Q:

give me process of createing Develop a reusable QuestionCard React component to dynamically fetch and display test questions. This component will serve as the primary interface for test takers to view their questions.

Answer:

Certainly! Below is a step-by-step approach to developing a reusable QuestionCard React component that dynamically fetches and displays test questions. This approach aims to provide an optimized solution suitable for competitive programming test cases.

**Step-by-Step Approach:**

1. **Component Design:**
   * Define the QuestionCard component structure, including necessary states and props.
   * Include placeholders for question content, answer options, and additional functionalities (like navigation).
2. **State Management:**
   * Utilize React’s useState hook to handle dynamic data such as the question text, options, and loading state.
   * Consider using a state management library (like Redux) if the application has complex state requirements or multiple interacting components.
3. **Props Definition:**
   * Define the expected props for the QuestionCard component. Common props include questionId, onAnswerSelect, and fetchQuestion.
   * Use PropTypes or TypeScript for prop type validation to ensure robustness.
4. **Fetching Data:**
   * Use React's useEffect hook to fetch question data when the component mounts or when the questionId prop changes.
   * Create an asynchronous function to fetch the question data from an API.
   * Handle loading, success, and error states appropriately to provide feedback to the user.
5. **Rendering Questions:**
   * Ensure the component renders a placeholder or loading animation while fetching data.
   * Once data is fetched, render the question text and the answer options.
   * Ensure answer options are interactive, allowing users to select their choice.
6. **Handling User Interaction:**
   * Implement an event handler for when an answer option is selected.
   * Invoke the onAnswerSelect callback prop, passing the selected answer back to the parent component.
   * Optionally, manage the user's current selection within the component’s state for enhanced user experience.
7. **CSS and Styling:**
   * Use CSS or a styling library like styled-components to style the QuestionCard.
   * Ensure the component is responsive and visually appealing.
8. **Optimizations:**
   * Prevent unnecessary re-renders by using React.memo for the QuestionCard component if props remain unchanged.
   * Use useCallback and useMemo hooks to optimize functions and values passed down to child components.
   * Implement pagination or batch loading of questions if fetching a large dataset to reduce load times.
9. **Testing:**
   * Write unit and integration tests to ensure the QuestionCard component works correctly.
   * Use testing libraries like Jest and React Testing Library to simulate user interactions and verify component outputs.
10. **Documentation:**
    * Document the component’s API clearly for future developers.
    * Include usage examples, prop descriptions, and notes on fetching logic.

By following these steps, you should be able to create an efficient and reusable QuestionCard React component that dynamically fetches and displays test questions. This approach ensures optimization for performance and usability, making it suitable for competitive programming scenarios.