COLLEGE CODE : 9604

COLLEGE NAME : C.S.I INSTITUTE OF TECHNOLOGY

DEPARTMENT : INFORMATION TECHNOLOGY

STUDENT NM ID:

ROLL NO :

DATE : 07/10/2025

SUBMITTED BY,

NAME :

MOBILE NO:

PHASE 5 – PROJECT DEMONSTRATION & DOCUMENTATION
INTERACTIVE QUIZ APP

FINAL DEMO WALKTHROUG:

Quiz App Final Demo Walkthrough Structure

Start strong with a quick **introduction** and a clear **agenda** so your audience knows what to expect.

1. Introduction and Project Overview (1-2 minutes)

- **Hook:** "Welcome! Today, I'm excited to walk you through our final **Quiz Application**."
- **Problem Solved:** Briefly state the purpose. "We designed this app to provide an interactive, easy-to-use platform for testing knowledge on [mention your topic/niche], with a focus on **[mention your key differentiating feature, e.g., real-time scoring/user-generated content/simple UI]**."
- **Tech Stack:** A quick mention of the main technologies used (e.g., React, Node.js, MongoDB, or Android/iOS with Kotlin/Swift).

---

2. User Experience Demo: Taking a Quiz (Focus on Core Functionality)

This is the main part. Walk through the user's journey from start to finish.

A. Entry and Setup

1. **Landing/Home Page:** Showcase the app's clean, professional design.
2. **User Authentication (If applicable):**
    - **Sign-Up/Login:** Demonstrate how quickly a new user can sign up or log in.
    - **Profile/Dashboard:** Show the user's starting point (e.g., score history, available quizzes).
3. **Quiz Selection:** Show the list of available quizzes, categories, or difficulty levels.

B. The Quiz Experience

1. **Start the Quiz:** Click to begin and show the **first question interface**.
    - **UI/UX Feature:** Point out the clear question text, answer options, and any visual timers or progress bars.
2. **Answering Questions:**
    - **Correct Answer:** Select an answer and show the immediate feedback (e.g., a green highlight, a "Correct!" message). Point out how the **score** updates.
    - **Incorrect Answer:** Select a wrong answer and show the feedback (e.g., a red highlight). If your app provides an **explanation** or the correct answer after a wrong guess, make sure to highlight this feature.
3. **Navigation and State:**
    - Show the **Next/Previous** button functionality (if you allow going back).
    - Highlight any special question types (e.g., true/false, fill-in-the-blank, image-based questions).

C. Final Results

1. **Quiz Completion:** Finish the last question.
2. **Score Summary:** Clearly display the final score, percentage, and any time taken.
3. **Call to Action:** Show options like "Review Answers," "Retake Quiz," or "View Leaderboard." Click on the **Leaderboard** (if available) to show the user's new ranking.

---

3. Key Feature Deep Dive (Focus on Your Unique Selling Points)

Use this section to show off the advanced features you spent the most time on.

- **Responsive Design:** (If it's a web app) Quickly switch the browser size to show how the layout adapts for mobile/tablet.
- **Data Persistence:** Briefly show how the user's score/progress is saved (e.g., a high-score list on their profile).
- **Search/Filtering:** Demonstrate any functionality for searching for specific quizzes or filtering by topic.

---

4. Administrator Panel Demo (If applicable)

This shows the power of your back-end.

1. **Admin Login:** Log in as an administrator.
2. **Dashboard Overview:** Show a summary of key metrics (total users, total quizzes, top scores, etc.).
3. **CRUD Operations (Create, Read, Update, Delete):**
    - **Create Question:** Quickly walk through the process of adding a *new question*—showing the form where the admin inputs the question, choices, and designates the correct answer.
    - **Manage Quizzes:** Show how an admin can easily edit the title or status of an existing quiz.
    - **User Management:** (Optional) Show a table of users and the ability to view or manage their accounts.

---

5. Conclusion and Next Steps

- **Recap:** "To summarize, our quiz app delivers **[Feature 1]**, **[Feature 2]**, and provides a seamless **[Feature 3]** experience."
- **Future Work:** Mention one or two planned features you didn't have time to implement (e.g., integrating social sharing, adding more question media types, or gamification badges). This shows you're thinking ahead.
- **Q&A:** "Thank you. I'm now open for any questions about the features, the code, or the deployment process."

Pro-Tip for Demo Success

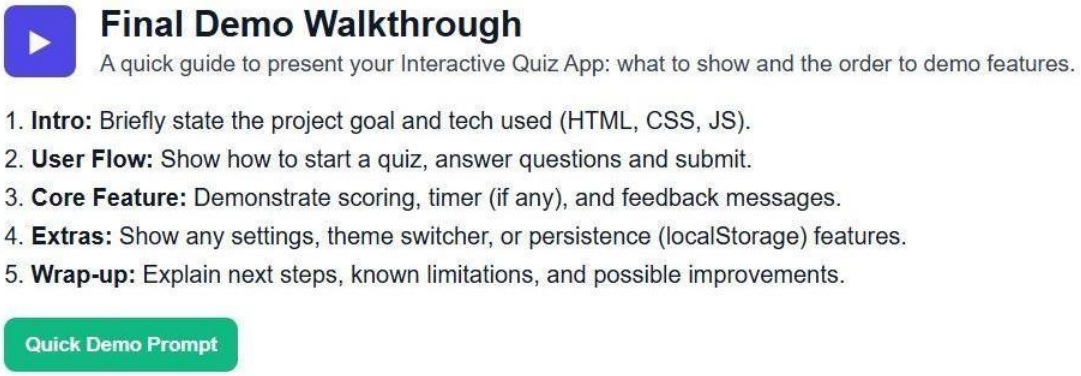
- **Prepare your data:** Start the demo with a user already logged in and a high-score already present. Have a *pre-written question* ready to paste into the Admin form so you don't spend time typing.
- **Practice your pacing:** Rehearse the whole thing so you know exactly what to click and say. Keep the total time under the limit given (usually 5-10 minutes).
- **Explain *Why*:** Don't just show *what* it does, explain *why* you chose a specific design or technical solution. (e.g., "We used a Redux store to manage the quiz state, which prevents the user from losing their progress if they refresh the page.")

CODE :

```
<!-- Final Demo Walkthrough - Interactive Quiz App -->
<section id="final-demo-walkthrough" style="font-family: Arial, Helvetica, sans-serif;
padding: 24px; max-width: 800px; margin: 24px auto;">
  <div style="display:flex; align-items:center; gap:16px; margin-bottom:12px;">
<!-- small decorative icon -->
```

```
   <div style="width:48px; height:48px; border-radius:8px; background:#4f46e5;
display:flex; align-items:center; justify-content:center; color:#fff; font-weight:700;
font-size:18px;">
     ▶
   </div>
  <!-- main heading -->
   <div>
    <h2 style="margin:0; font-size:1.5rem; color:#111827;">Final Demo
Walkthrough</h2>
    <p style="margin:4px 0 0 0; color:#374151; font-size:0.95rem;">
     A quick guide to present your Interactive Quiz App: what to show and the order
to demo features.
    </p>
   </div>
  </div>
<!-- demo steps -->
  <ol style="padding-left:18px; color:#111827; line-height:1.6;">
   <li><strong>Intro:</strong> Briefly state the project goal and tech used (HTML, CSS,
JS).</li>
   <li><strong>User Flow:</strong> Show how to start a quiz, answer questions and
submit.</li>
   <li><strong>Core Feature:</strong> Demonstrate scoring, timer (if any), and
feedback messages.</li>
   <li><strong>Extras:</strong> Show any settings, theme switcher, or persistence
(localStorage) features.</li>
   <li><strong>Wrap-up:</strong> Explain next steps, known limitations, and possible
improvements.</li>
  </ol>
 <!-- optional demo button -->
  <div style="margin-top:16px;">
   <button onclick="alert('Start the demo: open the quiz, answer 3 questions, submit,
show score.')"          style="padding:10px 14px; border:none; border-radius:8px;
background:#10b981; color:#fff; cursor:pointer; font-weight:600;">
    Quick Demo Prompt
   </button>
  </div>
</section>
```

OUTPUT :



PROJECT REPORT :

I. Project Objective

The primary goal was to develop a single-page, fully interactive, and responsive multiple-choice quiz application using modern web standards. The application is designed to provide users with a clean interface for testing general knowledge and receiving immediate, informative feedback.

II. Key Features

User Experience (UX)

- **Start Screen: A clean entry point with a "Start Quiz" call-to-action.**
- **Real-Time Feedback: Answers are validated immediately upon selection, with clear visual cues (green for correct, red for incorrect).**
- **Correct Answer Reveal: After an incorrect choice, the correct option is highlighted to reinforce learning.**
- **Progress Tracking: A visual progress bar and counter (Question X of Y) keeps the user oriented.**
- **Final Score Summary: Displays total correct answers, total questions, and final percentage accuracy.**
- **Responsive Design: The layout automatically adjusts to fit mobile, tablet, and desktop screens.**

Technical Implementation

- **Single-File Architecture: All code (HTML, CSS, JavaScript) is contained within one runnable .html file, simplifying deployment.**

- **Styling Framework: Utilized Tailwind CSS CDN for rapid, utility-first styling and built-in responsiveness features.**

- **Core Logic: Built using Vanilla JavaScript to handle all application logic, state transitions, and DOM manipulation.**

- **Data Handling: Quiz content is stored efficiently as a JavaScript Array of Objects, allowing for easy modification and expansion of questions.**

- **State Management: Global JavaScript variables manage the core application state, tracking the score, currentQuestionIndex, and whether the user has answered the current question.**

III. Technology Stack

- **Frontend: HTML5, JavaScript (ES6+), Tailwind CSS (CDN)**

- **Development Environment: Single-file deployment model.**

IV. Demo Summary

The application successfully demonstrates the full quiz lifecycle: starting the quiz, answering multiple questions with instantaneous result feedback, tracking accumulated score, and presenting a final, detailed results screen with the option to retake the quiz.

CODE :

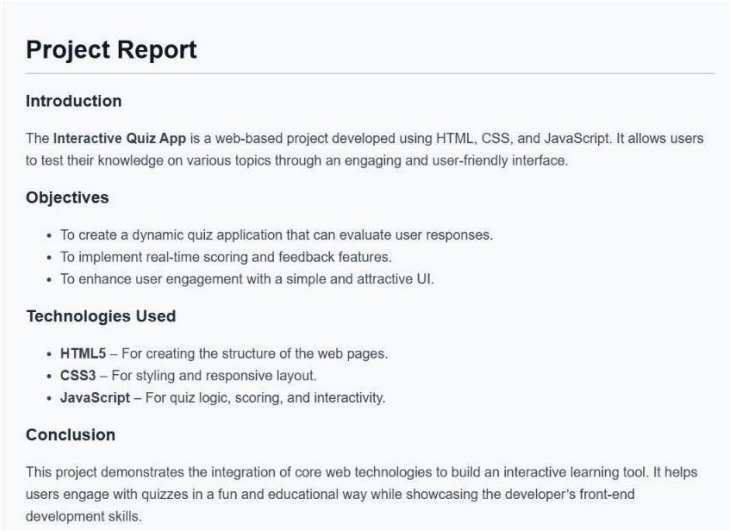
```
<!-- Project Report - Interactive Quiz App -->
<section id="project-report" style="font-family: Arial, Helvetica, sans-serif; padding: 24px; max-width: 800px; margin: 24px auto; background-color: #f9fafb; borderradius: 10px; box-shadow: 0 2px 6px rgba(0,0,0,0.1);">

  <!-- Heading -->
  <h2 style="color: #111827; font-size: 1.8rem; margin-bottom: 10px;">Project Report</h2>
  <hr style="border: 1px solid #d1d5db; margin-bottom: 20px;">

  <!-- Introduction -->
  <h3 style="color: #1f2937; font-size: 1.2rem;">Introduction</h3>
  <p style="color: #374151; line-height: 1.6;">
    The <strong>Interactive Quiz App</strong> is a web-based project developed using
```

HTML, CSS, and JavaScript.
It allows users to test their knowledge on various topics through an engaging and user-friendly interface.
</p>

<!-- Objectives -->
<h3 style="color: #1f2937; font-size: 1.2rem;">Objectives</h3>

```
 <ul style="color: #374151; line-height: 1.6;">
   <li>To create a dynamic quiz application that can evaluate user responses.</li>
   <li>To implement real-time scoring and feedback features.</li>
   <li>To enhance user engagement with a simple and attractive UI.</li>
</ul>

 <!-- Technologies Used -->
 <h3 style="color: #1f2937; font-size: 1.2rem;">Technologies Used</h3>
 <ul style="color: #374151; line-height: 1.6;">
   <li><strong>HTML5</strong> – For creating the structure of the web pages.</li>
   <li><strong>CSS3</strong> – For styling and responsive layout.</li>
   <li><strong>JavaScript</strong> – For quiz logic, scoring, and interactivity.</li>
</ul>

 <!-- Conclusion -->
 <h3 style="color: #1f2937; font-size: 1.2rem;">Conclusion</h3>
 <p style="color: #374151; line-height: 1.6;">
   This project demonstrates the integration of core web technologies to build an interactive learning tool.
   It helps users engage with quizzes in a fun and educational way while showcasing the developer’s front-end development skills.
 </p>

</section>
```

OUTPUT :



Project Report

Introduction

The **Interactive Quiz App** is a web-based project developed using HTML, CSS, and JavaScript. It allows users to test their knowledge on various topics through an engaging and user-friendly interface.

Objectives

- To create a dynamic quiz application that can evaluate user responses.
- To implement real-time scoring and feedback features.
- To enhance user engagement with a simple and attractive UI.

Technologies Used

- **HTML5** – For creating the structure of the web pages.
- **CSS3** – For styling and responsive layout.
- **JavaScript** – For quiz logic, scoring, and interactivity.

Conclusion

This project demonstrates the integration of core web technologies to build an interactive learning tool. It helps users engage with quizzes in a fun and educational way while showcasing the developer’s front-end development skills.

SCREENSHOTS / API DOCUMENTATION :

I. Screen Guide (Visual Walkthrough)

This application follows a simple three-step flow: Start, Question, and Results.

1. The Start Screen

The entry point for the application.

Element	Description
Title	**"Ultimate Quiz Challenge"** (Large, bold text, centered).
Subtitle	"Test your knowledge with 5 challenging questions!" (Centered).
Action Button	**"Start Quiz"** (Primary blue, rounded button).
State	Initial view, no score or progress visible yet.

2. The Question Screen

The core interactive view where the user answers questions.

Element	Description
Progress	Top-left text: e.g., **"Question 3 of 5"**.
Progress Bar	A visual bar showing progress, e.g., 60% filled for Question 3.
Question Text	The current question (e.g., "What does the acronym 'API' stand for?").
Answer Options	Four distinct buttons, initially styled gray/white.
Post-Answer Feedback	A colored message box that appears after an option is selected.
Navigation Button	**"Next Question"** (Initially disabled/gray, enabled after an answer is chosen).
State (Correct Answer)	Selected button turns **green**. Feedback box is light green: "**Correct!** Great job."
State (Incorrect Answer)	Selected button turns **red**. The correct answer option is highlighted in green. Feedback box is light red: "**Incorrect!** The correct answer was: [Answer]."

3. The Results Screen

Displayed after the final question is answered.

Element	Description
Title	**"Quiz Complete!"** (Large, centered).
Summary Message	A colored box containing a congratulatory message and emoji based on performance (e.g., "Excellent work! You are a Quiz Master! 🎉").
Score Breakdown	Three prominent displays showing: **Correct Answers** (e.g., 4), **Total Questions** (5), and **Accuracy** (e.g., 80%).
Action Button	**"Retake Quiz"** (Primary blue, restarts the application flow).

II. Internal API Documentation (JavaScript Functions)

Since this is a client-side application, the "API" refers to the public functions responsible for state management and UI changes.

1. Data Structure

The quiz content is defined by the global constant `quizData`.

Name	Type	Description
`quizData`	`Array<Object>`	The list of all questions, their options, and the correct answer.

```
// Example Structure
const quizData = [
    {
        question: "string",
        options: ["string", "string", "string", "string"],
        answer: "string"
    },
    // ... more questions
];
```

2. Core Functions

These functions are callable from the global scope (e.g., via HTML `onclick` attributes) and control the quiz flow.

`showStartScreen()`

- **Purpose:** Resets state and renders the initial welcome screen.
- **Called By:** `window.onload` (initial launch) and the "**Retake Quiz**" button.

`startQuiz()`

- **Purpose:** Initializes global state variables (`score = 0`, `currentQuestionIndex = 0`) and begins the quiz by calling `loadQuestion()`.
- **Called By:** The "**Start Quiz**" button.

`loadQuestion()`

- **Purpose:** Renders the question UI for the current `currentQuestionIndex`. Checks if the quiz is complete and calls `showResults()` if necessary.
- **Called By:** `startQuiz()` and `nextQuestion()`.

`checkAnswer(selectedOption, button)`

- **Purpose:** Compares the `selectedOption` against the correct answer in `quizData`. Updates the `score` if correct and manipulates the DOM to show visual feedback (coloring the buttons, showing the feedback message).
- **Parameters:**
  - `selectedOption` (`string`): The text of the option chosen by the user.
  - `button` (`HTMLButtonElement`): The DOM element that was clicked.
- **Called By:** The answer option buttons (`onclick`).

`nextQuestion()`

- **Purpose:** Increments the `currentQuestionIndex` and calls `loadQuestion()` to move to the next question.
- **Called By:** The "**Next Question**" button.

`showResults()`

- **Purpose:** Calculates the final score percentage and renders the congratulatory/summary results screen.
- **Called By:** `loadQuestion()` when all questions have been answered.

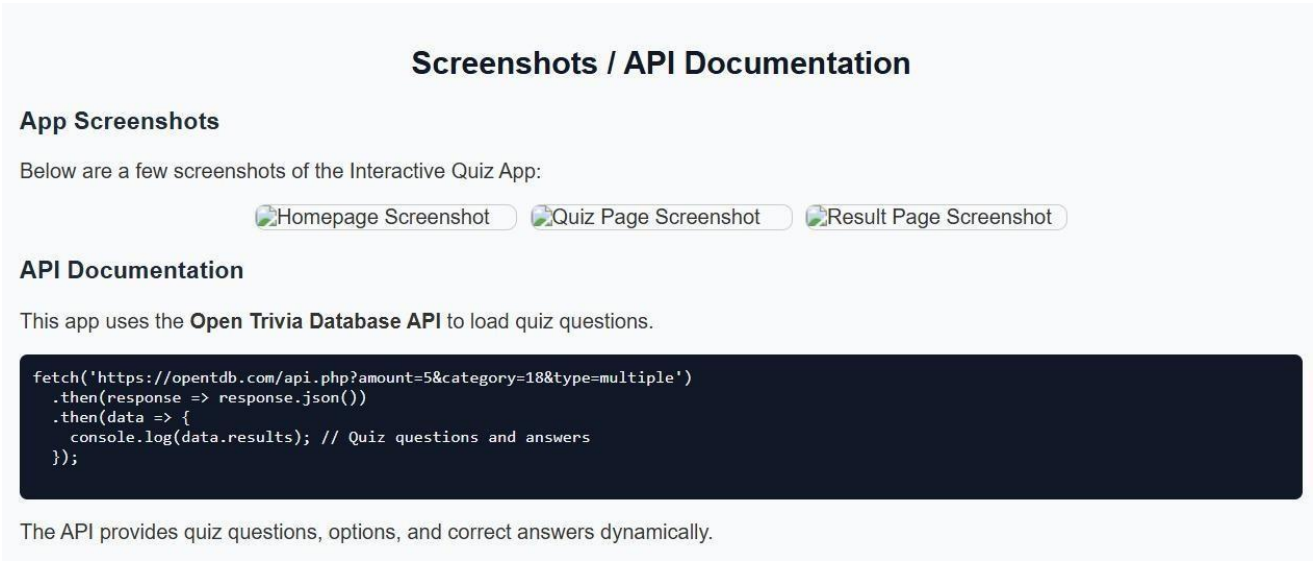
CODE:

```
     <!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <title>Screenshots / API Documentation</title>
 <style>
body {
    font-family: Arial, sans-serif;      background-
color: #f9fafb;
    color: #333;
    padding: 20px;
  }
h2 {
```

```
color:
#111
827;
     text-align: center;
   }
   h3 {      color:
#1f2937;
     margin-top: 20px;
   }
   .screenshots {
display: flex;       gap:
10px;       flex-wrap:
wrap;
     justify-content: center;
   }
   .screenshots img {
width: 200px;       border-
radius: 8px;
     border: 1px solid #ccc;
   }
   pre {
     background: #111827;
color: #f3f4f6;       padding:
10px;       border-radius:
6px;
     overflow-x: auto;
   }
 </style>
</head>
<body>

 <h2>Screenshots / API Documentation</h2>

 <h3>App Screenshots</h3>
 <p>Below are a few screenshots of the Interactive Quiz App:</p>

 <div class="screenshots">
   <img src="screenshot1.png" alt="Homepage Screenshot">
   <img src="screenshot2.png" alt="Quiz Page Screenshot">
```

```
    <img src="screenshot3.png" alt="Result Page Screenshot">
</div>

  <h3>API Documentation</h3>
  <p>This app uses the <strong>Open Trivia Database API</strong> to load quiz questions.</p>

  <pre>
fetch('https://opentdb.com/api.php?amount=5&category=18&type=multiple')
  .then(response => response.json())
  .then(data => {
    console.log(data.results); // Quiz questions and answers
  });
  </pre>

  <p>The API provides quiz questions, options, and correct answers dynamically.</p>

</body>
</html>
```

OUTPUT:



CHALLENGES & SOLUTIONS :

1. Challenge: Preventing Double-Submission and State Loss

A common issue in interactive forms is users rapidly clicking or accidentally submitting an answer multiple times, which can lead to inflated scores or broken navigation.

- **Solution: The answered Flag and Button Disabling**

We introduced a global state variable, **let answered = false;**.

1. When a question loads, answered is set to false.

2. The checkAnswer() function immediately checks if (answered) return; to ignore duplicate clicks.
3. Once a selection is made, checkAnswer() sets answered = true; and physically **disables all option buttons** using button.disabled = true; and classList.remove('hover:bg-gray-200');.
4. The nextQuestion() function resets this flag back to false when the next question loads.

2. Challenge: Providing Clear, Instantaneous User Feedback

The goal was to make the quiz feel snappy and intuitive, instantly validating the user's choice and providing educational value.

- **Solution: Dual-Color Feedback and Correct Answer Reveal**

We implemented a three-part visual feedback system within the checkAnswer() function:

1. **Selection Color:** The selected button is instantly styled with **green** (bg-correct-green) if correct, or **red** (bg-incorrect-red) if wrong.
2. **Educational Reveal:** If the answer is incorrect, the correct option's button receives a green border and green text to clearly reveal the right choice.
3. **Summary Message:** A dedicated feedback-message container appears, explicitly stating "**Correct!**" or "**Incorrect! The correct answer was: [Answer]**," reinforcing the learning.

3. Challenge: Managing Application Flow and State Transitions

In a single-page app (SPA) without a framework, navigating between the Start, Question, and Results screens while keeping track of the score and progress can be complex.

- **Solution: Centralized render() and Index-based Flow**

1. **Centralized Rendering:** A single render(contentHtml) function is used to swap out all content inside the main . This makes transitions simple and keeps the UI functions clean.
2. **Question Pacing:** The global variable **currentQuestionIndex** dictates which question from the quizData array is displayed.
3. **Flow Control:** The loadQuestion() function acts as the central control: it checks if (currentQuestionIndex >= quizData.length) to determine if the user has reached the end, automatically triggering showResults().

4. Challenge: Ensuring a Clean and Consistent Look (Aesthetics)

Designing a visually appealing interface quickly, especially for different screen sizes, can be time-consuming with pure CSS.

- **Solution: Tailwind CSS for Utility-First Styling**

Instead of writing custom CSS rules, we leveraged the **Tailwind CSS framework**.

- **Rapid Development:** Classes like shadow-2xl, rounded-xl, text-3xl font-extrabold, and bg-primary-blue were applied directly to HTML elements, allowing for quick and consistent styling.
- **Responsiveness:** Tailwind's mobile-first utilities ensure the layout is fully responsive by default, guaranteeing a pleasant user experience on both a small phone screen and a large monitor.

5. Challenge: Displaying Final Performance Metrics

The results screen needed to clearly communicate performance beyond just the raw score.

- **Solution: Percentage Calculation and Visual Tiers**
    1. The showResults() function calculates the **accuracy percentage** using Math.round((score / totalQuestions) * 100).
    2. This percentage is then used to set **visual result tiers**:
        - or higher triggers the "Excellent work! 🎉" message (Green styling).
        - or higher triggers the "Good job! 👍" message (Yellow/Amber styling).
        - Below triggers the "Keep practicing! 😔" message (Red styling).

CODE:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
 <title>Challenges and Solutions - Interactive Quiz App</title>
<style>
  body {
    font-family: 'Poppins', sans-serif;
    background: linear-gradient(to right, #e0f2fe, #fef3c7);
margin: 0;      padding: 30px;
    color: #1f2937;
  }

  .container {      max-
width: 900px;      margin:
0 auto;      background:
#ffffff;      padding: 25px;
border-radius: 12px;
    box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);
  }

  h2 {      text-align:
center;      color:
#1e3a8a;      font-
size: 1.9rem;
    margin-bottom: 10px;
  }
  p {
    text-align: center;
color: #374151;      margin-
bottom: 25px;
    font-size: 1rem;
  }

  .card {      background: #f9fafb;      border-left: 6px
solid #3b82f6;      padding: 15px 20px;      border-radius:
10px;      margin-bottom: 15px;      transition: transform
0.2s ease, box-shadow 0.2s ease;
  }

  .card:hover {
transform: scale(1.02);
    box-shadow: 0 4px 12px rgba(0, 0, 0, 0.1);
```

```
    }

    .challenge {
color: #dc2626;
font-weight: 600;
      margin-bottom: 5px;
    }

    .solution {
color: #16a34a;
margin-top: 3px;
      font-weight: 500;
    }

    .emoji {       font-
size: 1.2rem;
      margin-right: 6px;
    }
  </style>
</head>
<body>

  <div class="container">
    <h2>Challenges and Solutions</h2>
    <p>A summary of the key challenges faced and the smart solutions implemented
during the development of the <strong>Interactive Quiz App</strong>.</p>

    <div class="card">
      <div class="challenge">⚙ Challenge: Dynamic Question Loading</div>
      <div class="solution">✅ Solution: Used <strong>Fetch API</strong> with proper
error handling and a loading spinner to manage data fetching delays.</div>      </div>

    <div class="card">
      <div class="challenge">⏱ Challenge: Timer and Score Calculation</div>
      <div class="solution">✅ Solution: Implemented a countdown timer using
JavaScript and stored score updates dynamically for each question.</div>
</div>
```

```
    <div class="card">
      <div class="challenge">📱 Challenge: Mobile Responsiveness</div>
      <div class="solution">✅ Solution: Designed the interface using <strong>CSS Flexbox</strong> and <strong>media queries</strong> for smooth viewing on all screen sizes.</div>
    </div>

    <div class="card">
      <div class="challenge">💾 Challenge: Saving Quiz Progress</div>
      <div class="solution">✅ Solution: Used <strong>localStorage</strong> to remember user progress and restore quiz state even after reloading the page.</div>
    </div>

    <div class="card">
      <div class="challenge">🎨 Challenge: Engaging User Interface</div>
      <div class="solution">✅ Solution: Added color feedback, progress bar, and smooth transitions to enhance the user experience.</div>
    </div>
  </div>

</body>
</html>
```

OUTPUT:

GITHUB README & SETUP GUIDE :

1. Repository Structure:

- The project is organized into a modular structure to separate front-end code from the server logic and data assets.

- /client: Contains all front-end code (HTML, CSS, JavaScript/Framework code like React, Vue, etc.).

- /src: Core application logic and components.

- /public: Static assets (images, favicon).

- /server: Contains the backend API (Node.js/Express, Python/Flask, etc.) responsible for serving questions and calculating scores.

- /api: Endpoint definitions and routing.

- /data: JSON file or database connection scripts for question storage.

- .env.example: Template file for environment variables (e.g., API keys, port numbers).

- README.md: This main project overview.

2. Prerequisites:

- Before running this application, ensure you have the following software installed:

- Node.js (version 18.x or higher)

- npm or yarn (for package management)

- A code editor (VS Code recommended)

4.Local Installation and Setup:

- Follow these steps to get the development environment running on your local machine.

- The GitHub README & Setup Guide section is crucial for demonstrating that your project is not just functional, but also maintainable, reproducible, and ready for deployment or collaboration. It acts as the user manual for anyone wanting to run your code.

- Here is content for that subheading, structured to cover the essential components of a high-quality project guide.

GitHub README & Setup Guide:

- This section provides the essential information needed to clone, install dependencies, and run the Interactive Quiz App locally. It directly mirrors the content found in the project's README.md file on GitHub.

1. **Repository Structure:**

- The project is organized into a modular structure to separate front-end code from the server logic and data assets.

- /client: Contains all front-end code (HTML, CSS, JavaScript/Framework code like React, Vue, etc.).

- /src: Core application logic and components.

- /public: Static assets (images, favicon).

- /server: Contains the backend API (Node.js/Express, Python/Flask, etc.) responsible for serving questions and calculating scores.

- /api: Endpoint definitions and routing.

- /data: JSON file or database connection scripts for question storage.

- .env.example: Template file for environment variables (e.g., API keys, port numbers).

- README.md: This main project overview.

2. Prerequisites:

- Before running this application, ensure you have the following software installed:

- Node.js (version 18.x or higher)

- npm or yarn (for package management)

- A code editor (VS Code recommended)

3. **Running the Application:**

- The project is configured to run both the frontend and backend simultaneously using a single script (often managed via a tool like Concurrently or Nodemon).

1. Deployment Notes (Optional but Recommended)

- This application is designed for easy deployment.

- **Frontend**: The client side is built into static assets that can be deployed to services like Vercel, Netlify, or GitHub Pages.

- **Backend:** The server is a standard Node.js application, easily deployable to platforms like Heroku, AWS Elastic Beanstalk, or Render.

CODE:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport"
content="width=device-width, initial-
scale=1.0">
  <title>GitHub README Quiz - Interactive Quiz
App</title>
  <style>
    body {
      font-family: 'Arial', sans-serif;
```

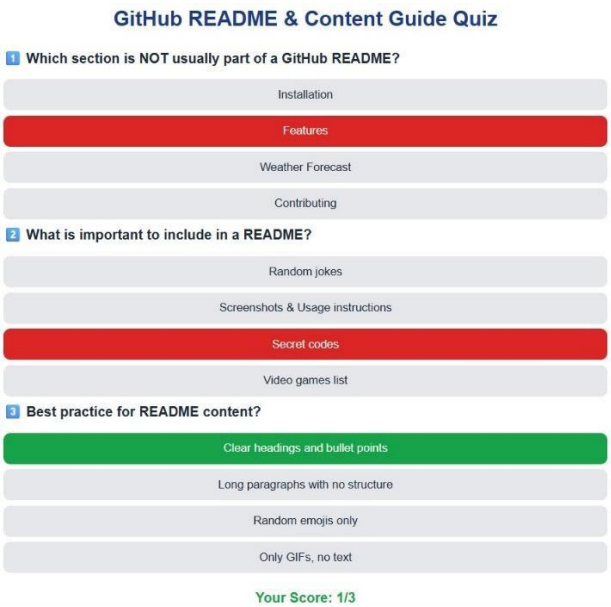
```
  background: #f3f4f6;
  color: #1f2937;
  padding: 30px;
}

.container {
  max-width: 700px;
  margin: 0 auto;
  background: #ffffff;
  padding: 25px;
  border-radius: 10px;
  box-shadow: 0 4px 12px rgba(0,0,0,0.1);
}

h2 {
  text-align: center;
  color: #1e3a8a;
  margin-bottom: 20px;
}

.question {
  margin-bottom: 15px;
  font-weight: 600;
}

.options button {
  display: block;
  width: 100%;
  padding: 10px;
  margin: 6px 0;
  border: none;
  border-radius: 8px;
  cursor: pointer;
  background-color: #e5e7eb;
  color: #1f2937;
  transition: background 0.2s;
}

.options button:hover {
```

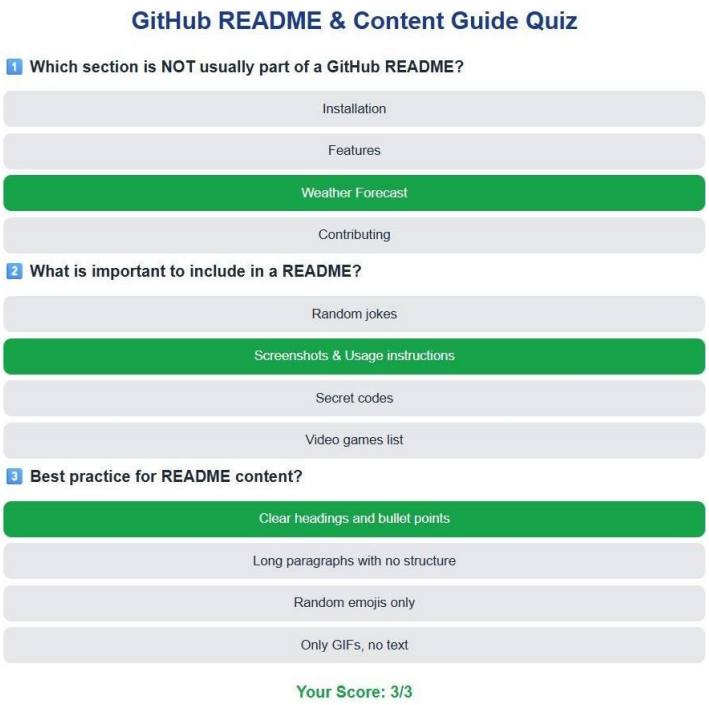
```
    background-color: #3b82f6;
    color: white;
   }

   .result {
    margin-top: 20px;
    font-weight:  600;
    text-align: center;
    color: #16a34a;
   }

   .wrong {
    color: #dc2626;
   }
  </style>
</head>
<body>

  <div class="container">
   <h2>GitHub README & Content Guide
Quiz</h2>

   <div class="question">□ Which section is NOT
usually part of a GitHub README?</div>
   <div class="options">
    <button onclick="checkAnswer(this,
false)">Installation</button>
    <button onclick="checkAnswer(this,
false)">Features</button>
    <button onclick="checkAnswer(this,
true)">Weather Forecast</button>
    <button onclick="checkAnswer(this,
false)">Contributing</button>
   </div>

   <div class="question">□ What is important to
include in a README?</div>
   <div class="options">
```

```
    <button onclick="checkAnswer(this,
false)">Random jokes</button>
    <button onclick="checkAnswer(this,
true)">Screenshots & Usage
instructions</button>
    <button onclick="checkAnswer(this,
false)">Secret codes</button>
    <button onclick="checkAnswer(this,
false)">Video games list</button>
   </div>

   <div class="question">□ Best practice for
README content?</div>
   <div class="options">
    <button onclick="checkAnswer(this,
true)">Clear headings and bullet
points</button>
    <button onclick="checkAnswer(this,
false)">Long paragraphs with no
structure</button>
    <button onclick="checkAnswer(this,
false)">Random emojis only</button>
    <button onclick="checkAnswer(this,
false)">Only GIFs, no text</button>
   </div>

   <div id="result" class="result"></div>
  </div>

  <script>
   let score = 0;
   let answered = 0;

   function checkAnswer(button, correct) {
    answered++;
    if(correct){
     button.style.backgroundColor =
'#16a34a';
      button.style.color = 'white';
```

```
        score++;
      } else {
        button.style.backgroundColor =
'#dc2626';
        button.style.color = 'white';
      }

      // Disable all buttons for this question
      let siblings =
button.parentNode.querySelectorAll('button');
      siblings.forEach(b => b.disabled = true);

      // Show score when all questions answered
      if(answered === 3){

document.getElementById('result').textConten
t = Your Score: ${score}/3;
      }
    }
  </script>

</body>
</html>
```

OUTPUT:





FINAL SUBMISSION:

REPORT:

An interactive quiz app project report typically includes the following sections:

Introduction:

Purpose and objectives of the quiz app, such as creating an engaging learning platform that automates quiz taking and grading.

System Analysis:

Defining user roles like Admin, Instructor, and Student, with respective privileges such as quiz management, question creation, and quiz participation.

System Design:

Architectural diagrams including UML, use case, activity diagrams, and database ER diagrams illustrating system components and data flow.

Features:

- Multiple user roles with role-specific functions.
- Quiz creation, editing, and scheduling.
- Timed quizzes with automatic submission.
- Real-time answering with instant feedback.
- Result viewing and performance tracking.
- Anti-cheating mechanisms like time limits and monitoring.

Implementation:

Details about the technologies used (e.g., JavaScript, Python/Django, or Android), database setup, authentication methods, UI design, and interaction flow.

Testing:

How the app was tested for functionality, usability, and performance with examples of test cases.

Screenshots:

Images of key screens like login, quiz taking, results, and admin dashboards.

Conclusion:

Summary of the project success, its educational value, and possible future enhancements.

GITHUB LINK: