaderboard Update

- **Database**: Updates leaderboard standings based on the latest scores.
- **Redis Cache**: Stores the updated leaderboard data to ensure fast retrieval.
- **SignalR Hub**: Broadcasts the refreshed leaderboard to all clients in the session.
- **Client Display**: The client applications receive the leaderboard update and immediately reflect the changes in real time.

4. Technologies and Tools

The following technologies and tools are selected to ensure reliability, scalability, and real-time performance:

Core Technologies

- **ASP.NET Core 8**: A robust framework for building scalable APIs, supporting REST and SignalR for real-time capabilities.
- **SignalR for .NET Core**: Manages real-time connections, allowing efficient data broadcast to users within quiz sessions.
- **Entity Framework Core**: Simplifies database interactions, providing an ORM layer for CRUD operations on user data and scores.

Database

- **SQL Server**: Handles structured data storage (e.g., users, scores, leaderboard), supporting complex queries and ensuring transactional integrity.
- **Redis**: Provides in-memory caching for high-speed access to leaderboard and session data, reducing the load on SQL Server.

Development Tools

- **Visual Studio/Visual Studio Code/Jetbrain Rider**: IDEs for coding, debugging, and testing the solution.
- **Postman**: Facilitates API testing, particularly useful for verifying real-time updates.

Justification of Technologies

- **Real-Time Communication**: SignalR is ideal for real-time .NET applications, enabling fast updates to all participants.
- **Performance and Scalability**: Redis, combined with SQL Server, allows the system to handle high user loads without compromising speed.
- **Data Integrity**: SQL Server supports data consistency, which is critical for scoring accuracy, while Entity Framework manages data relations efficiently.