Test 1**:** Create a RESTful API endpoint using Python and the Flask framework for a simple user feedback mechanism.

Here's the technical information you may need**:**

1. **API Endpoint Creation:**
    - Develop a POST API endpoint at the path /api/feedback.
    - The endpoint should accept feedback submissions from users.
    - The request body for submitting feedback **must be in JSON format**.
2. **Feedback Mechanism Details:** The feedback submission should include the following data points:
    - **a. Overall Rating:**
        - An integer value representing an overall rating from 1 to 5 (1 = poor, 5 = excellent).
        - **This field is required.**
    - **b. Opinion Text Box:**
        - A text string for the user to write their opinion.
        - Maximum length of 500 characters.
        - **This field is optional.**
    - **c. Research Email List Opt-in:**
        - Users are asked if they are interested in signing up for a research email list, with two options:
            - **i. "Not interested":** If selected, no further information is required for this option.
            - **ii. "Interested, sign me up":** If selected, the user **must provide a valid email address**.
    - **d. Submission Method:** The entire feedback form (all the above data) will be sent in a single POST request.
3. **Data Storage:**
    - For this test, you can store the submitted feedback data **in-memory** (e.g., a Python list of dictionaries) or in a **simple flat file** (e.g., a JSON file) for basic persistence between server restarts. A more robust database integration is not required for this part of the test.
4. **Error Handling and Validation:**
    - Implement robust error handling and input validation.
    - If a required field is missing or invalid (e.g., rating outside 1-5, email invalid when "interested", opinion exceeding 500 characters), return a 400 Bad Request HTTP status code.
    - The error response should include a clear, human-readable, and descriptive error message indicating the specific validation failure.
    - If the request is successful, return a 201 Created HTTP status code (or 200 OK) and a success message.
5. **Documentation and Approach:**
    - **Outline your approach** and **document your process** directly within the same Python (.py) component. This includes:
        - Briefly explaining your setup.
        - Describing your validation logic.

    - Any assumptions made.
  - Utilize **docstrings** for functions and classes, and add clear **comments** where the logic might be complex or require further explanation.

Test 2: Integrate the provided React components for the feedback mechanism with the backend API endpoint developed in Part 1. The goal is to send the collected feedback data to the backend when the user submits the form.

Provided Frontend Components:
You are provided with the following React components for the feedback page:

- **FeedBackPage.jsx**: The main container component that manages the multi-step feedback form and holds the overall feedbackData state.
- **FeedBackInterest.jsx**: Manages the "interested in research" option and email input.

Requirements for Integration:

1. **API Call on Submission:**
    - Modify the handleFinish function accordingly.
    - The feedbackData should now **make an HTTP POST request** to the backend API endpoint you created (/api/feedback).
    - The request body should contain the feedbackData in **JSON format**, adhering to the structure defined for the backend (rating, improvementText, interestedInResearch, email).
2. **Error Handling for API Calls** (you may use react-toastify)**:**
    - Implement client-side error handling for the API request.
    - If the API call fails (e.g., network error, 400 Bad Request, 500 Internal Server Error), **display a user-friendly error message** on the frontend.
    - Consider how to handle specific validation errors returned by the backend (e.g., if the backend returns an error indicating an invalid email, display that specific message to the user). You might choose to prevent submission if client-side validation catches issues first.
3. **Loading State:**
    - Implement a **loading state** to provide visual feedback to the user while the API request is in progress. This could involve disabling the "Submit Feedback" button or showing a spinner.
4. **Success State:**
    - Upon successful submission, the frontend should transition to the FeedBackThanks component as currently implemented.
5. **Technology Stack:**
    - You must use **React** for the frontend.
    - You can use **TypeScript** for type safety if you prefer, but it's not strictly required for this part, as the provided components are in JavaScript. However, a good applicant would consider adding types.
    - For making API requests, you are free to choose between:
        - The built-in **fetch API**.
        - A third-party library (**axios**)
6. **Code Organization and Documentation:**
    - Clearly outline your approach for integrating the API.
    - Document your changes with comments, especially in the FeedBackPage.jsx component where the API call will be made.

    - Ensure your code is clean, readable, and follows common React/TypeScript best practices.

Additional Context for the Applicant:

- **Backend API Endpoint:** Remember that the backend endpoint you created listens for `POST` requests at `/api/feedback`.
- **Data Structure:** The JSON payload sent to the backend should match the fields expected by your Python API:
  JSON
  ```json
  {
    "rating": 4,
    "improvementText": "The UI could be more intuitive on mobile.",
    "interestedInResearch": true,
    "email": "user@example.com"
  }
  ```
  (Note: `improvementText` and `email` might be empty strings or omitted if optional and not provided/not interested respectively, depending on your backend's exact handling of optional fields).