# *QUIZICALLY*

## PRODUCT DESIGN SPECIFICATION

Version 1.0
10/10/2025

# VERSION HISTORY

| Version # | Implemented By | Revision Date | Approved By | Approval Date | Reason |
|---|---|---|---|---|---|
| 1.0 | Khadija, Rana, Palmer, Jake, Katira | 10/10/2025 | | | Initial Design Definition draft |
| | | | | | |
| | | | | | |

**UP Template Version:** 12/31/07

# TABLE OF CONTENTS

## 1 INTRODUCTION

### 1.1 PURPOSE OF THE PRODUCT DESIGN SPECIFICATION DOCUMENT

The Product Design Specification document documents and tracks the necessary information required to effectively define architecture and system design in order to give the development team guidance on architecture of the system to be developed. The Product Design Specification document is created during the Planning Phase of the project. Its intended audience is the project manager, project team, and development team. Some portions of this document such as the user interface (UI) may on occasion be shared with the client/user, and other stakeholder whose input/approval into the UI is needed.

# 2 GENERAL OVERVIEW AND DESIGN GUIDELINES/APPROACH

This section describes the principles and strategies to be used as guidelines when designing and implementing the system.

## 2.1 ASSUMPTIONS / CONSTRAINTS / STANDARDS

- Stable funding and staffing remain available for the full development window.
- Target users (bar patrons and hosts) have smartphones and reliable internet access.
- Third-party AI services used for question generation are available and affordable.
- Mobile platforms (iOS/Android) continue to support required features without major disruption.

**Constraints:**

- Delivery within 6 months with phased prototypes and pilot testing.
- Working budget of $150,000; changes require sponsor approval via change control.
- Must comply with Apple App Store / Google Play policies and Colorado Privacy Act (CPA).
- Cross-platform support (iOS + Android) and up to 100 concurrent players.
- Availability target: 99% uptime during events; maintenance scheduled and announced.

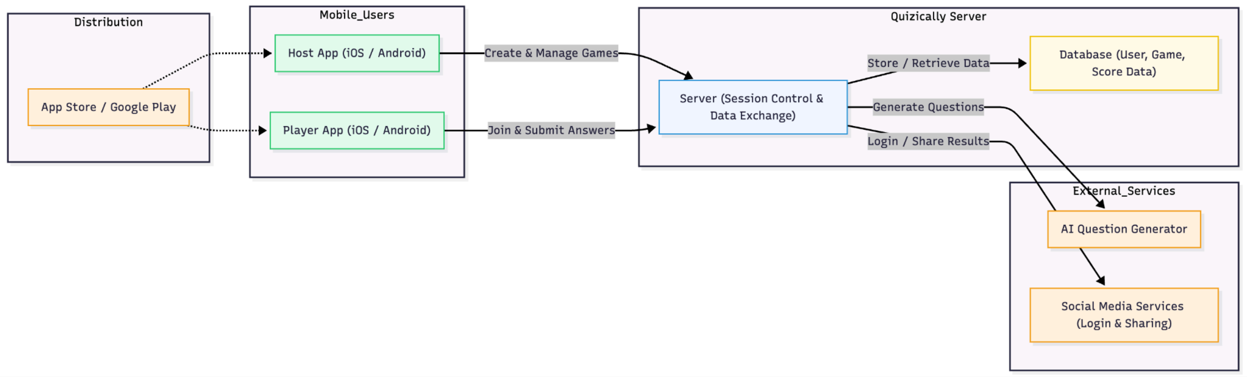
**Standards:**

- Auth & Security: User sign-in will be handled securely using the OAuth 2.0 protocol for easy social logins. To guard against common threats, we will follow the OWASP Application Security Verification Standard (ASVS) and enforce strong TLS encryption for all communications.
- Privacy & Compliance: The application will be designed to comply with the Colorado Privacy Act (CPA). This means we will practice data minimization by only collecting essential user information, obtain explicit consent, and encrypt all sensitive data both in transit and at rest.
- UX/UI: Our design will follow established platform guidelines, Apple's Human Interface Guidelines for iOS and Google's Material Design for Android, to ensure a familiar user experience. We will also prioritize accessibility by following WCAG 2.1 AA principles for color contrast, touch target size, and more.
- Quality/Perf Targets: We are targeting a high level of reliability and responsiveness. Our goal is to achieve 99% uptime during live events, a server response time (TTFB) of less than one second with 100 concurrent users, and include features that allow players to safely rejoin a game after a timeout.

## 3 ARCHITECTURE DESIGN

This section outlines the system and hardware architecture design of the system that is being built.

Quizically utilizes a mobile–cloud architecture, where both the Host and Player mobile applications connect to a centralized backend server over the internet. The server handles communication between users, stores data, and connects with approved third-party services. The system is designed for accessibility, reliability, and scalability during live trivia events.



### Data Flow Explanation

1. Host App → Backend: Sends game setup information such as number of questions or categories.

2. Backend → AI Service: Requests automatically generated questions and formats them for the host.

3. Player App → Backend: Sends player responses and requests updated scores.

4. Backend → Database: Saves game sessions, user profiles, and scoring data.

5. Backend ↔ Social Media Services: Handles user authentication and limited profile retrieval.

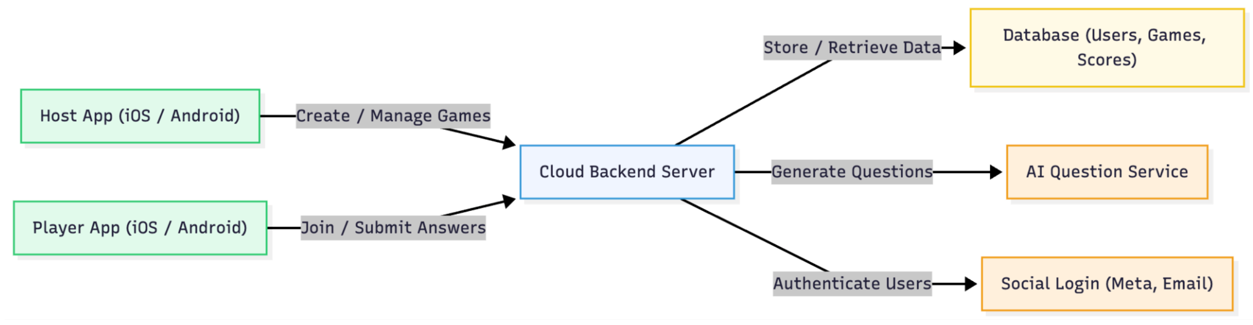
6. Backend → Player/Host Apps: Sends game updates, leaderboards, and final results.

Hardware:

1. Client Devices: Smartphones or tablets (iOS and Android) used by hosts and players.

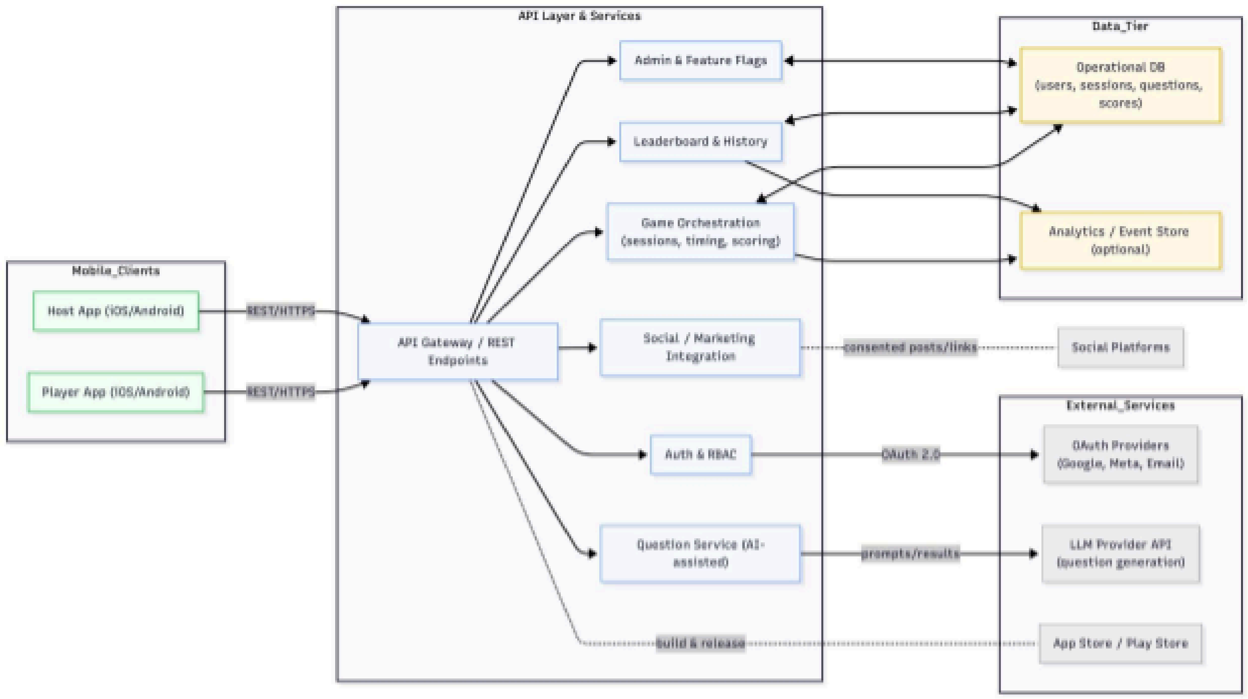
2. Server Environment: Cloud-hosted environment (such as AWS or Google Cloud) providing scalability, availability, and performance.

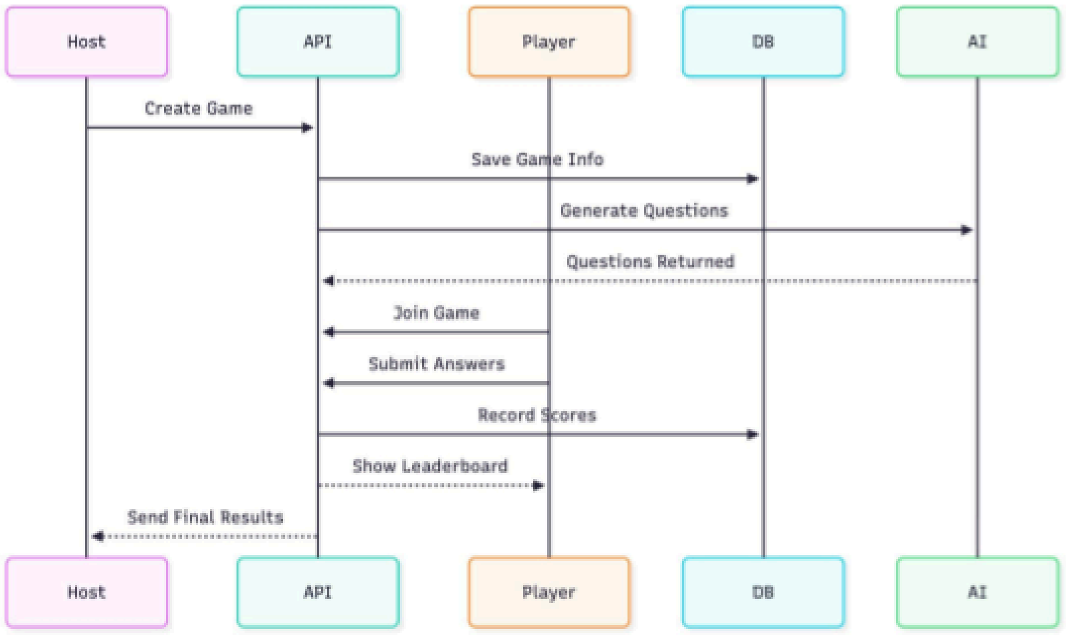
3. Database: Cloud-managed storage ensuring persistence and security of user/game data.



## 3.1 LOGICAL VIEW

- Host creates a game → backend persists session → optional AI call generates question set → host starts game → players join → answers are submitted → scoring/leaderboard updated → results archived for history/marketing.
- Player authenticates (social/email), joins via code/QR, submits answers, views team and global standings, and can review past games.


API Layer & Services
Data_Tier
Mobile_Clients
External_Services
Host App (iOS/Android)
Player App (iOS/Android)
REST/HTTPS
REST/HTTPS
API Gateway / REST Endpoints
Admin & Feature Flags
Leaderboard & History
Game Orchestration (sessions, timing, scoring)
Social / Marketing Integration
Auth & RBAC
Question Service (AI-assisted)
Operational DB (users, sessions, questions, scores)
Analytics / Event Store (optional)
consented posts/links
Social Platforms
OAuth 2.0
OAuth Providers (Google, Meta, Email)
prompts/results
LLM Provider API (question generation)
build & release
App Store / Play Store



## 3.2 HARDWARE ARCHITECTURE

The Quizically system utilizes a mobile-cloud architecture , and the hardware components are designed to ensure reliability and scalability during live trivia events. The key hardware components are:

- Client Devices: Smartphones or tablets (iOS and Android) used by hosts and players.
- Server Environment: A Cloud-hosted environment (such as AWS or
  Google Cloud) is used to provide scalability, availability, and performance for the backend server.
- Database: Cloud-managed storage is used to ensure the persistence and security of user and game data.

## 3.3 SOFTWARE ARCHITECTURE

Quizically uses a client-server architecture with cloud hosted services and mobile clients. The system follows a modular, layered design to separate concerns and improve scalability.

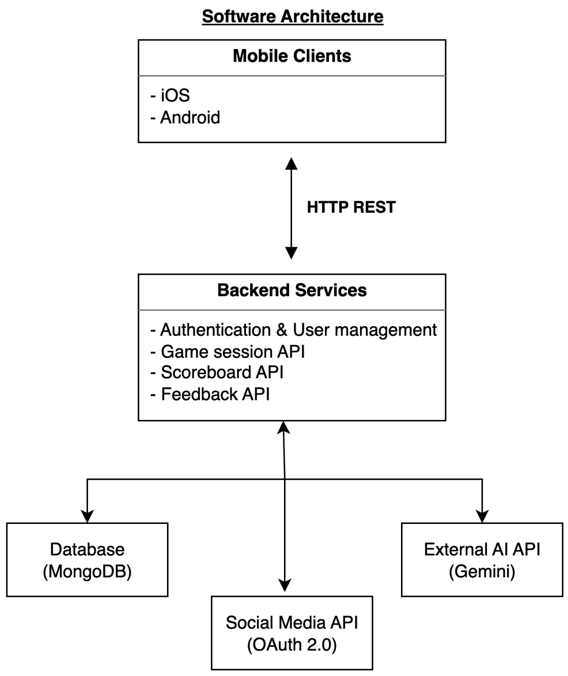


*Figure 3.3 depicts the organization of the Quizically system, detailing interactions between the mobile clients and backend server through REST. The server consists of three components, the MongoDB database, the Gemini API for the gameset generation AI tool, and the OAuth2.0 API for social media integration. (Tool: draw.io)*

## 3.4 SECURITY ARCHITECTURE

Quizzically is designed to ensure robust protection of sensitive user data and to mitigate potential security threats. The system follows OWASP, ASVS, and GDPR guidelines to protect user data and prevent unauthorized access.

- Authentication and Access Control:
  - The system uses OAuth2.0 to enable users to securely log in via social media platforms and ensure that their credentials are securely managed.

- ○ Role Based Access Control ensures only authorized users can access sensitive features.
- Data Encryption:
  - TLS, Transport Layer Security, encrypts all communications between the app and backend server to prevent unauthorized data access during transmission.
  - AES-256 encryption will secure sensitive data in the backend database.
- Compliance with Security Standards:
  - The app follows OWASP ASVS for security and GDPR for user data protection, such as providing users with the ability to manage and delete their data.
- Threat Prevention:
  - The app includes firewalls, secure coding practices, and intrusion detection systems to prevent security threats. Firewalls are configured to filter out suspicious traffic while intrusion detection will monitor for any potential threats.
- Incident Response:
  - An incident response plan will be in place to manage any security breaches or data loss incidents.
- Reference Documents:
  - Security- related documentation, including the vulnerability assessments and incident response plans will be stored in the security repository.

○

## 3.5 COMMUNICATION ARCHITECTURE

Quizically uses a mobile-cloud communication model that connects the mobile applications of the host and players to a centralized cloud backend. This setup guarantees real-time trivia gameplay, secure data transfer, and scalability during live events.

Communications components that Quizically employs is:

- Client-to-Server Communication:
  - Mobile clients communicated with the backend via REST APIs over HTTPS.
  - Players authenticate using OAuth 2.0 with social sign-ins (Meta, TikTok, Email). Secure tokens are then used for subsequent API calls.
- Server-to-Third-Party Communication:
  - The backend communicates with AI Question Generation Service (via OpenAI API) to create trivia question sets.

    - The backend communicates with OAuth 2.0 providers for secure authentication and limited user data retrieval.

## 3.6 PERFORMANCE

Quizically implements performance strategies such as:

- Cloud-Based Scaling:
    - Backend services are hosted in a scalable cloud environment, ensuring stability while handling simultaneous player connections at peak usage.
- Optimized Communication Paths:
    - Lightweight REST calls and efficient database access patterns minimize server load. Core interactions within a game (i.e. answer submission, leaderboard updates) are kept as stateless as possible to reduce contention.
- Load Distribution:
    - The architecture allows for integration of load balancers and caching layers if concurrency demands increase.
- Resilience and Failover:
    - Quizically contains graceful degradation mechanisms, such as fallback to manual question sets or hotspot internet if external APIs or internet connectivity fails.
- Monitoring and Maintenance:
    - Regular monitoring tracks uptime, latency, and error rates. These maintenance windows are planned outside peak usage time to preserve availability.

# 4 SYSTEM DESIGN

This section provides a detailed overview of how the core system components interact to support the Host and Player use cases. It expands upon the high-level architecture by describing the end-to-end behavior of the Create Game, Start Game, Question Display, and Scoring components, including all relevant data flows between the Host App, Player App, Backend Server, Database, and third-party services such as Gemini and OAuth.

## 4.1 USE-CASES

The core functionality of the Quizically platform is driven by the following key user roles and their associated use case flows, as outlined in the Logical View and Architecture Design.

Host Use Cases:

- Create and Manage Games - The Host initiates a new game, persists the session, and optionally calls the AI service to generate the question set. The Host manages the game state and timing.
- Start Game/Allow Joins - The Host starts the game to allow players to join via code or QR. The questions created by the host will be displayed onto the host’s device so they can project it for all the players.

- View/Archive Results - The Host is responsible for sending the final results and ensuring the results are archived for history and marketing purposes.

Player Use Cases:

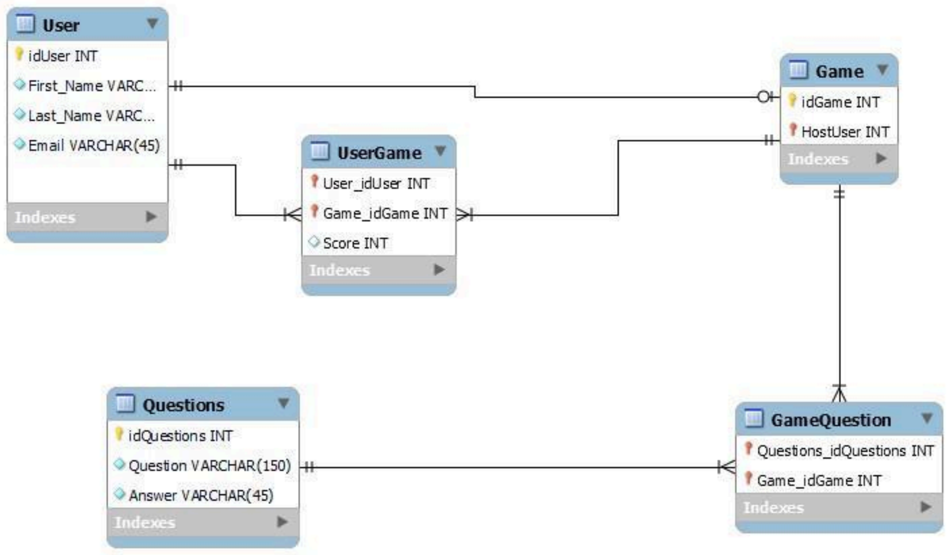
- Authenticate - The Player authenticates using a social sign-in provider (e.g., Google, Meta) or email (OAuth 2.0).
- Join Game - The Player joins a live session via a game code or QR code.
- Submit Answers - The Player submits answers during the game, which are recorded and used for score calculation.
- View Leaderboard - The Player views team and global standings (the leaderboard) during the game.
- Review History - The Player can review past game results

When a Host creates a game, the Host App sends game parameters to the Backend, which then optionally contacts the Gemini AI service to generate a question set. The Backend processes the AI response and saves each generated question into the Questions collection in the database. It then creates GameQuestion links that associate those questions with the newly created game. Players interact only with the Backend during gameplay—submitting answers and receiving updates—while the Host App controls game progression. This ensures that all system components (Host App, Player App, Backend, Database, and AI services) operate cohesively within each use case.

## 4.2 DATABASE DESIGN

The Quizically system's data persistence layer will be implemented using the following technology and design principles:

- Database System: MongoDB is the chosen NoSQL database technology.
- Data Stored (Operational DB): The database's primary function is to serve as the Operational DB for storing: Users, Games, and Questions



(Example DB)

## 4.3 APPLICATION PROGRAM INTERFACES

The Quizically platform will expose and consume several APIs to support functionality for both mobile clients and backend services. These APIs ensure consistent communication between system components, enabling scalability, maintainability, and third-party integration.

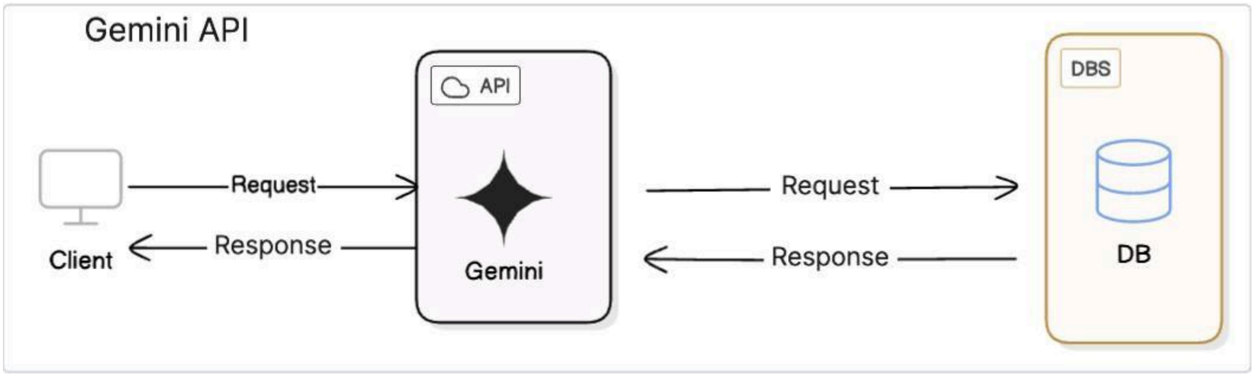


*Figure 4.3.1 depicts the service architecture of the Gemini API that will be utilized to build Quizically's Gameset Generation AI Tool. The client (user device) will send a GET request to Gemini API (free tier) and receive the desired response.(Tool: eraser.io)*

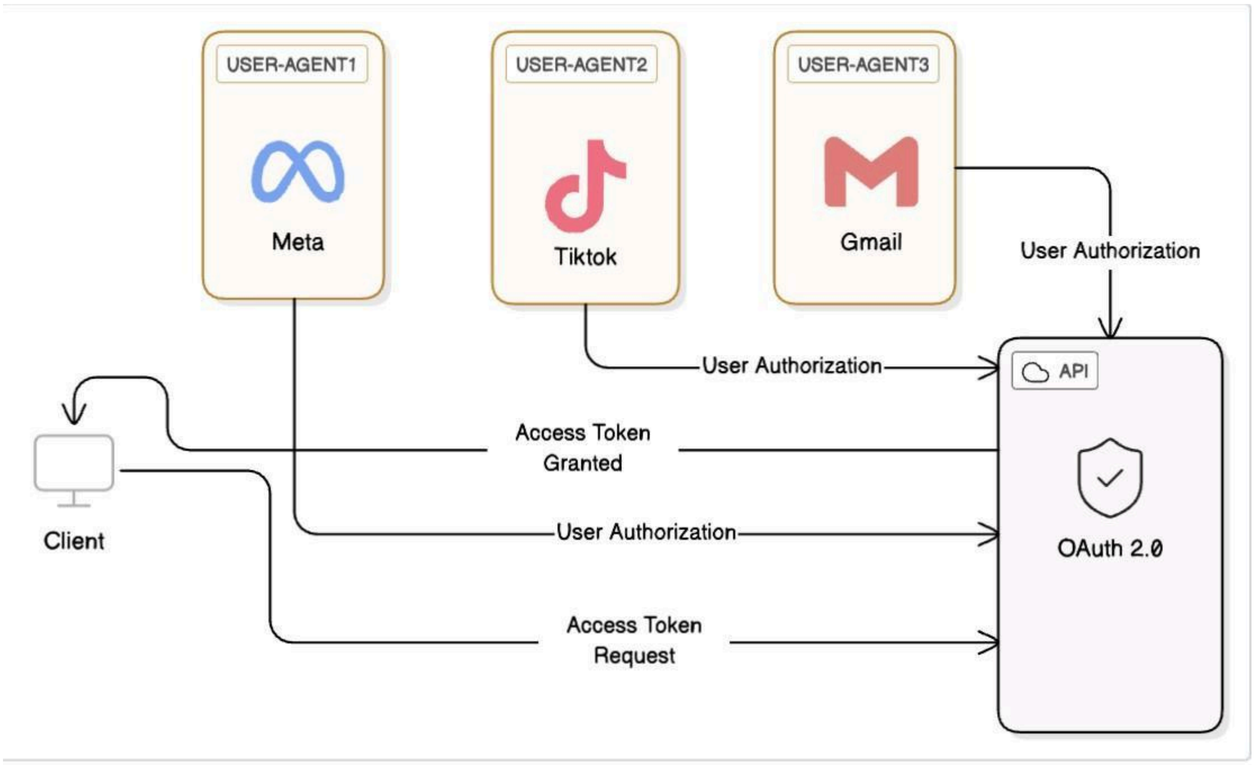


*Figure 4.3.2 depicts the service architecture of the OAuth 2.0 API that will be used for our social media integration component. It allows for our third-party application,*

*Quizically, to access user data from service providers (Meta, Tiktok, Gmail) securely utilizing tokens for permissions. (Tool: eraser.io)*

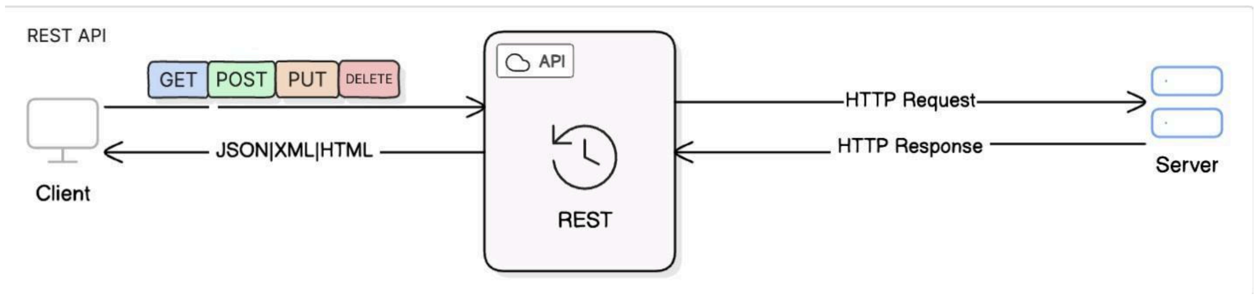


*Figure 4.3.3 depicts the service architecture of the REST API that will be used to support our client-server architecture for communication between the mobile clients (user devices) and our backend server. (Tool: eraser.io)*

When the createGame component sends a request to the Gemini API, the Backend parses the returned question-and-answer JSON and stores each question as a document in the Questions collection of the database. It then automatically populates the GameQuestion table by linking each saved question to the active game's gameId. After the database is updated, the Backend returns the full question set to the Host App for review and editing. This ensures that Gemini's output is fully integrated into the game's data model before the host selects or modifies questions.

## 4.4 USER INTERFACE DESIGN

The Quizically user interface is designed to be intuitive, mobile-friendly, and aligned with the functional roles of Hosts and Players. While a shared homepage provides universal access to major features, the UI diverges based on user type after authentication to ensure that Hosts and Players are guided toward their respective tasks efficiently.

**Landing Pages (Host vs. Player):**

When any user launches Quizically, they arrive at a general landing page containing navigation options to host or join a game, browse public game sets, access guides, view support resources, or manage their account. The host and join options then guide them to two different dashboards.

- **Host Dashboard:** Hosts see tools for creating games, editing their game library, generating question sets with AI, and launching active game sessions.

- **Player Dashboard:** Players see a streamlined interface focused on joining a game via code or QR and accessing past gameplay history.

This separation ensures that Hosts can access game management functions without cluttering the UI presented to Players.

**Join Game Flow:**

When a Player selects “Join Game,” the application displays a simple, dedicated join screen. This screen includes an input field for entering the alphanumeric game code, a button to activate the QR scanner, and a confirmation action. Once the code is validated by the backend, the Player is taken to a lobby screen, which displays the game name, participant count, and a waiting message until the Host begins the session. The lobby provides clear feedback so players know they have successfully joined the correct game.

**Create Game Flow:**

When a Host selects “Create Game,” the UI transitions to the Host’s game library, where existing games are displayed and options for creating a new game are available. Creating a game opens a configuration screen where the Host selects question categories, difficulty, and quantity or chooses to use the AI generation feature. If AI-generated questions are requested, the Host is shown a preview of the generated content and can edit questions before saving. This workflow ensures that Hosts can efficiently create high-quality game sets while maintaining full control over content.

**Host Game Flow:**

When a Host selects “Host Game,” the interface presents the Host with a game session control screen. This includes the unique join code and QR code that the Host may project for participants. Once the game begins, the Host’s screen displays the active question, answer options, and controls such as “Next Question,” “Show Answer,” “Show Leaderboard,” and “End Game.” The Host device serves as the primary display for the trivia questions and can be projected on a larger screen if desired.

**Player Question and Response Flow:**

Players do not receive all questions at once. Instead, when the Host advances to the next question, the Backend sends the current question and answer options to Player devices. Players submit their answers directly from this screen. This approach ensures synchronized gameplay, controlled pacing, and fairness during live events.

**Support, Guides, and Account Pages:**

- **Support Page:** Provides a form for reporting issues, contacting staff, and accessing FAQ resources. This page helps users resolve problems quickly.

- **Guides Page:** Contains step-by-step instructions for both Hosts and Players, including how to create games, join games, and understand scoring. These guides may include diagrams or mockup-rendered examples for clarity.

- **Account Page:** Allows users to view and edit profile information, manage authentication settings, and review previously played or created games. Hosts can access their saved game sets from this page, while Players can review past results.

Overall, the UI design ensures clear navigation, role-appropriate workflows, and accessibility across both iOS and Android devices.

## 4.5 PERFORMANCE

Performance for Quizically focuses on low latency, high availability, and scalability to support real-time trivia games for BarBar.

| Expectation | Performance Metric | Resolution (I/A) |
|---|---|---|
| Game Uptime | 99% | N/A |
| Concurrency | 100 users | Load balancing? |
| Infinite Loop Avoidance | 30 seconds recovery time | Issue timeout after recovery time maxes out, allow players to re-join |
| Acceptable Server Response Time | Time to first byte (TTFB) less than 1 second | N/A |
| Software Availability | 24/7 | Informed scheduled maintenance |

*Figure 4.5 depicts a table outlining the various performance expectations for Quizically, convering: speed, maintenance, latency, scalability and availability. It offers Team 10's implemented resolutions where necessary.*

## 4.6 SECTION 508 COMPLIANCE

Quizically will meet Section 508 requirements by ensuring that the app is accessible to all users, including those with disabilities. This includes making accommodations for users with motor, visual, and auditory impairments in accordance with WCAG 2.1 AA requirements.

- Accessibility Features: The app will support users with visual, auditory, and motor impairments by providing:
    - Screen reader compatibility, the application will work with JAWS and NVDA, to ensure that blind or visually impaired users can navigate and interact with the app. This tool will verbally explain and read what is on the screen to assist the visually impaired to participate in the quiz games.

  Text-to-speech function for images will also be provided to read the content aloud.
  - Text resizing, users will be able to resize the text without breaking the layout, this allows for users with visual impairments to read the content. This tool allows any user to increase their text size to allow someone with visual impairments to read what is on the screen.
  - High contrast design for better readability, the app will be tested using the color contrast analyzer to ensure it meets accessibility standards. This tool, similar to the text resizing tool, will allow any user to increase the contrast of the screen to help differentiate between icons and colors.

- Standards and Guidelines:
  - WCAG 2.1 AA- Web Content Accessibility Guidelines, ensures that the content is accessible to people with disabilities.
  - Section 508: The app complies with Section 508 of the Rehabilitation Act, which requires that electronic information and communication technology is accessible to people with disabilities.

- Accessibility Testing:
  - Automated Testing: The application will undergo continuous automated accessibility testing using AXE, an open source accessibility testing tool used to identify issues in web applications.
  - Manual Testing: Using screen readers such as, JAWS and NVDA, to ensure that the platform is fully navigable by users with visual or motor impairments.
  - Color Contrast Evaluation: Using color contrast analyzer to ensure readability for users with low vision or color blindness.

## 5 Product Design Specification APPROVAL

The undersigned acknowledge they have reviewed the Quizically Product Design Specification document and agree with the approach it presents. Any changes to this Requirements Definition will be coordinated with and approved by the undersigned or their designated representatives.

| | | | |
|---|---|---|---|
| Signature: | Khadija Warraich | Date: | 10/10/2025 |
| Print Name: | Khadija Warraich | | |
| Title: | Project Manager | | |
| Role: | Student | | |

| | | | |
|---|---|---|---|
| Signature: | Cecilia Newell | Date: | 10/10/2025 |
| Print Name: | Cecilia Newell | | |
| Title: | Sponsor | | |
| Role: | Professor | | |