**MODULE – 6**

**Assignment 11: Connect React with Flask**

**Objective**

Integrate the React frontend with the Flask backend by fetching test questions from the backend API and dynamically displaying them on the frontend.

To effectively tackle Assignment 11, follow these step-by-step instructions to ensure seamless integration between the React frontend and the Flask backend, while optimizing the solution:

**Step-by-Step Instructions:**

**1. API Integration**

* **Setup Axios:**
  + Install Axios in your React project using npm.
  + Create an Axios instance for consistent configuration.
* **Fetch Test Questions:**
  + Use the created Axios instance to make a GET request to /api/speaking-tests endpoint to fetch the test questions.

**2. Frontend Components**

* **Update Existing Components:**
  + Identify the components responsible for displaying test questions.
  + Ensure these components can accept dynamic data as props or state.
* **Render Test Data:**
  + Dynamically render question text and category. Use JSX to map over the fetched data and display it.

**3. State Management**

* **Manage State with React Hooks:**
  + Use useState to create state variables for storing fetched questions and loading/error states.
  + Use useEffect for side effects such as data fetching on component mount.
  + Utilize efficient state management practices to minimize unnecessary re-renders.

**4. Error Handling**

* **Graceful Error Handling:**
  + Implement error handling in the API request. Use try-catch blocks or Axios’ .catch() method.
  + Display error messages to the user appropriately. Consider implementing retry logic for transient errors.
* **Provide User Feedback:**
  + Create user feedback mechanisms such as loading indicators while data is being fetched and error messages when the fetch fails.

**5. Styling**

* **Visual Appeal:**
  + Align the display of questions with the platform’s existing UI design.
  + Ensure the component’s styling is responsive and user-friendly. Utilize CSS or a styling library like Styled-Components or CSS-in-JS.

**6. Testing and Evidence**

* **Test Integration:**
  + Ensure thorough testing of the API integration and dynamic rendering.
  + Capture screenshots or screen recordings as evidence of successful integration.
* **Setup Process Documentation:**
  + Document the steps required to set up and run your integrated React application.

**7. Optimize the Approach**

* **Efficient Data Fetching:**
  + Optimize data fetching by avoiding redundant API calls. Use dependency arrays in useEffect correctly to control when data fetches happen.
* **Minimize State Updates:**
  + Ensure only necessary state updates occur, reducing component re-renders and improving performance.
* **Error Handling Strategy:**
  + Develop a robust error handling strategy to manage different types of errors (network issues, server errors, client errors).
* **Maintainability and Scalability:**
  + Write clean, modular code to improve readability and maintainability. Structure components and functions for easy scalability.

**8. Submission Guidelines**

* **Submit React Files:**
  + Provide the updated React files with API integration.
* **Provide Description and Evidence:**
  + Include a detailed description of the setup process.
  + Attach evidence of successful API integration and data rendering, such as screenshots or screen recordings.

**Key Considerations:**

* Consistency between the frontend and backend data structures.
* Component reusability and modularity.
* Clear and responsive UI/UX.
* Comprehensive error handling and user feedback.

By systematically following these steps, you will ensure that the integration is seamless, performant, and aligns with the project requirements.