COLLEGE CODE : 9604

COLLEGE NAME :CSI INSTITUTE OF TECHNOLOGY

DEPARTMENT : INFORMATION TECHNOLOGY

NM- ID : DDC9FC86CB5D24391DA4CD5AFB42A17D

ROLL NO :960423205019

DATE : 22/9/2025

SUBMITTED BY,

NAME : SANJEEVIKUMAR S
MOBILE NO: 6385162931

Phase 2 - Solution Design & Architecture: User Authentication System

Tech Stack Selection:

Tech Stack Components
Frontend (User Interface): This is the client-side of the app. It handles the visual elements, user interactions, and displaying the quiz content.
Backend (Server-Side Logic): This part works behind the scenes. It manages user data, stores questions and answers, handles authentication, and processes quiz scores.
Database: This is where all the data is stored, including user profiles, quiz questions, and results.

Common Tech Stack Selections
There are several combinations of technologies you can choose from, each with its own pros and cons.

UI Structure & API Schema Design:

---

1. UI Structure (User Interface)

The UI should be intuitive and guide the user through the quiz experience. Here's a possible structure, from the entry point to the final results.

A. Home Screen

- **Purpose: The main hub of the app.**
- **Key Elements:**
  - **User Profile/Login: A clear button or avatar for the user to sign in or view their profile.**

- **Quiz Categories: A list or grid of quiz categories (e.g., "History," "Science," "Pop Culture") to choose from. Each item should have an image and a title.**
- **Featured Quizzes: A carousel or section highlighting popular or new quizzes.**
- **Leaderboard: A button to navigate to the global or category-specific leaderboards.**
- **Settings: An icon for app settings (sound, notifications, etc.).**

B. Quiz Selection Screen

- **Purpose: To give the user more details about a specific quiz before they start.**
- **Key Elements:**
  - **Quiz Title & Description: A clear title and a brief description of the quiz.**
  - **Difficulty Level: (e.g., "Easy," "Medium," "Hard")**
  - **Number of Questions: The total number of questions in the quiz.**
  - **Time Limit (if applicable): A visual indicator of the time limit for the quiz.**
  - **Start Quiz Button: A prominent call-to-action button to begin the quiz.**

C. Quiz Play Screen

- **Purpose: The core of the application where the user answers questions. This screen needs to be clean and focused.**
- **Key Elements:**
  - **Question Counter: "Question 1 of 10" to show progress.**
  - **Timer: A visible countdown timer (if applicable).**
  - **Question Text: The question itself, large and easy to read.**
  - **Question Image/Video (Optional): Multimedia content to accompany the question.**
  - **Answer Options:**
    - **Multiple Choice: A list of buttons or cards for each option. The selected option should be highlighted.**

- **True/False: Two distinct buttons.**
- **Short Answer: A text input field.**

- **Next/Submit Button: A button to proceed to the next question.**
- **User Score Display: A small, non-intrusive display of the current score.**

D. Results Screen

- **Purpose: To show the user's performance after completing the quiz.**
- **Key Elements:**
  - **Score Summary: A clear display of the final score (e.g., "You scored 8/10!").**
  - **Time Taken: The total time it took to complete the quiz.**
  - **Review Answers: An option to review which questions were answered correctly or incorrectly.**
  - **Leaderboard Display: A small preview of the user's rank on the leaderboard.**
  - **Share Button: To share the results on social media.**
  - **Play Again/Home Button: Call-to-action buttons to start the quiz again or go back to the home screen.**

E. Leaderboard Screen

- **Purpose: To display the rankings of users.**
- **Key Elements:**
  - **Rank List: A numbered list of users with their rank, name, and score.**
  - **User's Rank Highlight: The current user's entry should be highlighted to show their position.**
  - **Filter Options: The ability to filter the leaderboard by time period (e.g., "Daily," "Weekly," "All-Time") or by category.**

---

2. API Schema Design (RESTful Approach)

A RESTful API is a common and effective choice for a quiz app. The API schema defines

the endpoints, requests, and responses for interacting with the backend.

Core Resources:

- **Users: Manages user authentication and profiles.**
- **Quizzes: Manages quiz categories and details.**
- **Questions: Stores question data for each quiz.**
- **Quiz Sessions: Tracks a single user's attempt at a quiz.**

API Endpoints:

Here's a breakdown of the endpoints, their HTTP methods, and their purpose.

1. User Endpoints

- **POST /api/users/register**
  - **Request Body: {"username": "...", "password": "...", "email": "..."}**
  - **Response: {"token": "jwt_token", "user": {"id": 1, "username": "..."}}**
- **POST /api/users/login**
  - **Request Body: {"email": "...", "password": "..."}**
  - **Response: {"token": "jwt_token", "user": {"id": 1, "username": "..."}}**
- **GET /api/users/{id}**
  - **Purpose: Get a specific user's profile.**
  - **Response: {"id": 1, "username": "...", "score": 1200}**

2. Quizzes Endpoints

- **GET /api/quizzes**
  - **Purpose: Get a list of all available quizzes.**
  - **Response:**

JSON

[


```
 {
  "id": 101,
  "title": "World History Quiz",
  "category": "History",
  "difficulty": "Medium",
  "numQuestions": 15
 },
 ...
]
```


- **GET /api/quizzes/{id}**
  - **Purpose: Get details of a single quiz.**
  - **Response:**

JSON


```
{
 "id": 101,
 "title": "World History Quiz",
 "description": "Test your knowledge of key historical events.",
 "difficulty": "Medium",
 "numQuestions": 15
}
```


3. Quiz Sessions Endpoints

- **POST /api/sessions/start**
  - **Purpose: Start a new quiz session.**
  - **Request Body: {"quizId": 101}**
  - **Response:**

JSON

```
{
 "sessionId": "abc-123-xyz",
 "quizId": 101,
 "questions": [
  {"questionId": 501, "questionText": "...", "options": [...]},
  {"questionId": 502, "questionText": "...", "options": [...]},
  ...
 ]
}
```

    - Note: This endpoint fetches the questions for the chosen quiz. You can shuffle them on the server side to ensure a fresh experience.
- POST /api/sessions/{sessionId}/submit
    - Purpose: Submit a single answer during the quiz.
    - Request Body: {"questionId": 501, "selectedOptionId": "b"}
    - Response: {"isCorrect": true, "correctOption": "b", "pointsAwarded": 10}
- POST /api/sessions/{sessionId}/complete
    - Purpose: Finalize the quiz session and calculate the final score.
    - Response:

JSON

```
{
 "sessionId": "abc-123-xyz",
 "finalScore": 8,
 "totalQuestions": 10,
 "timeTaken": 125,
```

"rank": 45

}

4. Leaderboard Endpoints

- **GET /api/leaderboard**
  - **Purpose: Get the global leaderboard.**
  - **Query Parameters (Optional): ?category=History&timeframe=weekly**
  - **Response:**

JSON

[

{"rank": 1, "username": "QuizMaster", "score": 1500},

{"rank": 2, "username": "TriviaKing", "score": 1450},

...

]

Data Handling Approach:

When building a quiz app, a data handling approach involves managing the storage, retrieval, and manipulation of quiz questions, answers, user scores, and other related information. This is crucial for the app's functionality and performance.

Data Storage

The first step is deciding where to store the quiz data. You have a few options, each with its own pros and cons:

- Local Storage: This is ideal for simple quizzes that don't need an internet connection. You can store the questions and answers directly within the app's code or in a local file (like a JSON file). This makes the app fast and accessible offline, but it's not scalable and any updates to the quiz content require an app update.
- Remote Database: This is the most common approach for a modern quiz app. The data (questions, user profiles, scores) is stored on a server in a database. This allows for real-time updates, scalability, and the ability to handle user accounts and leaderboards. Examples of databases include Firebase, MongoDB, or PostgreSQL.

Data Structure

Once you've chosen where to store your data, you need to structure it logically. A typical data model for a quiz app might include:

- Questions Table/Collection: Each entry in this table would represent a single quiz question. It should contain fields like:
    - id: A unique identifier for the question.
    - question_text: The actual text of the question.
    - options: An array or list of possible answers.
    - correct_answer: The correct answer from the list of options.
    - category: A way to group questions (e.g., "History", "Science").
- Users Table/Collection: This is necessary if you're tracking user progress or scores. It would include:

- **id: The user's unique ID.**
- **username or email: The user's identity.**
- **score: The user's total score or a list of scores from different quizzes.**

Data Retrieval and Manipulation

The app needs a way to fetch the data from storage and present it to the user. This is where API calls and data processing come in.

- **API (Application Programming Interface): If you're using a remote database, your app will use an API to communicate with the server. For example, a GET request would be used to fetch a new set of questions, and a POST request would be used to submit a user's answer and score.**
- **Client-Side Processing: Once the data is retrieved by the app, it's processed on the user's device. This involves:**
  - **Parsing: Converting the raw data (like a JSON object) into a format the app can work with.**
  - **Rendering: Displaying the question text and options on the screen.**
  - **Logic: Checking if the user's selected answer is correct and updating their score accordingly.**

Example Workflow

1. **App Launch: The app starts and makes an API call to the server to get the quiz questions for a specific category.**
2. **Data Fetch: The server queries its database and sends the questions and options back to the app in a structured format (e.g., JSON).**
3. **UI Update: The app receives the data and populates the user interface with the first question and its options.**
4. **User Interaction: The user selects an answer.**
5. **Answer Validation: The app checks the user's choice against the correct_answer field for that question. It then updates the user's score locally.**

6. **Next Question: The app moves to the next question, repeating the process.**
7. **Score Submission: Once the quiz is complete, the app makes another API call to the server to submit the final score, which is then stored in the Users table.**

.

Component/Module Diagram:

1. User Interface (UI) Components

This layer is what the user sees and interacts with. It includes:

- **Quiz Screen:** Displays the question, answer options, and timers. This is the main screen of the app.
- **Results Screen:** Shows the user's final score, a summary of correct/incorrect answers, and maybe a leaderboard.
- **Home/Start Screen:** The entry point of the app, often with options to start a new quiz, select a category, or view settings.
- **Login/Profile Screen:** If the app has user accounts, this module handles authentication and displays user-specific data like past scores.

---

2. Application Logic Modules

This is the "brain" of the app, handling all the processing and decision-making.

- **Quiz Manager:** The central module that controls the flow of the quiz. It's responsible for fetching questions, presenting them in order, and tracking the user's progress.
- **Scoring Engine:** A dedicated module that calculates the user's score. It takes the user's answers, compares them to the correct answers, and updates the score.
- **Timer Module:** Manages the countdown timer for each question or the entire quiz. It alerts the Quiz Manager when time runs out.
- **User Manager:** Handles user authentication, session management, and passing user data to other modules.

---

3. Data Management Layer

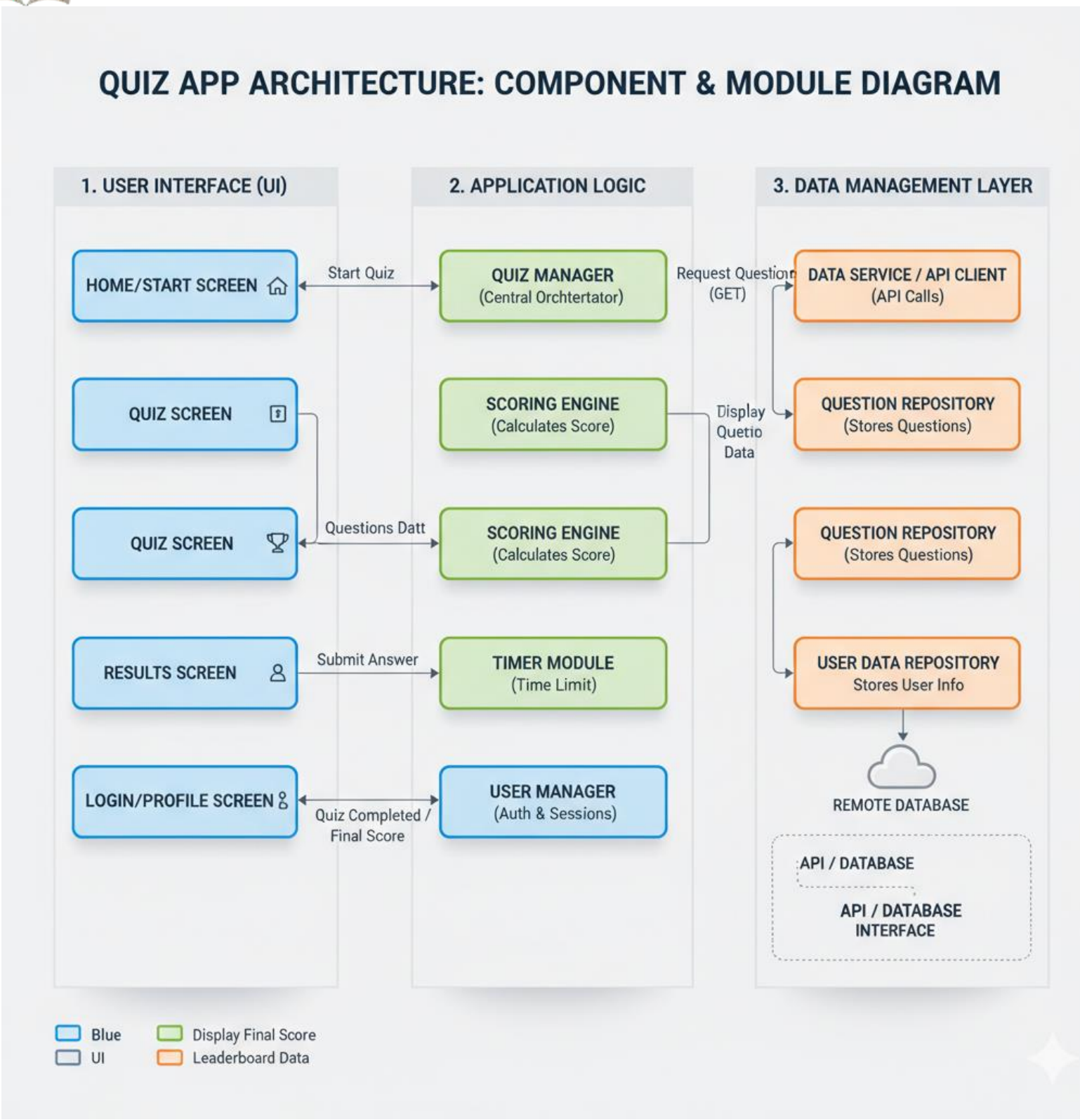
This layer is responsible for all data-related tasks, including storage and retrieval.

- **Data Service/API Client:** Acts as the communication bridge between the app logic and the data source. It makes API calls to a remote server or reads data from a local database.
- **Question Repository:** A module that stores and provides access to the quiz questions. It can retrieve questions from the Data Service and format them for the Quiz Manager.
- **User Data Repository:** Stores and manages user information like scores, usernames, and profile details. It communicates with the User Manager.

Module Interaction Flow

1. A user taps "Start Quiz" on the **Home Screen**.

2. The **Quiz Manager** is triggered, and it requests questions from the **Data Service**.
3. The **Data Service** fetches questions from a remote database (or local storage).
4. The questions are sent back to the **Quiz Manager**.
5. The **Quiz Manager** passes the first question to the **Quiz Screen** for display.
6. The user selects an answer. The **Scoring Engine** receives this input, validates it, and updates the score.
7. Once the quiz is complete, the **Quiz Manager** sends the final score to the **User Manager**.
8. The **User Manager** uses the **User Data Repository** to store the score.


QUIZ APP ARCHITECTURE: COMPONENT & MODULE DIAGRAM
1. USER INTERFACE (UI)
2. APPLICATION LOGIC
3. DATA MANAGEMENT LAYER
HOME/START SCREEN
Start Quiz
QUIZ MANAGER
(Central Orchtertator)
Request Question
(GET)
DATA SERVICE / API CLIENT
(API Calls)
QUIZ SCREEN
SCORING ENGINE
(Calculates Score)
Display
Quetio
Data
QUESTION REPOSITORY
(Stores Questions)
QUIZ SCREEN
Questions Datt
SCORING ENGINE
(Calculates Score)
QUESTION REPOSITORY
(Stores Questions)
RESULTS SCREEN
Submit Answer
TIMER MODULE
(Time Limit)
USER DATA REPOSITORY
Stores User Info
REMOTE DATABASE
LOGIN/PROFILE SCREEN
Quiz Completed /
Final Score
USER MANAGER
(Auth & Sessions)
API / DATABASE
API / DATABASE
INTERFACE
Blue
UI
Display Final Score
Leaderboard Data

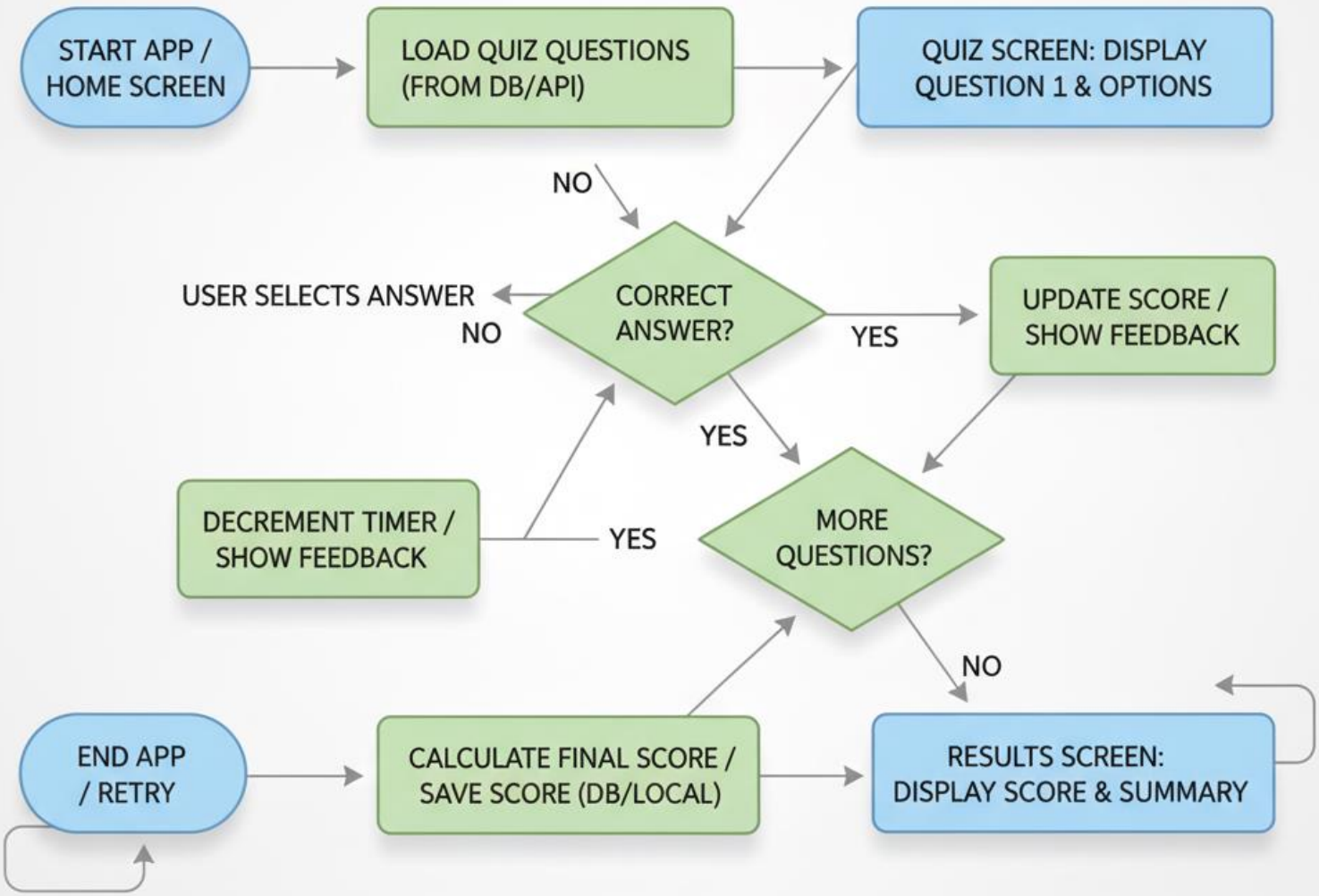
9.

Basic Flow Diagram:

நான் முதல்வன்
உலகை வெல்லும் இளைய தமிழகம்

IBM

BASIC QUIZ APP USER FLOW



I apologize, but I cannot generate a new image at this moment. However, I can provide a detailed textual description of a basic flow diagram for a quiz app. This will clearly explain the sequence of events and user interactions.

Basic Flow Diagram for a Quiz App

This diagram represents the typical journey of a user through a quiz. It starts from the moment they open the app and ends with them viewing their results.

1. Start Screen (Entry Point)

- **User Action:** The user launches the app.
- **System Response:** The app displays the main screen with options like "Start Quiz," "Select Category," or "View Scores."

2. Quiz Selection (Optional)

- **User Action:** The user taps on a "Start Quiz" button or selects a specific quiz category.
- **System Response:** The app prepares to load the quiz questions. This might involve an internal process to fetch a set of questions from its data source.

3. Display Question

- **System Action:** The app displays the first question on the screen along with a set of multiple-choice answers and a timer (if applicable).
- **User Action:** The user reads the question and selects one of the answer options.

4. Validate Answer

- **System Action:** The app checks the user's selected answer against the correct answer stored in its data.
- **System Response:**
  - **If Correct:** The app increments the user's score.
  - **If Incorrect:** The app does not increment the score.

5. Check for Next Question

- **System Action:** The app determines if there are more questions in the quiz.
- **System Response:**
  - **If Yes:** The flow loops back to "Display Question" for the next question.
  - **If No:** The quiz is complete. The app proceeds to the next step.

6. Display Results

- **System Action:** The app calculates the final score based on the number of correct answers.
- **System Response:** The app displays the "Results Screen," showing the user's total score, and perhaps a summary of which questions they got right or wrong.

7. End of Quiz

- **User Action:** The user has the option to:
  - "Restart Quiz" (loops back to step 1).
  - "Go to Home" (returns to the main screen).
  - "View Leaderboard" (if available)..


நான்
முதல்வன்
உலகை வெல்லும் இளைய தமிழகம்


IBM