1. Setup the Project\*

1. Initialize the project using React and TypeScript:

bash

npx create-react-app dynamic-form-generator --template typescript

cd dynamic-form-generator

2. Install necessary dependencies:

bash

npm install tailwindcss postcss autoprefixer react-hook-form react-syntax-highlighter @hookform/resolvers yup

npm install --save-dev jest @testing-library/react @testing-library/jest-dom playwright

3. Configure Tailwind CSS:

bash

npx tailwindcss init

Add the following to tailwind.config.js:

javascript

module.exports = {

content: ["./src/\*\*/\*.{js,jsx,ts,tsx}"],

theme: {

extend: {},

},

plugins: [],

};

Include Tailwind in index.css:

css

@tailwind base;

@tailwind components;

@tailwind utilities;

---

\*2. Main Interface\*

1. Create a split-screen layout:

- Use flex with Tailwind for a split-screen UI.

- Add a JSON editor (left) and a form preview (right).

jsx

const App = () => {

return (

<div className="flex h-screen">

<div className="w-1/2 border-r p-4">

<h1 className="text-xl font-bold">JSON Editor</h1>

<textarea

id="json-editor"

className="w-full h-full p-2 border rounded"

placeholder="Enter your JSON schema here..."

></textarea>

</div>

<div className="w-1/2 p-4">

<h1 className="text-xl font-bold">Form Preview</h1>

<div id="form-preview" className="p-4 border rounded"></div>

</div>

</div>

);

};

export default App;

2. Add syntax highlighting and JSON validation:

- Use react-syntax-highlighter for syntax highlighting.

- Use a library like yup for JSON schema validation.

3. Implement a mobile-responsive layout:

- Add a @media query using Tailwind (sm, md, lg) to stack editor and preview on smaller screens.

---

\*3. JSON Schema Handling\*

1. Define a TypeScript interface for the schema:

typescript

interface FormField {

id: string;

type: string;

label: string;

required: boolean;

placeholder?: string;

options?: { value: string; label: string }[];

validation?: { pattern: string; message: string };

}

interface FormSchema {

formTitle: string;

formDescription: string;

fields: FormField[];

}

2. Add a JSON parser and error handling:

- Use try-catch for JSON parsing errors.

- Display error messages inline.

---

\*4. Dynamic Form Generation\*

1. Use React Hook Form for form state and validation:

tsx

import { useForm } from "react-hook-form";

const DynamicForm = ({ schema }: { schema: FormSchema }) => {

const { register, handleSubmit, formState: { errors } } = useForm();

const onSubmit = (data: any) => {

console.log(data);

alert("Form submitted successfully!");

};

return (

<form onSubmit={handleSubmit(onSubmit)} className="space-y-4">

{schema.fields.map((field) => {

switch (field.type) {

case "text":

case "email":

return (

<div key={field.id}>

<label>{field.label}</label>

<input

type={field.type}

{...register(field.id, {

required: field.required,

pattern: field.validation?.pattern

? new RegExp(field.validation.pattern)

: undefined,

})}

placeholder={field.placeholder}

className="w-full p-2 border rounded"

/>

{errors[field.id] && (

<p className="text-red-500 text-sm">

{field.validation?.message || "This field is required"}

</p>

)}

</div>

);

case "select":

return (

<div key={field.id}>

<label>{field.label}</label>

<select

{...register(field.id, { required: field.required })}

className="w-full p-2 border rounded"

>

{field.options?.map((option) => (

<option key={option.value} value={option.value}>

{option.label}

</option>

))}

</select>

{errors[field.id] && (

<p className="text-red-500 text-sm">This field is required</p>

)}

</div>

);

default:

return null;

}

})}

<button type="submit" className="px-4 py-2 bg-blue-500 text-white rounded">

Submit

</button>

</form>

);

};

---

\*5. Testing\*

1. Use Jest for unit tests:

- Test JSON validation logic.

- Test form generation based on schema.

2. Use Playwright for E2E tests:

- Test form submission and success messages.

- Test error messages for invalid JSON and invalid form inputs.

---

\*6. Bonus Features\*

1. \*Copy Form JSON\*:

Add a button to copy the schema using navigator.clipboard.writeText.

2. \*Validation Preview\*:

Show live validation messages for each field while editing.

3. \*Dark Mode\*:

Use Tailwind’s dark mode configuration.

4. \*Download Form Data\*:

Convert form submission data to JSON and download using Blob.

---

\*Deployment\*

1. Deploy to Vercel or Netlify:

bash

npm run build

npx vercel --prod

2. Include a README.md with:

- Setup instructions.

- Example JSON schemas.

- Local development guide.