### MODULE - 6

**Assignment 12: Implement Test Submission**

**Objective**

Create a form in the React frontend to collect test responses from users and submit the data to the backend via a POST /submit API endpoint.

### Step-by-Step Approach

1. **Form Design**:
   1. **Create the Form Structure**:
      1. Start by designing the form structure utilizing HTML elements in JSX within your React component.
      2. Include necessary fields:
         1. **Question ID** - For identification of the question.
         2. **User Response** - For the test taker's response.
         3. **User ID** - Optional, for identifying the user.
         4. **Comments** - Optional, for any additional remarks.
   2. **Clear and User-Friendly Interface**:
      1. Use labels and placeholders for each input field to ensure users understand what information they need to provide.
      2. Make sure the form has proper layouts and spacing for readability.

**2 .State Management**:

1. **Initialize State**:

- Use useState hook to initialize state variables for each input field, as well as submission status and any potential error messages.

- Example state elements:

* + - 1. questionID
      2. userResponse
      3. userID (optional)
      4. comments (optional)
      5. submissionStatus
      6. errorMessage

1. **Update State on Input Change**:
   * 1. Define handler functions that update the respective state variables whenever the user inputs data into the form.

**3 . Form Validation**:

* 1. **Ensure Required Fields are Filled**:
     1. Implement validation logic within the form submission handler:

- Ensure questionID and userResponse are not empty.

- Optionally check for valid data formats if required.

4 . **API Integration**:

1. **Prepare Data for Submission**:

- Collect the form data into an object when the user submits the form.

* 1. **Send Data via POST Request**:

- Use the fetch or axios library to send a POST request to the /submit API endpoint with the collected form data.

* 1. **Handle API Response**:

- Validate the API response:

* + - 1. If successful, update the submissionStatus state to indicate success and optionally provide a success message.
      2. If unsuccessful, handle errors by updating the errorMessage state with the relevant error message.

**5 . Error Handling**:

**1 . Display Relevant Error Messages**:

- If the API call fails, display a user-friendly error message to inform the test taker.

- If required fields are missing, prompt the user to fill them appropriately before submission.

**2 . Feedback for Successful Submissions**:

- Upon successful form submission, inform the user of the successful submission through a message or visual indicator.

- Optionally, redirect or clear the form fields.

**6 . Styling**:

* 1. **Ensure Responsiveness and Accessibility**:

- Style the form to be user-friendly and responsive.

- Use CSS or styled-components to ensure the form adapts to various screen sizes.

- Provide visual feedback (e.g., highlighting empty required fields) to improve user experience.

**7 . Validation and Testing**:

* 1. **Test the Form Functionality**:

- Ensure the form works as expected by testing different input scenarios and submission flows.

- Validate that input data is correctly transmitted to the backend.

- Confirm that error messages and success feedback are correctly displayed in all scenarios.

**8 . Evidence of Integration**:

* 1. **Capture Screenshots or Recordings**:

- Provide evidence by capturing screenshots or recording videos demonstrating form submission, error handling, and API response handling.

* 1. **Comprehensive Submission**:

- Submit the React form component file along with the backend API details (if available).

- Include a description of how you tested the form and any evidence of functionality.

### Optimizations for Competitive Programming

* **Optimize State Management**:
  + Efficiently manage state to avoid unnecessary re-renders which can slow down the form.
* **Ensure Fast API Handling**:
  + Implement a loading state to handle API calls efficiently and give feedback to the user.
* **Error Handling**:
  + Use optimized error checking mechanisms to quickly identify and handle errors, providing immediate feedback.