Quiz Game: Web Application

Jasmine K Sanders

Sullivan University

CSC272: Principles of System Design

Prof. Cordle

February 08, 2026

Quiz Game: Web Application

The Quiz Game Web Application is an interactive, real-time platform designed to facilitate trivia sessions for educational and casual/social purposes. Unlike traditional static quizzes, this application leverages modern web technologies to provide a synchronized experience where multiple participants can compete simultaneously. The project focuses on a seamless transition from session creation by a host to the final declaration of a winner, ensuring that the competitive nature of a live quiz is maintained through low-latency updates, simple user experience, and an automated leaderboard system.

Objectives

The primary objective of this project is to develop a functional web-based quiz game using HTML, CSS, and JavaScript that supports live, concurrent sessions. Key goals include the implementation of a robust session management system, real-time question delivery, and a dynamic scoring engine that updates a leaderboard for all participants. Additionally, the project aims to demonstrate high standards of UI/UX design, including accessibility features such as high color contrast and screen reader compatibility, ensuring that the application is usable by a diverse audience.

Overview

- Project Title: Quiz Game: Web Application
- Project Team Member: Jasmine Sanders (Developer)
- Customer Name: Administrators, Teachers, and Casual Hosts
- Project Manager: Prof. Scott Cordle

System Description

The system is built on a client-server architecture model utilizing Firebase as a backend-as-a-service (BaaS) to handle real-time data synchronization. When a host creates a session, a unique join code is generated and stored in the database. As players join using this code, the system establishes a listener that pushes questions to their devices simultaneously. The frontend, constructed with JavaScript, processes user inputs and sends scoring data back to Firebase, which then broadcasts the updated rankings to all connected clients, culminating in a final winner declaration.

System Requirements

The system requires a modern web browser with JavaScript enabled. Functionally, the application must be able to handle session initialization, fetch questions from a predefined bank, and enforce submission rules per question. From a performance standpoint, the system is designed to minimize latency to ensure that the "live" feel of the game is not disrupted. Reliability requirements include session validation to prevent unauthorized users from joining or altering scores via the browser console.

Major Feature Specifications

- Dynamic Role-Based Interface: A unified single-page application that branches into distinct host and player workflows upon launch.
- Real-Time Live Lobby: A centralized waiting room where player names appear as "chips" in real-time as they join using the generated 4-digit pin.
- Live Response Analytics: A proprietary bar chart visualization for the host that updates instantly as players submit answers, showing the distribution of responses across the four color categories (red, blue, yellow, green).

- Competitive Scoring Engine: A time-sensitive algorithm that rewards both accuracy and speed.
- Visual Feedback System: Real-time feedback messages for players ("Correct!" or "Incorrect!") along with color-coded input borders for the creator to ensure high usability during quiz setup.

System Diagrams

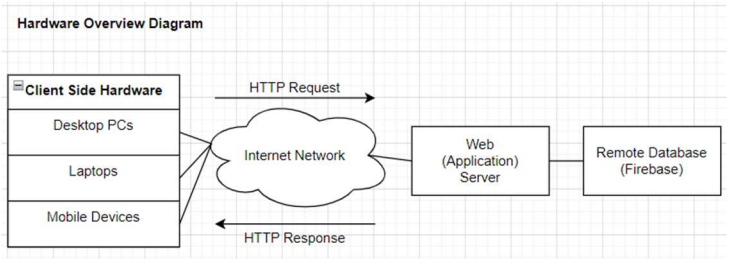
The system architecture is visualized through a Software Overview Diagram (see Appendix B, Figure 2) and a Hardware Overview Diagram (see Appendix B, Figure 1). The software diagram illustrates the flow of data from the user interface (HTML/CSS) through the controller (JavaScript) to the persistence layer (Firebase). The hardware diagram shows the interaction between the end-user devices and the cloud-based servers hosting the application and database. These diagrams represent a centralized hub-and-spoke model where the Firebase database serves as the central point of information for content distributed to users.

Hardware and Software Overview Diagrams

In the Hardware Overview Diagram, the system consists of client-side hardware (personal computers or mobile devices) communicating over HTTPS to cloud servers. There is no requirement for specialized local server hardware, as the infrastructure is handled by the cloud provider.

Figure 1 (see Appendix B)

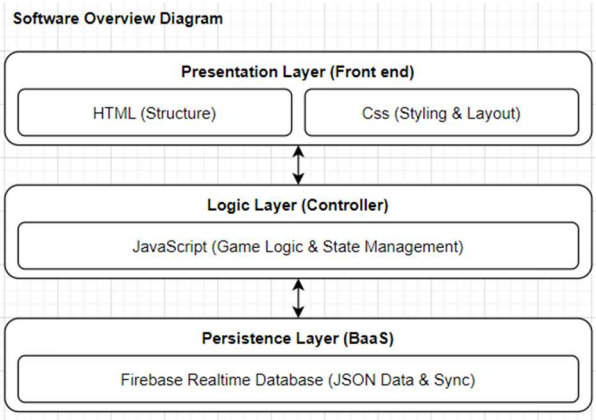
Hardware Overview Diagram



In the Software Overview Diagram, the stack is divided into a three-tier architecture: the presentaiton layer, the logic layer, and the persistence layer. The presentation layer is constructed using HTML for structural semantics and CSS for visual styling. The logic layer is the “brain” of the web application. Written in JavaScript, it manages the gameState object, controls the 10-second timers using setInterval, and handles event listeners for user inputs. The persistence layer utilizes the Firebase SDK to maintina a persistent conneciton to the realtime database. It uses a JSON-based structure where the sessions node stores the live question index, the players node stores scores, and the responses node tracks real-time votes for the host's chart.

Figure 2 (see Appendix B)

Software Overview Diagram



Economical, Technical, and Time Constraints

This project was developed under a strict five-week timeline, requiring efficient task management and prioritization of core features. Economically, the project utilized free-tier open-source tools and hosting services to remain cost-effective. Technically, the primary challenge was implementing real-time synchronization; while initially attempted with manual management, the technical constraint of ensuring all players saw the same question at the same time

necessitated the integration of Firebase, which provided the necessary backend infrastructure within the project's technical scope.

Hardware Detailed Implementation

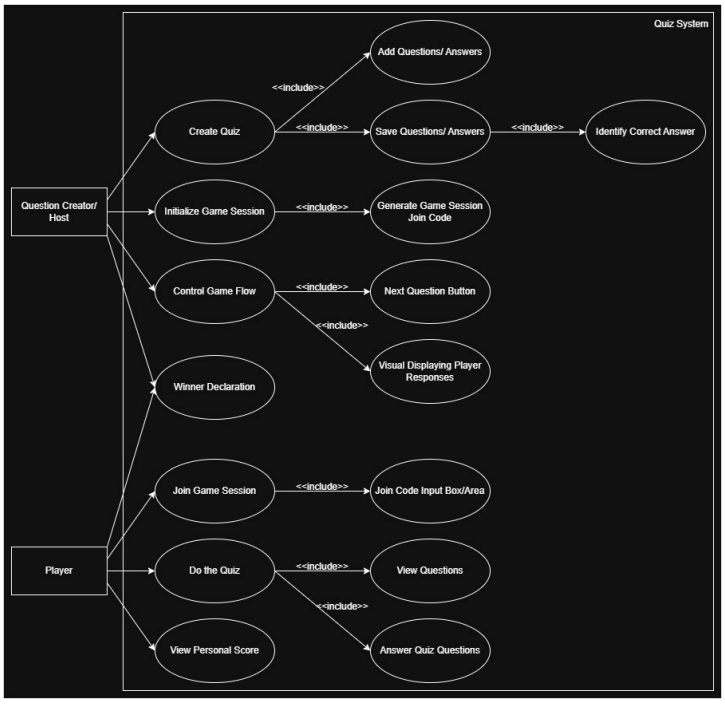
The hardware required for this project includes a development machine (laptop or PC) with an internet connection. For the end-users, any device capable of running a standard web browser is sufficient. This includes smartphones, tablets, and desktop computers. The project does not require high-performance hardware, as the heavy lifting of data synchronization is handled by cloud-based servers, making the application highly accessible to users with older or lower-spec devices.

Software Detailed Implementation

The software architecture of the Quiz Game is designed where each technical component directly supports the functional requirements illustrated in the project's Use Case Diagram.

Figure 3 (see Appendix B)

Use Case Diagram



The frontend was implemented using HTML for structure, CSS for layout and UI theme, and JavaScript for logic. More specifically, HTML was used to structure the interactive interfaces required for the Host to create quizzes, specifically facilitating the "Add Questions/Answers" and "Save Questions/Answers" use cases. It also provides the critical input area for players to join a game session via a unique join code. To maintain a professional and consistent UI, CSS is applied to create a glassmorphism layout that enhances usability for visual elements, such as the real-time display of player responses during active game sessions. JavaScript was used for managing the real-time functionality required for game flow control and answer validation. It implements the "Identify Correct Answer" use case by comparing player inputs against stored data and calculates competitive scores based on a speed-sensitive formula: Score = 50 + (time remaining*100) for correct answers. This logic also powers the "Next Question" functionality and handles the automated transition to the final winner declaration screen by comparing all participant scores at the session's conclusion.

Firebase was integrated as the backend software to handle the realtime database and user role permissions. More specifically, it provides the backend persistence and real-time synchronization necessary for the live competitive environment. It satisfies the "Initialize Game Session" use case by generating and storing unique 4-digit join codes while simultaneously managing the real-time broadcast of questions to all connected player devices. Furthermore, Firebase handles specific user role permissions to ensure that only the Host can control the game flow, while maintaining a central record of scores so that players can accurately "View Personal Score" in real-time.

The development process was centered around Visual Studio Code as the primary IDE and GitHub for version control, which ensured a clear history of each feature's implementation.

Final testing was conducted using browser developer tools and simulated environments to ensure the application performed correctly across various operating systems and browser engines. This comprehensive approach ensures that every use case - from the initial creation of a quiz by the host to the final leaderboard display for the players - is technically supported and reliably executed.

Test/Evaluation Experimental Procedure and Analysis of Results

Testing was conducted in phases, starting with unit tests for the scoring logic and moving to integration testing for the Firebase synchronization. I performed load testing by simulating multiple concurrent sessions to observe latency and data consistency. A significant barrier identified during testing was the difficulty of managing user roles manually. This was resolved by utilizing Firebase for database needs. The scoring accuracy was tested by comparing manual calculations to displayed scores with the formula: Score = 50 + (time remaining*100) for correct answers. Additionally, during testing there were latency concerns. Using the Firebase transaction method for response counts, we successfully eliminated "race conditions" where two players submitting at the exact same millisecond might have previously caused the host's bar chart to miscount. Altogether, the final evaluation showed that the scoring and winner declaration was accurate across all devices and the UI remained responsive under simulated stress.

Societal Impact including Legal and Ethical Considerations

The Quiz Game Web Application provides a significant benefit to general education by offering a free, engaging tool for assessment. Ethically, the project prioritizes user privacy by not requiring extensive personal data to join a game. Legal considerations include adherence to accessibility standards (WCAG), ensuring that users with visual or motor impairments can still

participate. Additoanlly, the project addresses digital equity by ensuring the application remains lightweight and functional on low-bandwidth connections.

Conclusions

The development of the Quiz Game Web Application successfully demonstrated the principles of system design within a compressed timeframe. By utilizing a modular approach and integrating cloud-based services, the project met all its functional requirements, including live session management and accurate winner declaration. The transition from a static design to a real-time synchronized system was the most significant achievement, providing a solid foundation for a scalable trivia platform.

Recommendations for Future Work

Future iterations of this project could include a custom "Question Builder" interface where hosts can upload their own question sets via CSV or JSON files. Additionally, integrating system to create and save user profiles would provide a more secure and personalized experience for returning hosts.

References

Coyier, C. (2025, December 17). *CSS Flexbox Layout Guide*. CSS-Tricks. https://css-tricks.com/snippets/css/a-guide-to-flexbox/

Design and user experience : web.dev. web.dev. (n.d.-a). https://web.dev/learn/accessibility/design-ux?continue=https%3A%2F%2Fweb.dev%2Flearn%2Faccessibility%23article-https%3A%2F%2Fweb.dev%2Flearn%2Faccessibility%2Fdesign-ux

Design and user experience : web.dev. web.dev. (n.d.-b). https://web.dev/learn/accessibility/design-ux?continue=https%3A%2F%2Fweb.dev%2Flearn%2Faccessibility%23article-https%3A%2F%2Fweb.dev%2Flearn%2Faccessibility%2Fdesign-ux

GeeksforGeeks. (2025a, July 23). *Create a quiz app with timer using HTML CSS and JavaScript*. https://www.geeksforgeeks.org/javascript/create-a-quiz-app-with-timer-using-html-css-and-javascript/

GeeksforGeeks. (2025b, July 27). *Create a quiz application using JavaScript*. https://www.geeksforgeeks.org/javascript/how-to-create-a-simple-javascript-quiz/

Google. (n.d.). *Firebase realtime database*. Firebase Realtime Database. https://firebase.google.com/docs/database#:~:text=Firebase%20Realtime%20Database,Where%20do%20I%20start?

Graham, G. (2026, January 27). *CSS Grid Layout Guide: CSS-tricks*. CSS-Tricks. https://css-tricks.com/css-grid-layout-guide/

How to Become a Developer. (2023, July 2). *Create a Quiz App using HTML CSS & JavaScript | Quiz Web App using JavaScript*. YouTube. https://www.youtube.com/watch?v=MgeQa7qXIwI

web.dev. (n.d.-c). *Color and contrast*. web.dev. https://web.dev/learn/accessibility/color-contrast?continue=https%3A%2F%2Fweb.dev%2Flearn%2Faccessibility%23article-https%3A%2F%2Fweb.dev%2Flearn%2Faccessibility%2Fcolor-contrast

web.dev. (n.d.-d). *Typography*. web.dev. https://web.dev/learn/accessibility/typography?continue=https%3A%2F%2Fweb.dev%2Flearn%2Faccessibility%23article-https%3A%2F%2Fweb.dev%2Flearn%2Faccessibility%2Ftypography

Appendix A

User Instructions and Awknowledgements

Software Installation Instructions

As a web application, no traditional installation is required for the end-user. The application is deployed via GitHub Pages or a similar web hosting service. To run the project locally, a user must clone the GitHub repository, navigate to the project folder, and open the index.html file in any modern web browser.

User's Manual

1. Getting Started

Navigate to the application URL. On the landing screen, select your role: "I am a Player" or "I am the Host".

2. For the Host (Quiz Creator)

- Create Questions: Enter your question and four possible answers.
- Set Correct Answer: Use the dropdown menu to select which color (red, blue, yellow, or green) is the correct one.
- Launch: Click "+ Add Question" to save. Once your bank is ready, click "Launch Quiz".
- The Lobby: A 4-digit game pin will appear. Display this to your players. Once their nicknames appear on the screen, click "Start Game".
- Managing Play: Watch the live bar chart to see how many people have answered. Click "Next Question" to advance the group when the timer expires.

3. For the Player

- Joining: Enter a nickname and the 4-digit pin provided by the host.

- Answering: When a question appears on the host's screen, your device will show four colored buttons.
- Speed Matters: Tap the color that matches the correct answer. The faster you tap, the higher your score multiplier will be.
- Results: After the final question, the winner declaration will display a crown icon and the name of the top-scoring player.

Acknowledgments

Special thanks to Prof. Scott Cordle for project management guidance and to the MDN Web Docs community for providing comprehensive documentation on JavaScript and CSS standards. I would also like to acknowledge the Firebase Google Cloud team for providing the real-time database infrastructure that made the 4-digit PIN synchronization possible, and the CSS-Tricks Flexbox Guide for the layout logic used in the host's response chart.

Appendix B

Figures Referenced in the Document

Figure 1

Hardware Overview Diagram

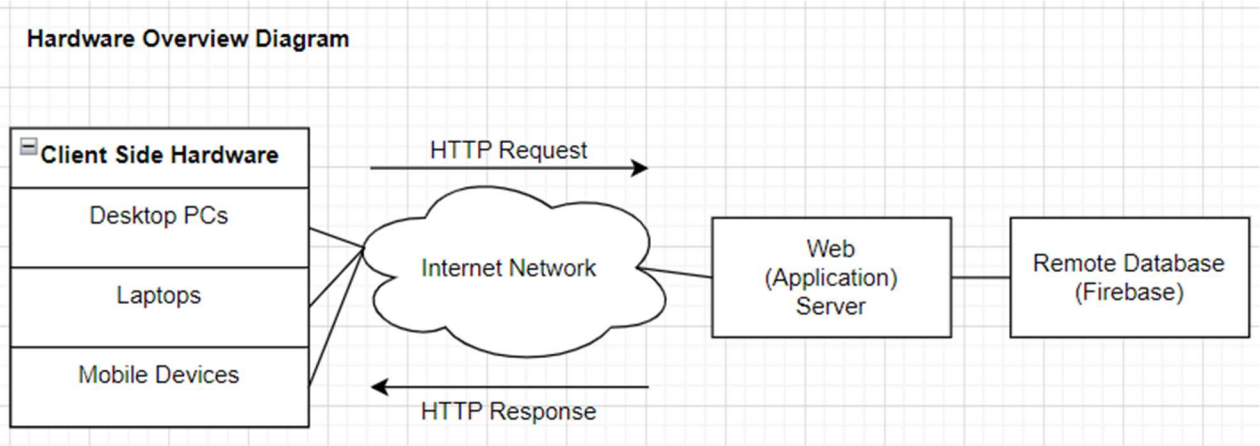


Figure 2

Software Overview Diagram

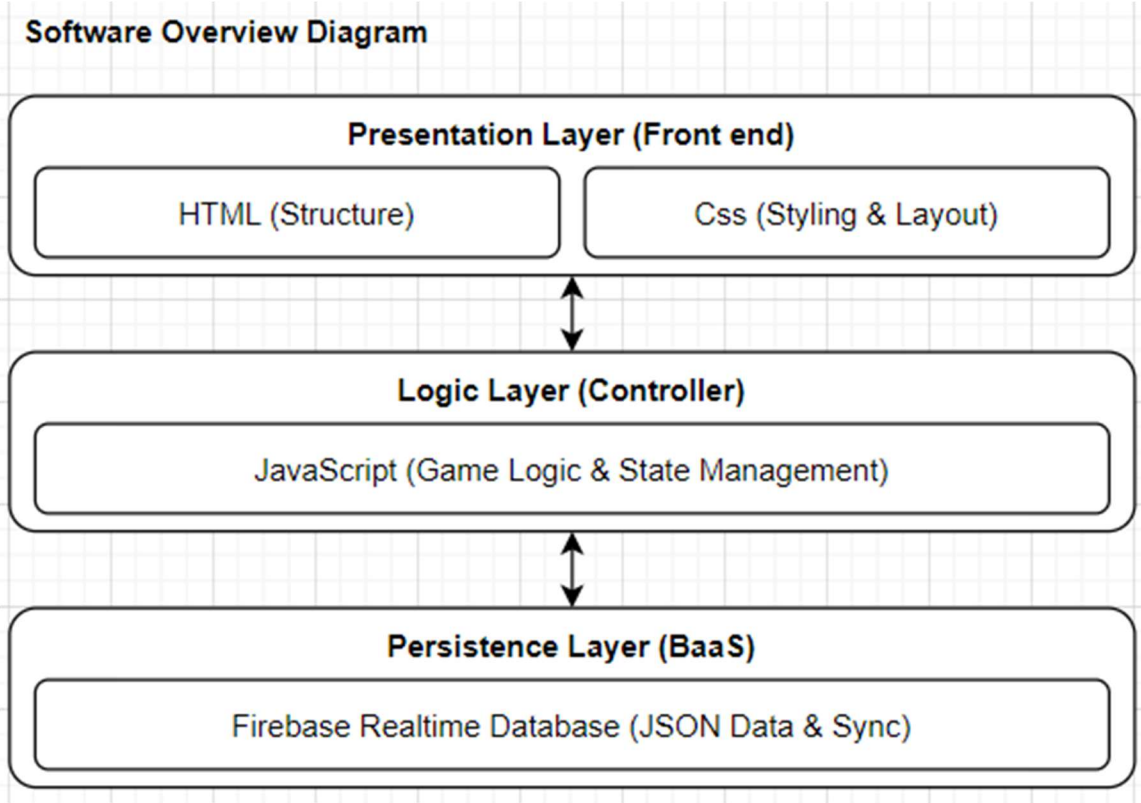
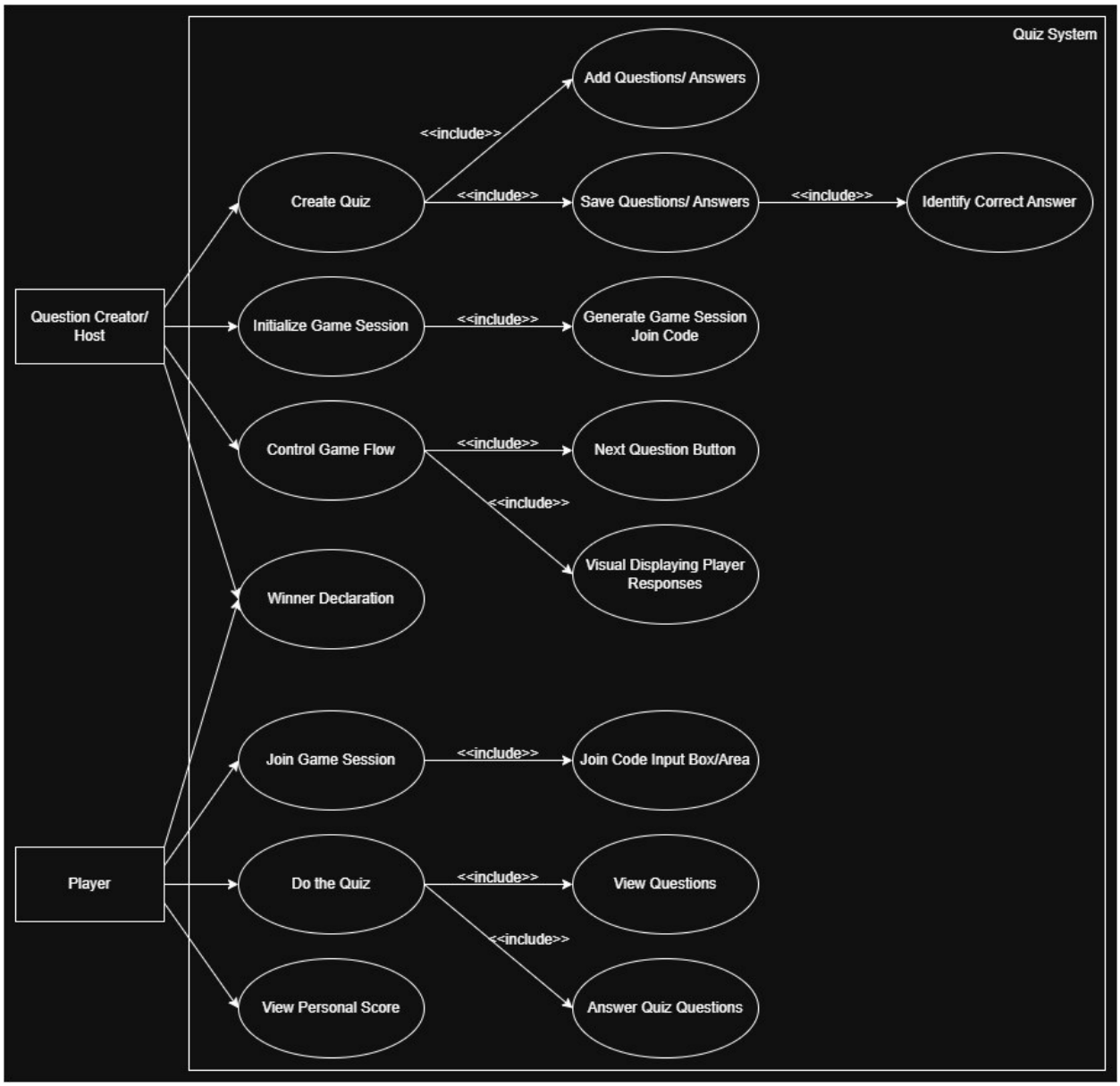


Figure 3

Use Case Diagram



Appendix C

Source Code Files

CSS File Source Code

```
:root {
    --red: #e63946;
    --blue: #457b9d;
    --yellow: #e9c46a;
    --green: #2a9d8f;
    --purple: #6c5ce7;
    --dark-bg: #1a1a2e;
    --glass: rgba(255, 255, 255, 0.98);
    --text-main: #333;
}

body {
    background: linear-gradient(135deg, var(--dark-bg), #16213e);
    font-family: 'Segoe UI', system-ui, sans-serif;
    margin: 0;
    min-height: 100vh;
    display: flex;
    justify-content: center;
    align-items: center;
    color: var(--text-main);
```

```
}

.container { width: 95%; max-width: 600px; padding: 10px; }

.card {
    background: var(--glass);
    padding: 2rem;
    border-radius: 20px;
    box-shadow: 0 8px 32px rgba(0,0,0,0.3);
    text-align: center;
    transition: transform 0.3s;
}

.hidden { display: none !important; }
.sr-only { position: absolute; width: 1px; height: 1px; padding: 0; overflow: hidden; clip:
rect(0,0,0,0); border: 0; }

/* Typography */
h1, h2 { color: var(--purple); margin-top: 0; }
.logo-text { font-size: 3rem; margin-bottom: 0.5rem; }
.big-pin { font-size: 4rem; letter-spacing: 5px; margin: 10px 0; color: var(--text-main); }

/* Inputs */
```

```
input, select {
    width: 100%;
    padding: 15px;
    margin: 10px 0;
    border: 2px solid #ccc;
    border-radius: 10px;
    font-size: 1.1rem;
    box-sizing: border-box;
}
input:focus { border-color: var(--purple); outline: none; }

/* Buttons */
button { font-family: inherit; }
.btn-primary {
    background: var(--purple); color: white; width: 100%; padding: 15px;
    border: none; border-radius: 12px; font-weight: bold; cursor: pointer;
    font-size: 1.2rem; margin-top: 10px; transition: filter 0.2s;
}
.btn-primary:hover { filter: brightness(1.1); }
.btn-secondary {
    background: white; color: var(--purple); width: 100%; padding: 13px;
    border: 2px solid var(--purple); border-radius: 12px; cursor: pointer;
    font-weight: bold; margin-top: 10px;
```

```
}
.btn-text { background: none; border: none; color: #666; cursor: pointer; margin-top: 10px; text-decoration: underline; }

/* Chart Visualization */
.chart-container {
    display: flex;
    justify-content: space-around;
    align-items: flex-end;
    height: 200px;
    margin: 20px 0;
    background: #f4f4f4;
    border-radius: 10px;
    padding: 10px;
}
.bar-group { display: flex; flex-direction: column; align-items: center; width: 18%; height: 100%; justify-content: flex-end; }
.bar {
    width: 100%;
    transition: height 0.5s ease;
    border-radius: 5px 5px 0 0;
    color: white;
    font-weight: bold;
```

```
    display: flex;
    align-items: flex-end;
    justify-content: center;
    padding-bottom: 5px;
    min-height: 0; /* Important for flex */
}
.bar-label { font-size: 0.8rem; margin-top: 5px; font-weight: bold; }

/* Game Elements */
.answer-grid { display: grid; grid-template-columns: 1fr 1fr; gap: 15px; margin-top: 20px; }
.answer-opt {
    padding: 30px 10px; border: none; border-radius: 12px;
    font-weight: bold; font-size: 1.2rem; cursor: pointer; color: white;
    box-shadow: 0 4px 6px rgba(0,0,0,0.1);
}
.answer-opt:disabled { opacity: 0.5; cursor: not-allowed; }

/* Colors */
.opt-0 { background-color: var(--red); }
.opt-1 { background-color: var(--blue); }
.opt-2 { background-color: var(--yellow); }
.opt-3 { background-color: var(--green); }
```

```
/* Input Colors for Creator */
.input-red { border-left: 5px solid var(--red); }
.input-blue { border-left: 5px solid var(--blue); }
.input-yellow { border-left: 5px solid var(--yellow); }
.input-green { border-left: 5px solid var(--green); }

/* Lobby List */
.lobby-player-list {
    display: flex; flex-wrap: wrap; justify-content: center; gap: 10px;
    margin: 20px 0; min-height: 50px;
}
.player-chip { background: #eee; padding: 5px 15px; border-radius: 20px; font-weight: bold;
animation: popIn 0.3s; }

@keyframes popIn {
    from { transform: scale(0); }
    to { transform: scale(1); }
}

/* Timer */
.timer-badge { background: var(--text-main); color: white; padding: 5px 15px; border-radius:
20px; font-weight: bold; }
```

HTML File Source Code

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Quiz Game</title>
    <link rel="stylesheet" href="index-styles.css">
    <script src="https://www.gstatic.com/firebasejs/9.6.1/firebase-app-compat.js"></script>
    <script src="https://www.gstatic.com/firebasejs/9.6.1/firebase-database-compat.js"></script>
    <script src="https://www.gstatic.com/firebasejs/9.6.1/firebase-auth-compat.js"></script>
    </head>
<body>

    <div id="app" class="container">

        <section id="landing-screen" class="screen card">
            <h1 class="logo-text">Quiz Game</h1>
            <p>Select your role to begin:</p>
            <div class="action-buttons">
                <button class="btn-primary" onclick="setupJoinView()">I am a Player</button>
                <button class="btn-secondary" onclick="showScreen('editor-screen')">I am the
Host</button>
```

```
    </div>
</section>

<section id="join-screen" class="screen card hidden">
    <h2>Join Session</h2>
    <div class="join-section">
        <label for="player-name" class="sr-only">Display Name</label>
        <input type="text" id="player-name" placeholder="Enter your Nickname">

        <label for="join-code" class="sr-only">Game PIN</label>
        <input type="number" id="join-code" placeholder="Game PIN">

        <button class="btn-primary" onclick="initSession('player')">Enter Game</button>
        <button class="btn-text" onclick="location.reload()">Back</button>
    </div>
</section>

<section id="editor-screen" class="screen card hidden">
    <h2>Create Quiz</h2>
    <div class="input-group">
        <input type="text" id="q-input" placeholder="Type your question here...">
    </div>
    <div class="editor-grid">
```

```
            <input type="text" id="opt-0-in" placeholder="Red Option" class="input-red">
            <input type="text" id="opt-1-in" placeholder="Blue Option" class="input-blue">
            <input type="text" id="opt-2-in" placeholder="Yellow Option" class="input-yellow">
            <input type="text" id="opt-3-in" placeholder="Green Option" class="input-green">
        </div>
        <label for="correct-opt">Select Correct Answer:</label>
        <select id="correct-opt">
            <option value="0">Red Option</option>
            <option value="1">Blue Option</option>
            <option value="2">Yellow Option</option>
            <option value="3">Green Option</option>
        </select>
        <div class="action-buttons">
            <button class="btn-secondary" onclick="saveQuestion()">+ Add Question</button>
            <button class="btn-primary" id="start-session-btn" onclick="initSession('creator')"
style="display:none;">Launch Quiz</button>
        </div>
    </section>

    <section id="host-lobby-screen" class="screen card hidden">
        <p class="subtitle">Join at <strong>quizlive.web.app</strong> using PIN:</p>
        <h1 id="lobby-pin-display" class="big-pin">---</h1>
        <div class="lobby-player-list" id="lobby-players">
```

```
        <p>Waiting for players...</p>
    </div>
    <button class="btn-primary" onclick="startActualGame()">Start Game</button>
</section>

<section id="host-game-screen" class="screen card hidden">
    <div class="host-header">
        <span id="host-q-counter">Q: 1/5</span>
        <span class="timer-badge" id="host-timer">10</span>
    </div>
    <h2 id="host-question-text">Question Loading...</h2>

    <div class="chart-container">
        <div class="bar-group">
            <div id="bar-0" class="bar opt-0">0</div>
            <span class="bar-label">Red</span>
        </div>
        <div class="bar-group">
            <div id="bar-1" class="bar opt-1">0</div>
            <span class="bar-label">Blue</span>
        </div>
        <div class="bar-group">
            <div id="bar-2" class="bar opt-2">0</div>
```

```
            <span class="bar-label">Yel</span>
          </div>
          <div class="bar-group">
            <div id="bar-3" class="bar opt-3">0</div>
            <span class="bar-label">Grn</span>
          </div>
        </div>

        <button class="btn-primary" id="next-q-btn" onclick="nextQuestion()">Next
Question</button>
      </section>

      <section id="player-game-screen" class="screen card hidden">
        <div class="player-header">
          <div id="player-score-display">Score: 0</div>
          <div id="player-timer-circle" class="timer-small">10</div>
        </div>

        <div id="waiting-msg" class="hidden">
          <h3>Look at the host screen!</h3>
          <div class="loader"></div>
        </div>
```

```
            <div id="answer-grid" class="answer-grid">
                </div>
            <p id="feedback-msg"></p>
        </section>

        <section id="result-screen" class="screen card hidden">
            <h2>🏆 Winner 🏆</h2>
            <div id="winner-display">
                </div>
            <button class="btn-primary" onclick="location.reload()">Play Again</button>
        </section>

    </div>

    <script src="script.js"></script>
</body>
</html>
```

JavaScript File Source Code

```
// 1. Firebase Config (Using provided keys)
const firebaseConfig = {
    apiKey: "AIzaSyDy6NACds1W1t-JKgII9nbeM8pvFIIiRgg",
    databaseURL: "https://quizgamewebapp-default-rtdb.firebaseio.com/",
```

```
    projectId: "quizgamewebapp",
};
firebase.initializeApp(firebaseConfig);
const db = firebase.database();

// 2. Global State
let questions = [];
let gameState = {
    role: null, // 'creator' or 'player'
    sessionId: null,
    currentQuestionIndex: -1,
    score: 0,
    timer: 10,
    timerId: null,
    playerName: ""
};

/** UI NAVIGATION */
function showScreen(screenId) {
    document.querySelectorAll('.screen').forEach(s => s.classList.add('hidden'));
    document.getElementById(screenId).classList.remove('hidden');
}
```

```
function setupJoinView() {
    showScreen('join-screen');
}

/** CREATOR: QUESTION MANAGEMENT */
function saveQuestion() {
    const qText = document.getElementById('q-input').value;
    const opts = [
        document.getElementById('opt-0-in').value,
        document.getElementById('opt-1-in').value,
        document.getElementById('opt-2-in').value,
        document.getElementById('opt-3-in').value
    ];

    if(!qText || !opts[0] || !opts[1]) {
        alert("Enter a question and at least two options.");
        return;
    }

    questions.push({
        q: qText,
        a: opts,
        correct: parseInt(document.getElementById('correct-opt').value)
```

```
    });

    // Reset UI
    document.getElementById('q-input').value = "";
    document.querySelectorAll('.editor-grid input').forEach(i => i.value = "");
    document.getElementById('start-session-btn').style.display = "block";
    alert(`Saved! Total questions: ${questions.length}`);
}

/** SESSION INITIALIZATION */
async function initSession(role) {
    gameState.role = role;

    // Auth Check
    await firebase.auth().signInAnonymously();

    if (role === 'creator') {
        if(questions.length === 0) return alert("Create questions first!");

        // Generate a random 4-digit PIN
        gameState.sessionId = Math.floor(1000 + Math.random() * 9000).toString();

        // Create Session in DB
```

```
    await db.ref(`sessions/${gameState.sessionId}`).set({
        status: "lobby", // New state: Lobby
        questions: questions,
        currentQuestion: -1,
        timestamp: Date.now()
    });

    // Show Lobby
    showScreen('host-lobby-screen');
    document.getElementById('lobby-pin-display').innerText = gameState.sessionId;

    // Listen for players joining in real-time
    listenForPlayers();

} else {
    // PLAYER FLOW
    gameState.playerName = document.getElementById('player-name').value;
    gameState.sessionId = document.getElementById('join-code').value;

    if(!gameState.playerName || !gameState.sessionId) return alert("Name and PIN required.");

    // Check if session exists
    const sessionRef = db.ref(`sessions/${gameState.sessionId}`);
```

```
        const snapshot = await sessionRef.once('value');

        if(!snapshot.exists()) return alert("Invalid PIN");

        // Add player to lobby
        await db.ref(`sessions/${gameState.sessionId}/players/${gameState.playerName}`).set({
            score: 0
        });

        showScreen('player-game-screen');
        document.getElementById('answer-grid').innerHTML = "<h3>Waiting for host to
start...</h3>";

        // Start listening to game state
        listenToGame();
    }
}

/** HOST: LOBBY LOGIC */
function listenForPlayers() {
    db.ref(`sessions/${gameState.sessionId}/players`).on('value', snapshot => {
        const list = document.getElementById('lobby-players');
        list.innerHTML = "";
```

```
        snapshot.forEach(child => {
            list.innerHTML += `<span class="player-chip">${child.key}</span>`;
        });
    });
}

function startActualGame() {
    // Transition from Lobby to Active Game
    db.ref(`sessions/${gameState.sessionId}`).update({
        status: "active",
        currentQuestion: 0 // Start Q1
    });
    showScreen('host-game-screen');
    listenForHostLiveUpdates();
    loadHostQuestion(0);
}

/** HOST: LIVE GAME CONTROL */
function listenForHostLiveUpdates() {
    // Listen for responses to update Bar Chart
    db.ref(`sessions/${gameState.sessionId}/responses`).on('value', snapshot => {
        const data = snapshot.val() || {};
        const total = Object.values(data).reduce((a,b) => a+b, 0); // Total votes
```

```
        for(let i=0; i<4; i++) {
            const count = data[i] || 0;
            const bar = document.getElementById(`bar-${i}`);
            // Calculate percentage for height (Max 100%)
            const pct = total > 0 ? (count / total) * 100 : 0;

            bar.style.height = `${pct}%`;
            bar.innerText = count;
        }
    });
}

function loadHostQuestion(index) {
    if(!questions[index]) return finishGame();

    gameState.currentQuestionIndex = index;
    const q = questions[index];

    document.getElementById('host-question-text').innerText = q.q;
    document.getElementById('host-q-counter').innerText = `Q: ${index +
1}/${questions.length}`;
```

```
    // Reset Chart
    [0,1,2,3].forEach(i => {
        document.getElementById(`bar-${i}`).style.height = '0%';
        document.getElementById(`bar-${i}`).innerText = '0';
    });

    // Reset DB responses for this question
    db.ref(`sessions/${gameState.sessionId}/responses`).remove();

    runTimer('host-timer', () => {
        // Optional: Reveal answer on host screen automatically
    });
}

function nextQuestion() {
    const nextIdx = gameState.currentQuestionIndex + 1;
    if(nextIdx >= questions.length) {
        finishGame();
    } else {
        db.ref(`sessions/${gameState.sessionId}`).update({ currentQuestion: nextIdx });
        loadHostQuestion(nextIdx);
    }
}
```

```
/** PLAYER: GAME LOGIC */
function listenToGame() {
    db.ref(`sessions/${gameState.sessionId}`).on('value', snapshot => {
        const data = snapshot.val();
        if(!data) return;

        // 1. Check for Game Over
        if(data.status === "finished") {
            showResults(data.winner);
            return;
        }

        // 2. Load New Question
        if(data.currentQuestion !== undefined && data.currentQuestion !==
gameState.currentQuestionIndex) {
            gameState.currentQuestionIndex = data.currentQuestion;
            questions = data.questions; // Sync questions
            renderPlayerQuestion(data.questions[data.currentQuestion]);
        }
    });
}
```

```
function renderPlayerQuestion(qData) {
    const grid = document.getElementById('answer-grid');
    grid.innerHTML = '';
    document.getElementById('feedback-msg').innerText = "";

    // Show waiting animation logic could go here if using "Show Question" vs "Show Answers"
states

    qData.a.forEach((opt, idx) => {
        if(opt !== "---") { // Filter empty options
            const btn = document.createElement('button');
            btn.className = `answer-opt opt-${idx}`;
            btn.innerText = opt; // In Kahoot usually shapes, but here text
            btn.onclick = () => submitAnswer(idx, qData.correct);
            grid.appendChild(btn);
        }
    });

    runTimer('player-timer-circle', () => {
        document.querySelectorAll('.answer-opt').forEach(b => b.disabled = true);
    });
}
```

```
function submitAnswer(idx, correctIdx) {
    // Disable buttons
    document.querySelectorAll('.answer-opt').forEach(b => b.disabled = true);

    const isCorrect = (idx === correctIdx);
    const feedback = document.getElementById('feedback-msg');

    if(isCorrect) {
        // Calculate Score: (Score= 50+ (Time remaining * 100))
        const points = (gameState.timer * 100) + 50;
        gameState.score += points;
        feedback.innerText = "Correct! +" + points;
        feedback.style.color = "var(--green)";

        // Update Score in DB
        db.ref(`sessions/${gameState.sessionId}/players/${gameState.playerName}/score`).set(gam
eState.score);
    } else {
        feedback.innerText = "Incorrect!";
        feedback.style.color = "var(--red)";
    }

    document.getElementById('player-score-display').innerText = `Score: ${gameState.score}`;
```

```
    // Record response count for Host Graph
    db.ref(`sessions/${gameState.sessionId}/responses/${idx}`).transaction(val => (val || 0) + 1);
}

/** SHARED UTILS */
function runTimer(elemId, onFinish) {
    gameState.timer = 10;
    const el = document.getElementById(elemId);
    if(el) el.innerText = 10;

    if(gameState.timerId) clearInterval(gameState.timerId);

    gameState.timerId = setInterval(() => {
        gameState.timer--;
        if(el) el.innerText = gameState.timer;

        if(gameState.timer <= 0) {
            clearInterval(gameState.timerId);
            if(onFinish) onFinish();
        }
    }, 1000);
}
```

```
/** END GAME */
async function finishGame() {
    // HOST calculates winner to ensure single source of truth
    const playersRef = db.ref(`sessions/${gameState.sessionId}/players`);
    const snap = await playersRef.once('value');
    let bestPlayer = "No One";
    let maxScore = -1;

    snap.forEach(p => {
        if(p.val().score > maxScore) {
            maxScore = p.val().score;
            bestPlayer = p.key;
        }
    });

    // Write winner to DB so all players see it
    await db.ref(`sessions/${gameState.sessionId}`).update({
        status: "finished",
        winner: { name: bestPlayer, score: maxScore }
    });

    showResults({ name: bestPlayer, score: maxScore });
```

```
}

function showResults(winnerObj) {
    showScreen('result-screen');
    const container = document.getElementById('winner-display');

    // If passed directly or waiting for DB update
    if(!winnerObj) return;

    container.innerHTML = `
        <div style="font-size: 5rem">👑</div>
        <h1 style="font-size: 3rem">${winnerObj.name || winnerObj}</h1>
        <h3>Score: ${winnerObj.score || 0}</h3>
    `;
}
```