Software Architecture Design Specification

Real-Time Quiz

Document ID:	
Total: 15	Page No. 1

Revision History

Content

1 Introduction

This document outlines the design of a real-time quiz feature for an English learning application where users can participate in dynamic quizzes, view live leaderboards, and access their score history. The system is designed to support scalability, multimedia questions, and user-friendly features.

Intended for developers, software architects, and system administrators involved in designing, building, and maintaining the real-time quiz application, this document provides a comprehensive overview of the architecture, database design, data flow, and key functionalities.

2 References

For the fundamental requirements, please consult this link.

3 Architecture Overview

The architecture consists of client applications, a backend server, a centralized database, and AWS services for automation and cleanup. Below is a detailed explanation of the components and how they interact.

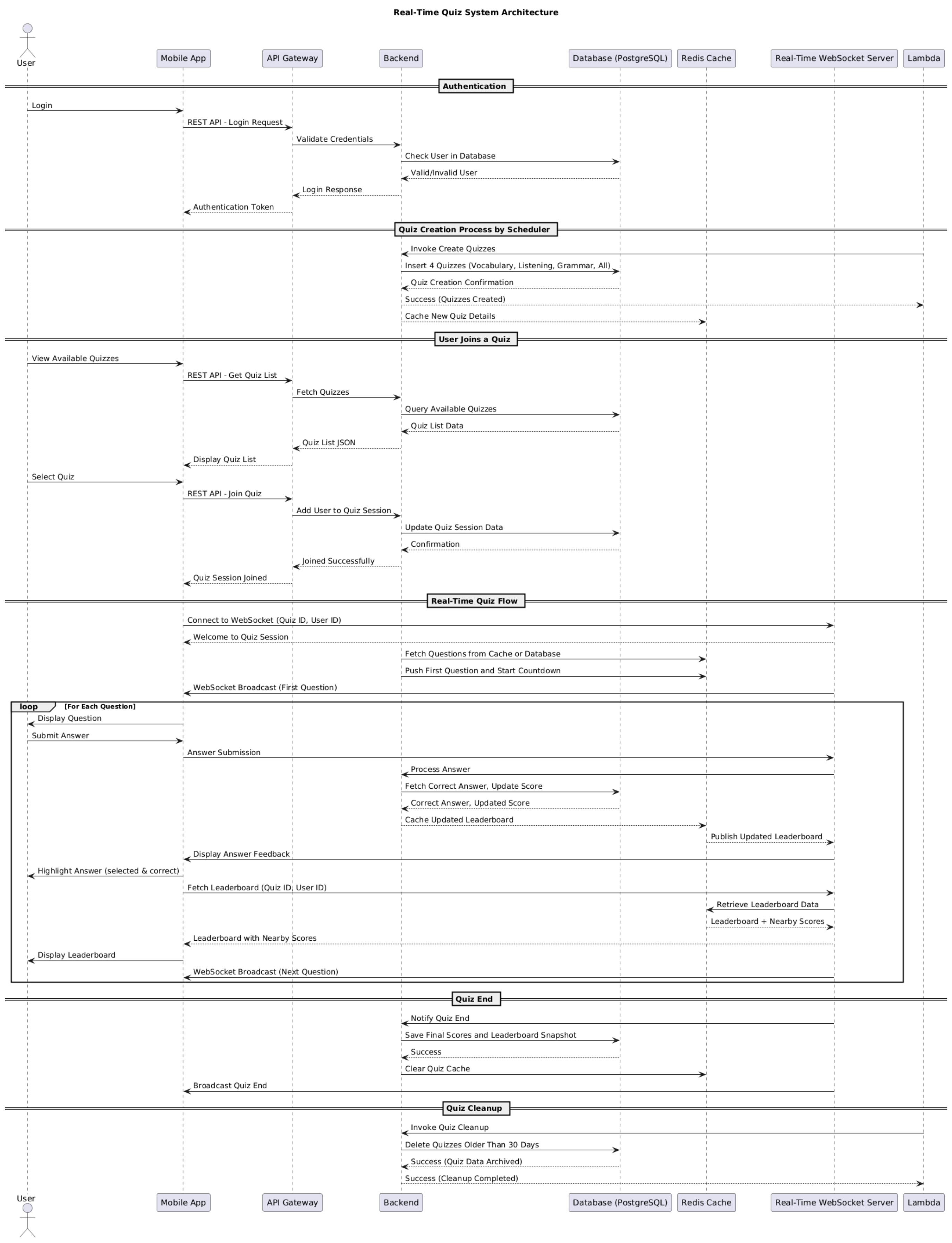
Real-Time Quiz System Architecture
User
Mobile App
API Gateway
Backend
Database (PostgreSQL)
Redis Cache
Real-Time WebSocket Server
Lambda
Authentication
Login
REST API - Login Request
Validate Credentials
Check User in Database
Valid/Invalid User
Login Response
Authentication Token
Quiz Creation Process by Scheduler
Invoke Create Quizzes
Insert 4 Quizzes (Vocabulary, Listening, Grammar, All)
Quiz Creation Confirmation
Success (Quizzes Created)
Cache New Quiz Details
User Joins a Quiz
View Available Quizzes
REST API - Get Quiz List
Fetch Quizzes
Query Available Quizzes
Quiz List Data
Quiz List JSON
Display Quiz List
Select Quiz
REST API - Join Quiz
Add User to Quiz Session
Update Quiz Session Data
Confirmation
Joined Successfully
Quiz Session Joined
Real-Time Quiz Flow
Connect to WebSocket (Quiz ID, User ID)
Welcome to Quiz Session
Fetch Questions from Cache or Database
Push First Question and Start Countdown
WebSocket Broadcast (First Question)
loop [For Each Question]
Display Question
Submit Answer
Answer Submission
Process Answer
Fetch Correct Answer, Update Score
Correct Answer, Updated Score
Cache Updated Leaderboard
Publish Updated Leaderboard
Display Answer Feedback
Highlight Answer (selected & correct)
Fetch Leaderboard (Quiz ID, User ID)
Retrieve Leaderboard Data
Leaderboard + Nearby Scores
Leaderboard with Nearby Scores
Display Leaderboard
WebSocket Broadcast (Next Question)
Quiz End
Notify Quiz End
Save Final Scores and Leaderboard Snapshot
Success
Clear Quiz Cache
Broadcast Quiz End
Quiz Cleanup
Invoke Quiz Cleanup
Delete Quizzes Older Than 30 Days
Success (Quiz Data Archived)
Success (Cleanup Completed)

Figure 3-1: System overview

3.1 Client Application (Mobile App)

The English learning mobile app, a cross-platform mobile app for iOS and Android, is the user interface for the system. It provides an intuitive experience for users to browse quizzes, participate, and see real-time updates.

Users can view quizzes by name but also see quiz history sorted by date for traceability.

The app establishes WebSocket connections for real-time updates and uses REST APIs for non-real-time actions.

3.2 Backend

The server acts as the heart of the system, processing requests and managing the flow of data.

It handles:

- Quiz logic, such as selecting unique questions.
- Real-time updates via WebSocket.
- Scoring and leaderboard calculations.

3.3 Database

A relational database stores all data in a structured manner, including:

- Questions Table: Maintains the pool of questions, with metadata for text, image, and audio.
- Quizzes Table: Tracks each quiz, ensuring unique IDs and timestamped creation.
- Scores and Leaderboards: Stores participant performance for real-time and historical analysis.

It also enables complex queries for detailed insights (e.g., historical scores or user progress).

Instead of using multiple databases, to balance complexity and performance, a single PostgreSQL database with multiple tables and proper indexing is chosen. This ensures easier management while leveraging PostgreSQL's scalability features (e.g., partitioning, indexing, caching).

3.4 WebSocket Server

Real-time communication between the mobile app and backend. It directs real-time updates (e.g., score and leaderboard changes) to connected users

A scalable WebSocket server can be hosted on AWS using Elastic Beanstalk or EC2 instances.

3.5 Caching

Redis provides low-latency caching for frequently accessed data like active quizzes and leaderboard rankings.

The Redis caching layer not only ensures low-latency leaderboard access but also decouples WebSocket communication from database writes. In scenarios where Redis is unavailable, the backend falls back to querying the database directly, with a potential performance trade-off.

3.6 Scheduled Jobs

AWS Lambda automates quiz creation and cleanup to ensure system scalability.

Its features including:

- Quiz Scheduler: Creates new quizzes every 15 minutes with 4 types: vocabulary, listening, grammar, and mixed manage transitions between quiz's states (upcoming → active → completed)
- Cleanup Process: Deletes quizzes older than 30 days but preserves user score and leaderboard history.

3.7 Other components

API Gateway: Serves as a single-entry point for the REST APIs.

Load Balancer: Distributes traffic to backend servers for scalability.

CloudWatch: Monitors backend performance and logs.

4 Data Flow

4.1 Quiz Creation

The scheduler (AWS Lambda) triggers every 15 minutes and:

- Creates 4 quizzes (vocabulary, listening, grammar, and all types) which are inserted into the database. Quizzes start 3 minutes after being listed
- Unique Quiz IDs are generated using the format xxxYYYYMMDDHHMM where xxx is abbreviation of quiz type (voc for vocabulary, lis for listening, gra for grammar, and all for all - mixed questions from previous 3 types) and YYYYMMDDHHMM is the creation timestamp.
- Quiz names are user-friendly format quiz type + DDHHMM (e.g., Vocabulary 271415, Listening 011730, Grammar 310045, All 090500), but the full quiz ID is retained for history tracking. For certain special occasions, quiz names can incorporate playful wordplay such as Vocabulary Virtuoso, Listening Luminary, Grammar Guru, and All Ace.
- Selects 10 questions for each quiz, ensuring no duplicates by checking question IDs. Questions can include images, audio, and text.

4.2 User Joins Quiz

User retrieves the list of available quizzes via a REST API.

On selecting a quiz, the client sends a POST request to the backend API /join_quiz/ with.

```
{
  "quiz_id": "quiz123",
  "user_id": "user456"
}
```

The server registers the user in the quiz and sends a confirmation.

4.3 Quiz Execution

Questions are streamed one by one at 10-second intervals (to show scores and leaderboard after each).

Users have 20 seconds per question to answer, with their score calculated as:

$$Points = Question\ Score\ \times Seconds\ Left\ at\ Submission$$

The question score may vary. For more details, refer to section 7.1.

The app sends the answer over the WebSocket:

```
{
  "question_id": "q1",
  "user_id": "user456",
  "selected_option": 2,
  "remaining_seconds": 15
}
```

Mobile app highlights both the selected answer and the correct answer on the UI for the user and disables further changes to the answer for that question.

4.4 Leaderboard Updates

After each question, scores are recalculated for all participants.

When multiple users submit answers simultaneously, scores are updated in Redis. If conflicts arise (e.g., two users achieve the same score), Redis employs a sorted set data structure to rank users by score and submission time. Updates are batch-processed into PostgreSQL periodically.

The leaderboard is updated and broadcast via WebSocket.

It also includes the total score of the user and the scores of the 2 participants immediately above and below the user. If the user is in the first or last position, the display will be adjusted to show either the next or previous participant only.

4.5 Cleanup

The cleanup process removes old quizzes while retaining historical scores and leaderboards.
Alternatively, set an automatic expiration time (TTL) for all quiz-related keys in Redis. This ensures stale data is automatically removed if manual cleanup fails.

5 Technology Justification

Component	Technology	Justification
Mobile App	Flutter	A single codebase simplifies cross-platform app development, ensuring consistent user experiences on Android and iOS.
Backend	FastAPI (Python)	High-performance, easy-to-implement, and suitable for REST APIs and WebSocket communication.
Database	AWS PostgreSQL	Reliable, scalable, and supports advanced queries for data consistency and performance. Provides relational data storage with support for JSONB, ensuring flexibility for multimedia questions.
Real-Time Updates	WebSocket	Enables low-latency, two-way communication for real-time score and leaderboard updates.
Scheduler	AWS Lambda	Serverless, cost-efficient, and easily integrates with AWS services to handle quiz creation and cleanup.

6 Database Design

6.1 Questions Table

Stores all questions with detailed metadata for categorization and validation.

```
{
   "question_id": "TEXT (Primary Key)",
   "type": "TEXT (e.g., vocabulary, listening)",
   "content": "TEXT (e.g., question text or description)",
   "audio_url": "TEXT (nullable, URL for audio-based questions)",
   "image_url": "TEXT (nullable, URL for image-based questions)",
   "options": "JSONB (Array of possible answers)",
   "correct_option": "INTEGER (Index of the correct answer)",
   "created_at": "TIMESTAMP DEFAULT NOW()"
}
```

The `type` field is used to filter questions based on type requirements (vocabulary, listening, etc.).
The `options` JSONB field is used to store the possible answers for a question in a flexible, structured format. This format supports text, image, or audio-based answers by including metadata for each option, such as the type and content (e.g., a URL for an image or audio file), making it suitable for dynamic and diverse types of possible answers.
Each option is stored as an object in a JSON array, containing attributes like:

- `type`: Specifies the type of the option (text, image, audio).
- `content`: Holds the actual answer text or a URL pointing to an image/audio resource.
- Additional `metadata` (optional), such as alt text for images or transcripts for audio. It can also be used (e.g., audio presence in listening quizzes).

Example questions:

- Text-Based Question

```
[
   {"type": "text", "content": "Apple"},
   {"type": "text", "content": "Orange"},
   {"type": "text", "content": "Banana"},
   {"type": "text", "content": "Grape"}
]
```

- Image-Based Question

```
[
   {"type": "image", "content": "https://example.com/images/apple.jpg"},
   {"type": "image", "content": "https://example.com/images/orange.jpg"},
   {"type": "image", "content": "https://example.com/images/banana.jpg"},
   {"type": "image", "content": "https://example.com/images/grape.jpg"}
]
```

- Audio-Based Question

```
[
   {"type": "audio", "content": "https://example.com/audio/apple.mp3"},
   {"type": "audio", "content": "https://example.com/audio/orange.mp3"},
   {"type": "audio", "content": "https://example.com/audio/banana.mp3"},
   {"type": "audio", "content": "https://example.com/audio/grape.mp3"}
]
```

Frontend Rendering

- Parse the `options` JSONB and render each option based on its `type`.
- Display
    - o Text answers as clickable buttons.

  - Images using `<img>` tags.
  - Audio options using `<audio>` elements

6.2 Quizzes Table

Stores metadata about each quiz, including its type, unique ID.

```
{
   "quiz_id": "TEXT (Primary Key, format: xxxYYYYMMDDHHMM)",
   "state": "TEXT (e.g., upcoming, active, completed)",
   "type": "TEXT (e.g., vocabulary, listening)",
   "name": "TEXT (e.g., question text or description)",
   "question_ids": "JSONB (Array of question ids)",
   "created_at": "TIMESTAMP DEFAULT NOW()"
}
```

To retrieve the contents of the associated questions, perform a join operation with the questions table based on the question IDs stored in the `question_ids` array then cache the results (the content of all questions for a given quiz) in Redis. This will offload frequent reads from the database and make access much faster, as Redis is designed for quick retrieval of data

6.3 Score Table

Stores individual scores for each participant in a quiz.

```
{
   "id": "INTEGER (Primary Key)",
   "quiz_id": "TEXT (Foreign Key to Quizzes Table)",
   "user_id": "TEXT",
   "score": "INTEGER",
   "updated_at": "TIMESTAMP DEFAULT NOW()"
}
```

6.4 Leaderboard Table

Caches leaderboard rankings for quick access.

```
{
  "quiz_id": "TEXT (Unique identifier for the quiz)",
  "leaderboard": [
    {
      "user_id": "TEXT (Unique identifier for the user)",
      "score": "INTEGER (User's score in the quiz)",
      "rank": "INTEGER (User's rank in the quiz)"
    },
    {
      "user_id": "TEXT",
      "score": "INTEGER",
      "rank": "INTEGER"
    }
  ],
  "nearby_scores": {
    "user_score": {
      "user_id": "TEXT (The requesting user's ID)",
      "score": "INTEGER (The requesting user's score)",
      "rank": "INTEGER (The requesting user's rank)"
    },
    "above": {
      "user_id": "TEXT (User ID of the next higher scorer)",
      "score": "INTEGER (Score of the next higher scorer)",
      "rank": "INTEGER (Rank of the next higher scorer)"
    },
    "below": {
      "user_id": "TEXT (User ID of the next lower scorer)",
      "score": "INTEGER (Score of the next lower scorer)",
      "rank": "INTEGER (Rank of the next lower scorer)"
    }
  },
  "updated_at": "TIMESTAMP (Timestamp of the last leaderboard update)"//
}
```
Redis maintains a cache of the leaderboard as a sorted set, where scores are the ranking key. This structure supports efficient range queries for fetching top scores or nearby scores in O(log(n)) time.
Leaderboards in Redis have a time-to-live (TTL) of 30 minutes after the quiz ends. This reduces memory usage while ensuring leaderboard data is readily available during the quiz.

7 Scoring Logic and Leaderboard

7.1 Scoring Logic

Each question is assigned a fixed score of 100 points. The points awarded to users who answer correctly are determined by the "winners take all" rule. This means that all users who answer correctly will receive an equal share of the 100 points, divided by the number of correct answers. However, the score per user will not be less than 1 point.

- Question Score Calculation:
  Divide 100 by the number of correct answers:
  $$Question\ Score = \left[\frac{100}{number\ of\ correct\ submissions}\right]$$
  If no correct answers, no points are awarded for the question.
- User Score (for a question, if the answer is correct):
  Multiply the points by the remaining time at submission:
  $$User\ Score = Question\ Score \times Time\ Left\ (Seconds)$$

Example:
Question score = [100/19] = 5 points.
User answers with 15 seconds left:
User Score = 5 × 15 = 75.

7.2 Leaderboard Calculation

Sort all participants by their total score in descending order.
In case of ties, rank participants by the earliest submission time for the last question answered.
Sample JSON response

```
{
  "leaderboard": [
    {"user_id": "user_1", "score": 250, "rank": 1},
    {"user_id": "user_2", "score": 200, "rank": 2},
    {"user_id": "user_3", "score": 150, "rank": 3}
  ],
  "nearby_scores": {
    "user_score": {"user_id": "user_2", "score": 200, "rank": 2},
    "above": {"user_id": "user_1", "score": 250, "rank": 1},
    "below": {"user_id": "user_3", "score": 150, "rank": 3}
  }
}
```

8 Error Handling

This section describes some common errors and their corresponding error handling strategies.

- Database Unavailability: If PostgreSQL is unavailable, leaderboard updates are queued in an in-memory buffer and retried periodically.
- WebSocket Disconnections: If a user disconnects, their progress is saved, and they can resume by reconnecting within 1 minutes. After 1 minutes, the session is considered abandoned.

9 Security

This section describe a few security considerations.

- Data encryption:
  - WebSocket communications are secured via TLS.
  - REST API endpoints require JWT tokens for authentication.
- Rate limiting: Users are restricted to submitting one answer per question within the allowed time window to prevent spamming

10 Performance Metrics

This section describes some performance metrics.

- Maximum WebSocket connections: 10,000 per server instance.
- Average leaderboard query time: 2 ms (cached in Redis).
- Average question broadcast latency: <100 ms.

11 Monitoring and Alerts

This section describes monitoring and alerts.

- AWS CloudWatch monitors
  - Memory usage of Redis
  - Latency for WebSocket connections
  - Error rates for API endpoints
- Alerts
  - Trigger if WebSocket connection count exceeds 80% of the server limit.

12 Future Enhancements

This section describes potential future enhancements.

- Implement AI-based adaptive quizzes that adjust question difficulty based on user performance.
- Allow quiz creators to add custom questions via an admin panel.
- Integrate a real-time chat feature for participants during the quiz.