## Prerequisites for Developing a Timer-Based Quiz Application

Here's a breakdown of the technical and development requirements for successfully building and managing this timer-based quiz application:

---

## 1. Technical Skills

### Core JavaScript Concepts

1. **Closures**:
    - Used to create a `createTimer` function that encapsulates the timer state and provides a cleanup mechanism.
2. **Async/Await and Promises**:
    - Handle asynchronous operations for fetching questions via the Open Trivia Database API (`fetchQuestions`).
3. **Array Manipulations**:
    - Shuffle answers using `sort(() => Math.random() - 0.5)`.
    - Compare user responses with correct answers for score calculation.
4. **DOM Manipulation**:
    - Dynamically update the DOM using `innerHTML`.
    - Attach event listeners to buttons for user interaction.
5. **Event Handling**:
    - Manage click events for selecting answers and proceeding to the next question.
6. **Timers**:
    - Use `setInterval` and `clearInterval` to implement countdown functionality.

---

## 2. Functional Requirements

### Question Fetching

- Fetch questions dynamically using the Open Trivia Database API.
- Support specifying a category and the number of questions.

**Dynamic Question Rendering**

- Display the question and shuffled answer choices.
- Highlight the current question's timer.

**Answer Selection**

- Capture the user's answer for each question.
- Disable further selection after the user has chosen an answer.

**Timer Functionality**

- Start a 15-second countdown for each question.
- Automatically move to the next question if the timer expires.

**Score Calculation**

- Compare user responses with correct answers.
- Display the final score at the end of the quiz.

---

## 3. Development Workflow

**Initialization**

1. Fetch questions using `fetchQuestions`.
2. Start with the first question and initialize the timer.

**Rendering**

1. Display the current question and shuffled answer choices dynamically.
2. Update the timer countdown on each tick.

**User Interaction**

1. Allow the user to select an answer and stop the timer.
2. Proceed to the next question when the "Next" button is clicked.

**Quiz Completion**

1. Stop all active timers when the quiz ends.
2. Calculate and display the user’s score.

---

## 4. Frontend Design

**UI Elements**

1. **Question Display**:
    - Show the current question prominently.
    - Include a timer below the question.
2. **Answer Choices**:
    - Display answers as buttons.
    - Highlight the selected answer.
3. **Navigation**:
    - A "Next" button to proceed after selecting an answer.
4. **Score Screen**:
    - Display the total score out of the total number of questions.

**Styling**

- Use basic CSS or frameworks like **Tailwind CSS** for:
    - Layout (`flex`, `grid`).
    - Button styles (`bg-gray-200`, `hover:bg-gray-300`).
    - Responsive design.

---

## 5. Key Functions

### `createTimer(duration, onTick, onEnd)`

- Encapsulates the timer logic with:
    - `onTick`: Updates the UI with the remaining time.
    - `onEnd`: Executes when the timer reaches 0.

### `fetchQuestions(category, amount)`

- Fetches questions dynamically from the Open Trivia Database API.

### `renderQuestion(questions)`

- Displays the current question and possible answers dynamically.
- Starts a new timer for each question.

### `selectAnswer(answer)`

- Records the user's selected answer.
- Stops the timer and enables the "Next" button.

`moveToNextQuestion(questions)`

- Moves to the next question or ends the quiz if no questions remain.

`calculateScore(answers, correctAnswers)`

- Compares user answers to correct answers and calculates the score.

---

## 6. Testing Scenarios

### Functional Tests

1. Verify that questions are fetched dynamically and displayed correctly.
2. Check if the timer starts and stops correctly for each question.
3. Confirm that selecting an answer stops the timer.
4. Ensure score calculation is accurate.

### Edge Cases

1. Handle cases where the user doesn't select an answer before the timer expires.
2. Validate API response errors and provide a fallback (e.g., empty quiz).

---

## 7. Advanced Enhancements (Optional)

1. **Difficulty Levels**:
    - Allow users to select difficulty (easy, medium, hard).
2. **Categories**:
    - Provide a dropdown for category selection.
3. **Progress Bar**:
    - Show quiz progress visually.
4. **Leaderboard**:
    - Save and display high scores locally using `localStorage`.

By following these prerequisites and the outlined workflow, you can build a fully functional and user-friendly timer-based quiz application. Let me know if you need help with specific aspects!