Architecture and Design Decisions

I developed this quiz application with a primary focus on clarity, simplicity, and React fundamentals. My objective was to implement the necessary features in a clean manner, while simultaneously incorporating subtle enhancements to enhance the user experience.

Overall Approach

I employed React with functional components and hooks (useState, useEffect, useMemo, useRef). The application state is managed locally within the primary QuizPage component, which maintains the following:

* Current question index
* User's selected answers
* Navigation state (enabling Next/Submit buttons)

For improved code clarity, I separated the results display into a ResultsView component. Additionally, I incorporated React Router to maintain a separation between the quiz flow (/), results page (/results), and other components.

State Flow

The quiz flow is straightforward:

1. Upon startup, the application loads questions from a local JSON file.
2. For each question, the selected answer index is stored in an array.
3. Next functionality is enabled only after an option has been selected, while Previous functionality is accessible except on the initial question.
4. Upon Submit, the application calculates the score and navigates to the results page.
5. The results page presents a comprehensive summary and facilitates the user's ability to restart.

UI/UX Choices

* Presenting one question at a time to minimize distractions.
* Utilizing a progress bar to indicate completion percentage.
* Implementing modern option cards with hover and selected states to enhance the UI's interactivity.
* Employing an Aurora background with glass cards to create a distinctive and visually appealing design.
* Incorporating green/red chips in results for prompt feedback on correctness.
* Utilizing responsive buttons for optimal usability on mobile devices.

Accessibility

The application is designed to be accessible via keyboard and screen readers:

* Focus automatically moves to the question text during navigation.
* Radio groups are appropriately labeled with ARIA attributes.
* Buttons are disabled when actions are invalid.
* A small guard is added to prevent double-clicking on Next/Submit.

Data Source

For simplicity and reliability, I opted to utilize a local JSON file (questions.json).
Each question includes:

* **id:** A unique identifier for the question.
* **question:** The text of the question.
* **options:** A list of possible answers for the question.

* **correctIndex:** The index of the correct answer.

This structure makes it easy to swap the data source with an API like Open Trivia DB in the future.

Deployment

I deployed the app on Vercel, which provides automatic builds from GitHub.
To ensure that routes like /results work on refresh, I included a vercel.json file with a rewrite rule that redirects all routes to index.html.

Tradeoffs

I focused on completing all mandatory requirements and polishing the UI/UX.
Optional features like timers, difficulty filters, and persistent high scores were not added, as I preferred to keep the project simple and readable. These can be added later if needed.

Summary

This project fulfills all requirements:

* Functional quiz flow with navigation and score calculation.
* Detailed results summary.
* Responsive and user-friendly UI.
* Proper state management with React hooks.
* Deployment on Vercel with working routes.

In addition, I added extra touches like the progress bar, accessibility improvements, and a unique design to make the app stand out. calculation

- Detailed results summary
- Responsive and user-friendly UI
- Proper state management with React hooks
- Deployment on Vercel with working routes

In addition, I added extra touches like the progress bar, accessibility improvements, and a unique design to make the app stand out.