1. Features for the Dynamic Form Generator

The **Dynamic Form** should be capable of handling a variety of field types as defined in a JSON schema. Here's an outline of the required features:

Field Types Supported from the Schema

- **Text** (input type text)
- **Number** (input type number)
- **Email** (input type email)
- **Password** (input type password)
- **Checkbox** (checkbox input)
- **Radio** (radio button)
- **Select** (dropdown list)
- **Textarea** (textarea input)
- **Date** (input type date)
- **File** (input type file)

Each field type will be dynamically rendered based on the schema provided.

Validation Messages

- Each field will have corresponding validation rules (e.g., required, minLength, maxLength, email format, etc.).
- **React Hook Form** will be used to handle form validation. If a user fails to meet a validation condition, an error message will be displayed below the respective field.

Loading States

- While the form is being generated (for example, when fetching the schema or submitting the form), show a loading spinner or placeholder content.

Submit Data to console.log()

- On form submission, the form data should be logged to the console for inspection.

Success Message After Submission

- Upon successful form submission, a success message like "Form submitted successfully!" should appear.

Consistent Styling with Tailwind CSS

- All form fields, buttons, and error messages should be styled using **Tailwind CSS** for consistency.

2. Technical Stack Overview

React 18+:

- Build the user interface components using React (latest version).

TypeScript:

- Use TypeScript for type safety, ensuring that form fields and validation data are structured correctly.

Tailwind CSS:

- For styling, **Tailwind CSS** will be used to quickly apply responsive and consistent styles.

React Hook Form:

- This library will be used for managing form state and validation. It provides built-in support for handling complex forms with minimal boilerplate code.

Playwright and Jest for Testing:

- **Playwright** will be used for **E2E testing** to simulate user interactions, form validation, and submission.
- **Jest** will be used for **unit testing** to verify individual functions like validation logic and form rendering.