Quiz App

Objective:

The goal of this assignment is to assess front-end fundamentals, state management, and the ability to build a clean, user-friendly application in React by creating a Quiz App with scoring and results.

Requirements:

1. UI/UX:
    - Clean, responsive layout that works on desktop and mobile.
    - Show one question at a time with four options.
    - Prominent navigation/actions (Next, Previous/Skip if implemented, Submit/Finish).
    - Display score and progress clearly.
    - Use any modern, readable font (e.g., system default or Inter/Roboto).

2. Core features:

    Quiz Page
    - Load 5–10 multiple-choice questions either from the Open Trivia DB API (https://opentdb.com/api_config.php) or a provided local JSON file.
    - Render a single question at a time with 4 options.
    - The user selects an answer before moving to the next question.

    Score Tracking

    - Track correct/incorrect selections.
    - At the end, show a final score (e.g., “You scored 7/10”).

    Results Page

    - Show a summary of answers: which ones were correct/incorrect and the user’s selected option vs. the correct option.
    - Provide a Restart Quiz action to attempt again.

3. Technical Requirements:
    - Use React functional components with hooks (at minimum: `useState`, `useEffect`).
    - Use props effectively to pass data into presentational components.
    - Style with CSS / Tailwind / or Styled Components.
    - Manage state transitions for the quiz flow (Question → Answer → Next Question → Results).

    - Bonus: Add React Router with routes like `/quiz` and `/results`.

4. State flow:
    - Load questions (from API or JSON) → initialize quiz state.
    - For each question: capture selection → lock in answer → navigate to next.
    - On completion: compute total score → navigate to Results → allow Restart (reset state).

5. Data source:
    - Option A (API): Use Open Trivia DB. Handle loading & error states and normalize API results into your UI model.
    - Option B (Local): Ship a `questions.json` with exactly the fields your UI needs.

Testing:

Basic Buttons

- Handle edge cases: no internet (if using API), empty/short data, timeouts, rapid clicks, and page refreshes.
- Prevent progressing without a selection (unless you implement an explicit Skip feature).
- Ensure mobile responsiveness..

Bonus Features:

- Timer per question (e.g., 30 seconds) that auto-locks the answer when time runs out.
- Progress indicator (e.g., “Question 3 of 10” or a progress bar).
- Difficulty levels (easy/medium/hard) that change the question set.
- Persistent high scores via `localStorage`.
- Subtle animations (fade-in questions, button tap feedback).
- Accessibility considerations (keyboard navigation, ARIA labels, focus states).

Submission Guidelines:

1. **Repository:** Submit the code through a GitHub repository. Include a README file with instructions on how to run the project.
2. **Timeline:** The challenge must be completed within 24 hours of receiving it.
3. **Demo:** Provide a link to a live demo (using services like GitHub Pages, Netlify, or Vercel).
4. **Documentation:** Include comments in the code and a brief document explaining the architecture and design decisions.
5. **Questions:** if you need any clarifications regarding the challenge, you may write to us at hiring@todaypay.me.

6. **Submission:** After successful completion of the challenge within the deadline, You can submit your solution through this Form (https://forms.gle/PePm24oeYzPBKnKW8) for review and next steps.

Follow-Up Interview:

1. **Code Walkthrough:** Be prepared to walk through your code, explaining your approach and any challenges faced.
2. **Feature Expansion:** Be ready for a small real-time coding task, such as adding a new feature or fixing a bug in your implementation.

🎉Good luck and we are looking forward to going through your solutions!