**Dynamic-Form-Generator**

**dynamic-form-generator/**

**│**

**├── public/**

**│ └── index.html # Main HTML file, entry point for the app.**

**│**

**├── src/**

**│ ├── assets/ # Store images, fonts, or other static assets.**

**│ ├── components/ # Reusable React components**

**│ │ ├── FormField.tsx # FormField component for rendering dynamic form fields.**

**│ │ ├── FormPreview.tsx # Component to preview the form.**

**│ │ └── JSONEditor.tsx # Component to edit JSON for form generation.**

**│ │**

**│ ├── utils/ # Utility functions or helpers**

**│ │ └── TSValidateJSON.ts # Validate JSON format.**

**│ │**

**│ ├── interfaces/ # TypeScript interfaces for type safety.**

**│ │ └── FormTypes.ts # Interfaces for form data, fields, etc.**

**│ │**

**│ ├── App.tsx # Main App component.**

**│ ├── index.tsx # React entry point, renders App component.**

**│ ├── index.css # Global styles, possibly including Tailwind.**

**│ └── setupTests.js # Setup for testing framework.**

**│**

**├── .gitignore # Git ignore file for excluded files/folders.**

**├── package.json # Project metadata, dependencies, scripts.**

**├── package-lock.json # Lock file for consistent dependency versions.**

**├── postcss.config.js # Configuration for PostCSS (Tailwind CSS).**

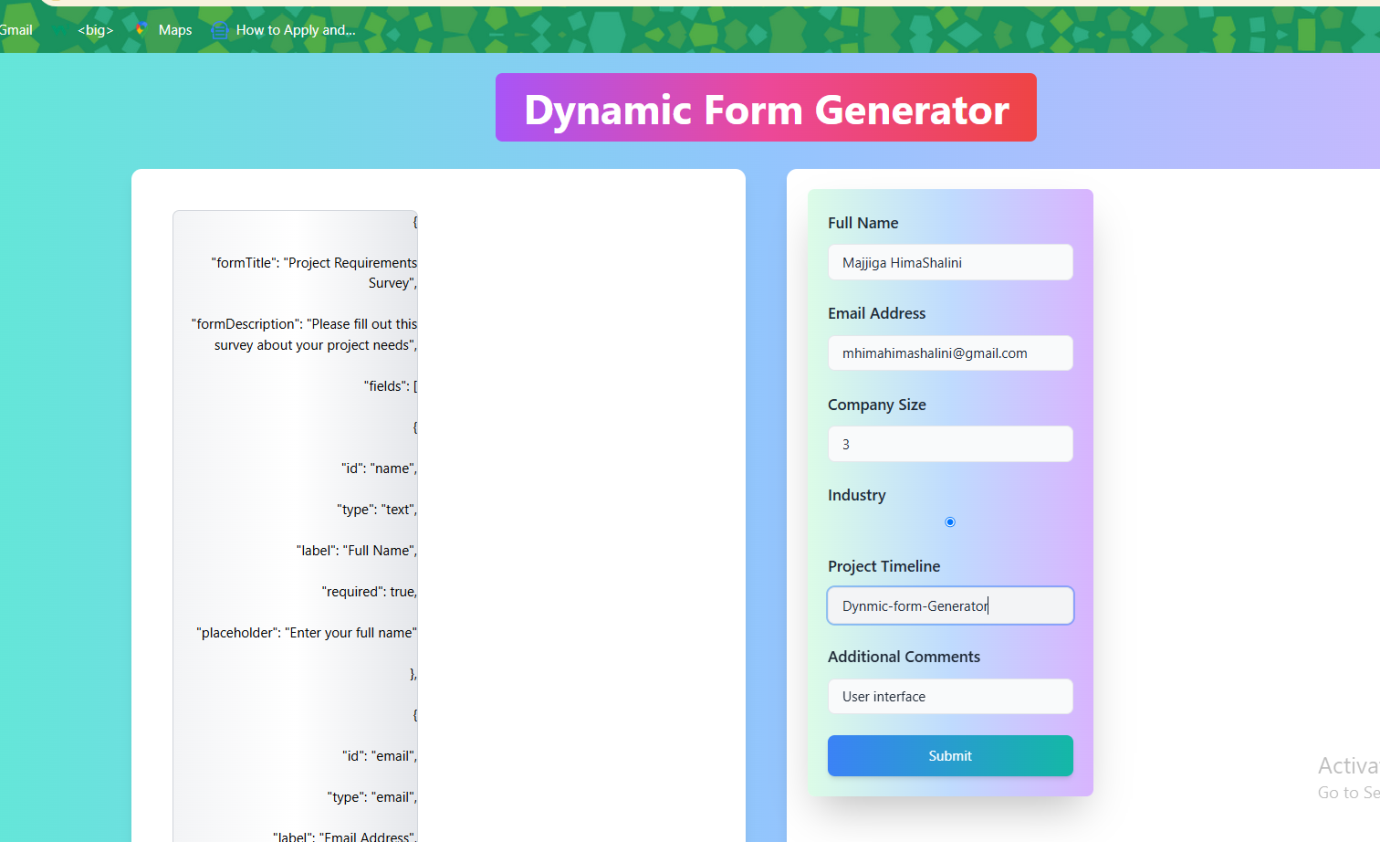
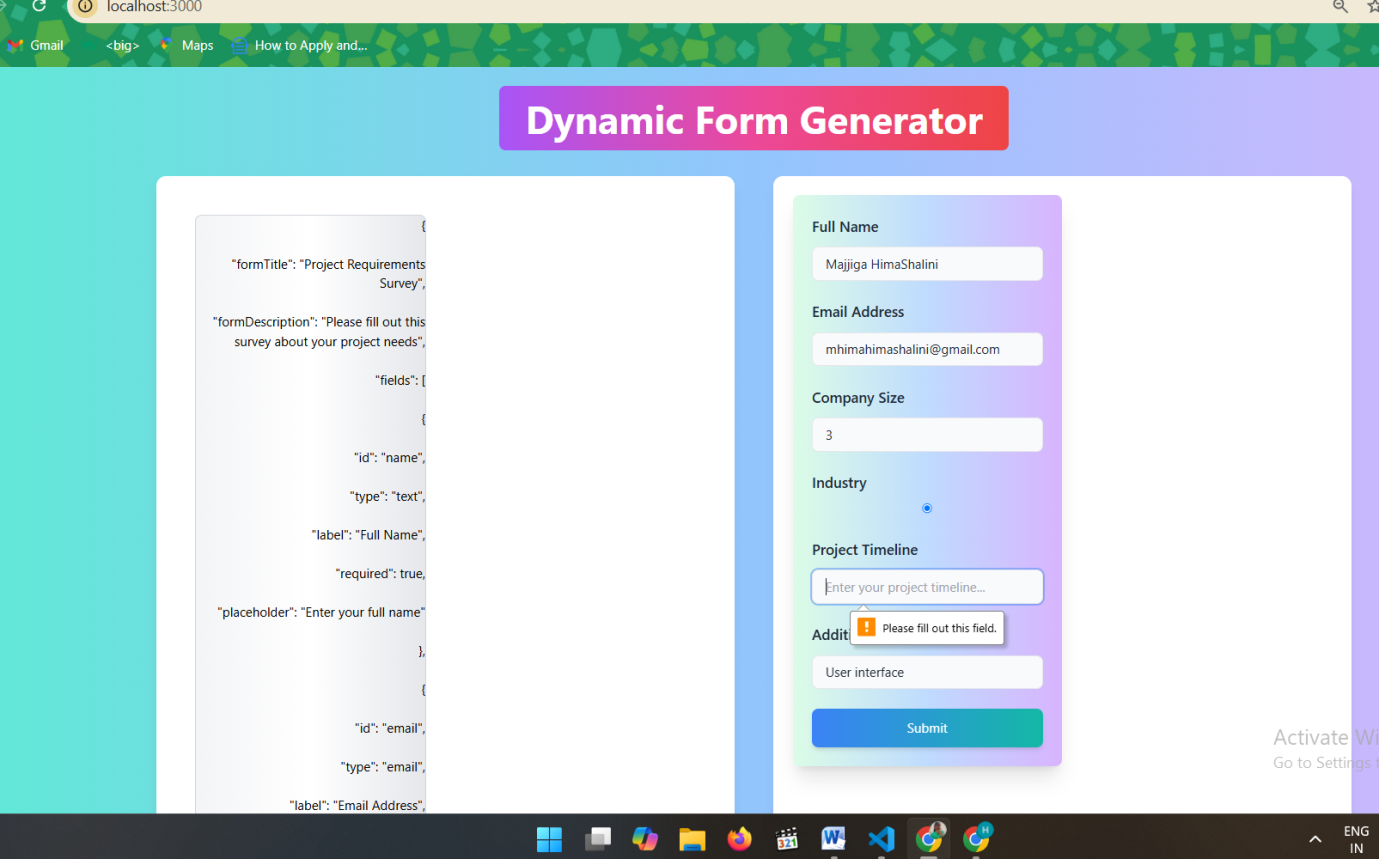
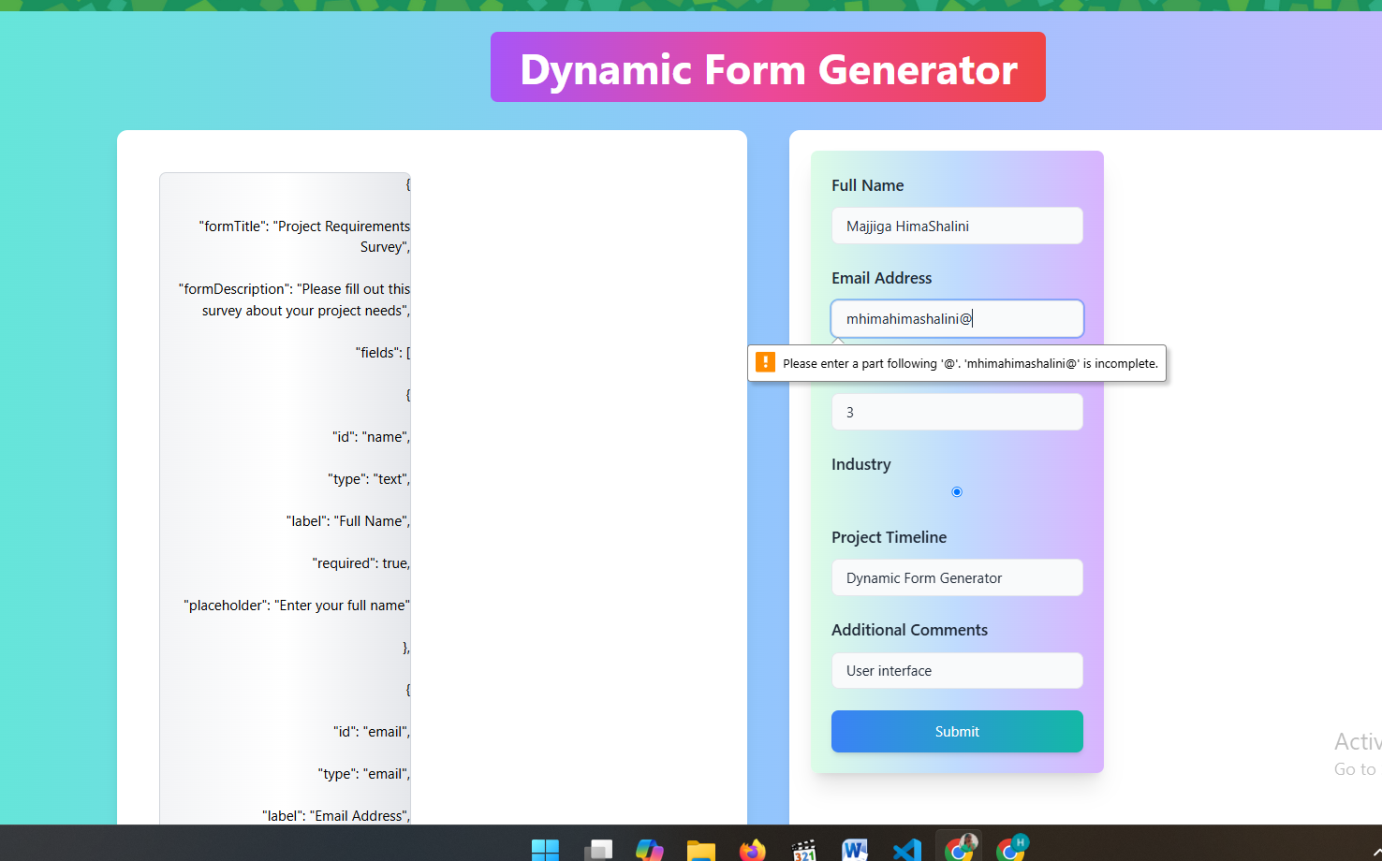
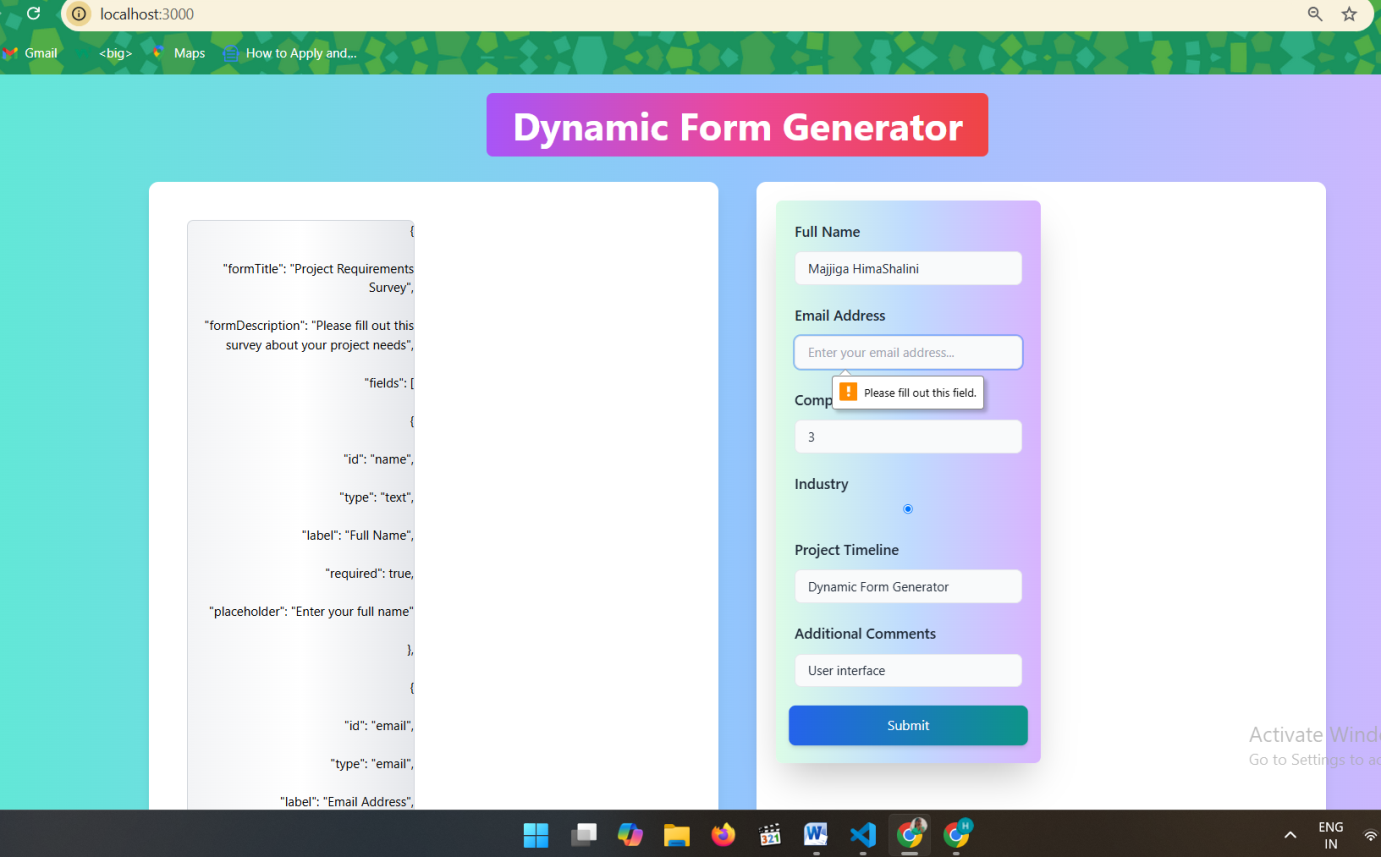
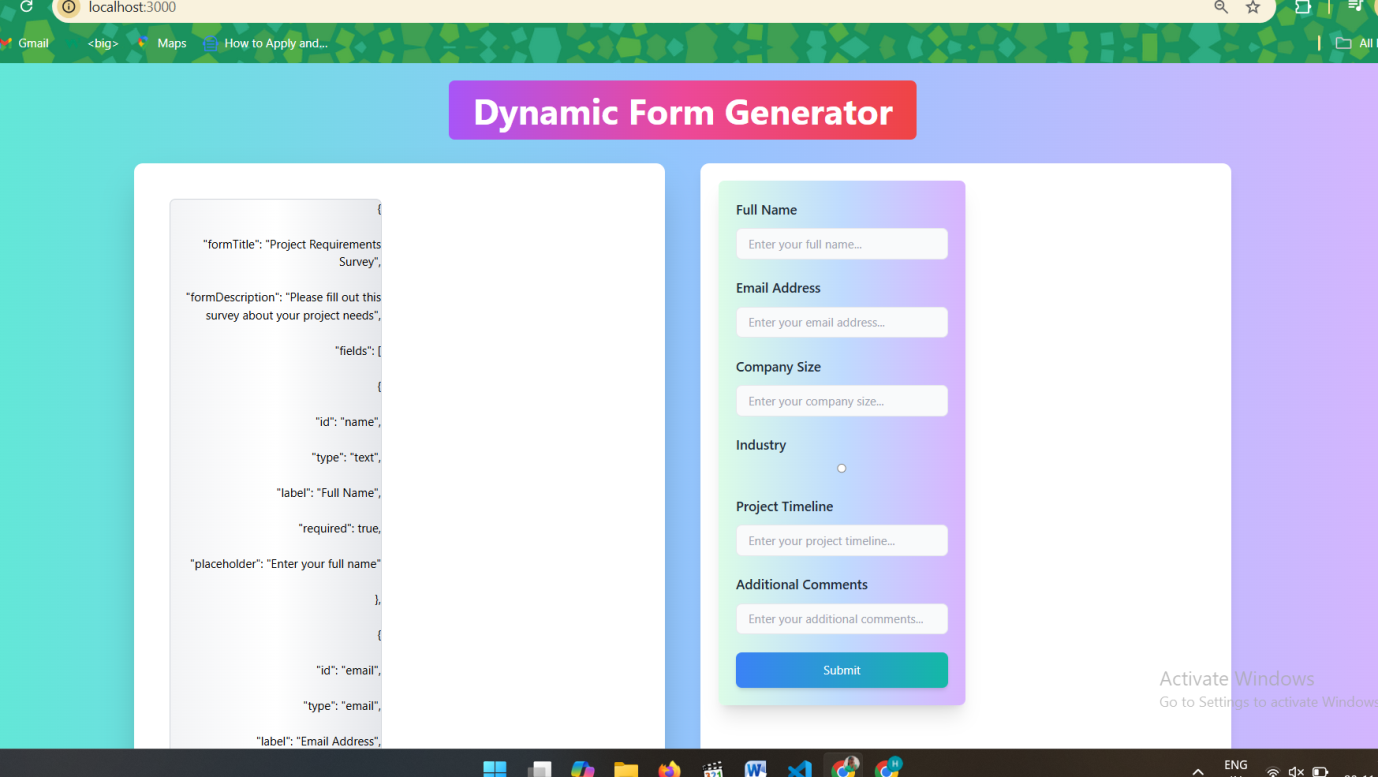
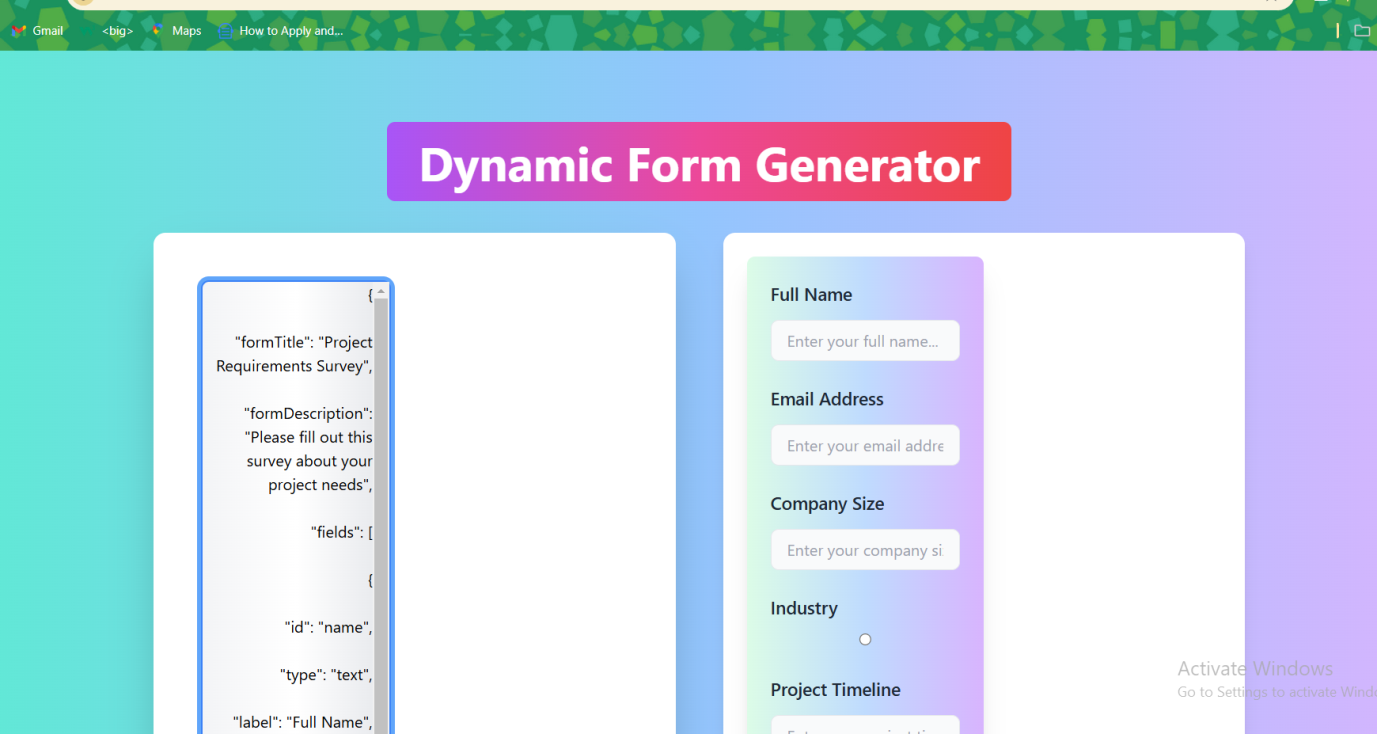
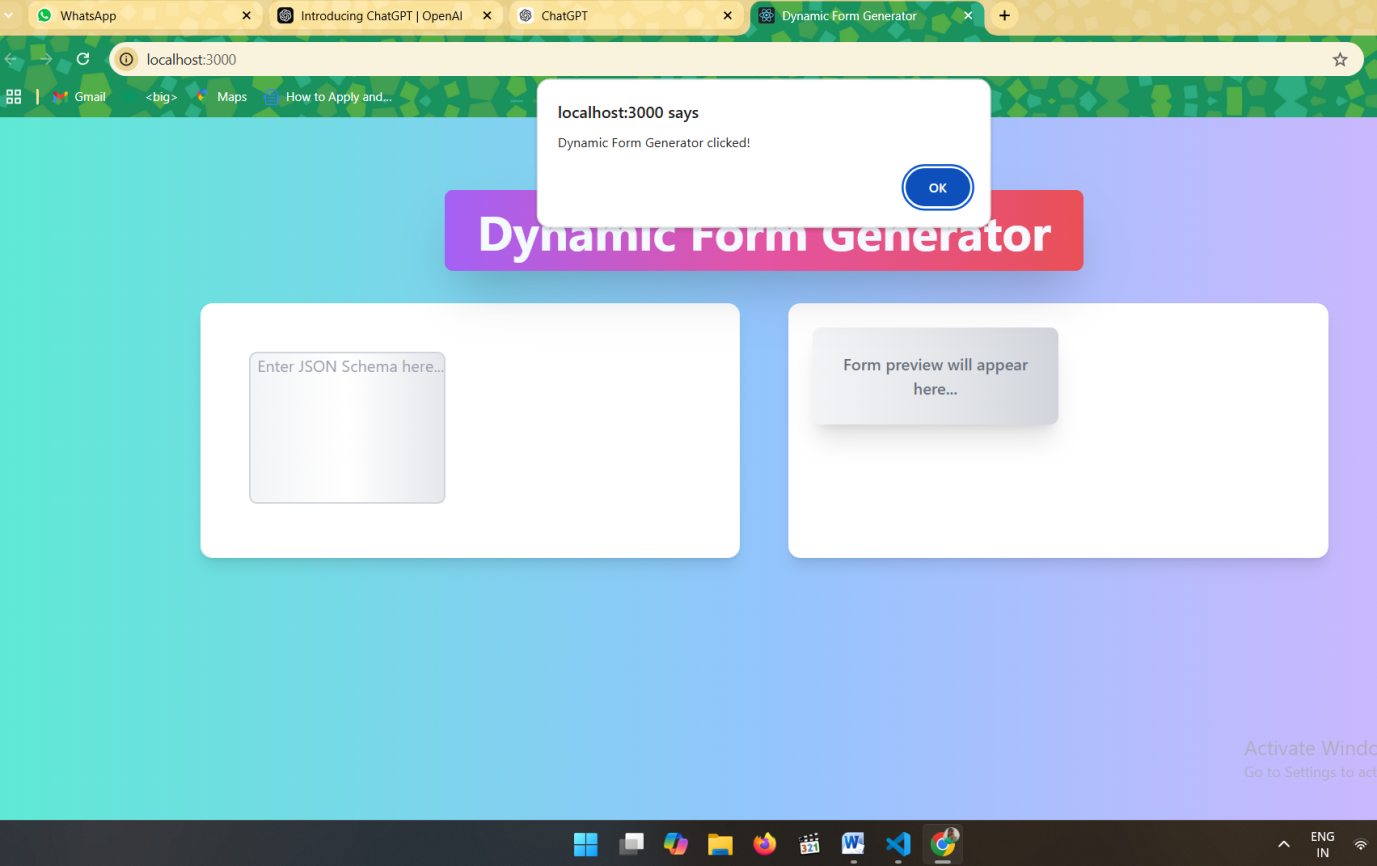
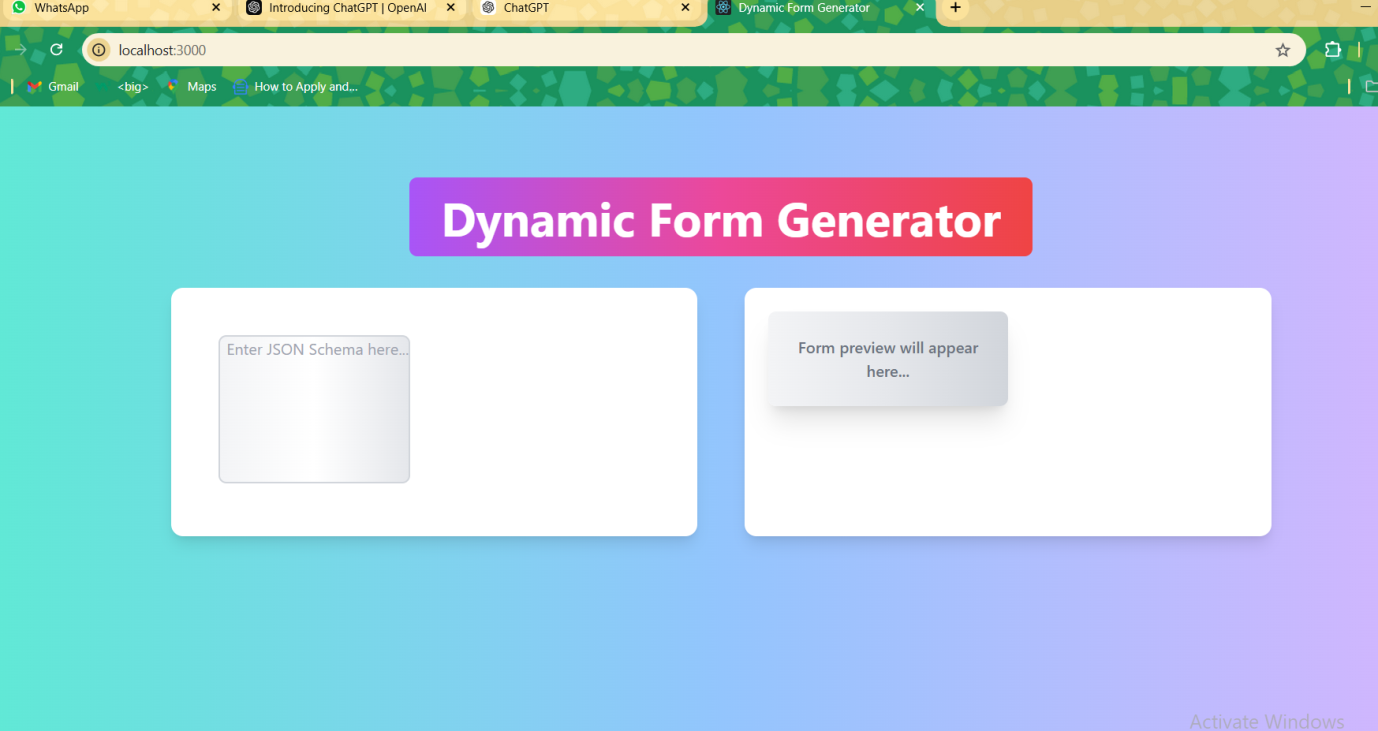
**├── tailwind.config.js # Tailwind CSS configuration.**

**├── tsconfig.json # TypeScript configuration.**

**├── README.md # Project description and setup instructions.**

**└── node\_modules/ # Installed dependencies.**

**Outputs:-**

****

**Public/index.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>Dynamic Form Generator</title>

<script src="https://cdn.tailwindcss.com"></script>

<style>

body

{

background-color: gray;

}

</style>

</head>

<body>

<div id="root"></div>

</body>

</html>

**Src/Components/FormField.tsx**

import React from 'react';

interface FormFieldProps {

field: {

label: string;

type: string;

name: string;

required?: boolean;

};

}

const FormField: React.FC<FormFieldProps> = ({ field }) => {

return (

<div className="space-y-2">

{/\* Label \*/}

<label className="block text-lg font-medium text-gray-800 mb-1">{field.label}</label>

{/\* Input Field \*/}

<input

type={field.type}

name={field.name}

required={field.required}

className="w-full px-4 py-2 border rounded-lg bg-gray-50 text-gray-800 placeholder-gray-400 focus:outline-none focus:ring-2 focus:ring-blue-500 focus:ring-opacity-50 transition duration-300 ease-in-out hover:bg-gray-100"

placeholder={`Enter your ${field.label.toLowerCase()}...`}

/>

</div>

);

};

export default FormField;

**Src/Components/FormPreview.tsx**

import React from 'react';

import FormField from './FormField';

interface FormPreviewProps {

schema: any;

}

const FormPreview: React.FC<FormPreviewProps> = ({ schema }) => {

const handleSubmit = (e: React.FormEvent) => {

e.preventDefault();

const formData = new FormData(e.target as HTMLFormElement);

// Manually create the data object

const data: { [key: string]: string } = {};

formData.forEach((value, key) => {

data[key] = value as string;

});

console.log(data);

alert('Form submitted successfully!');

};

if (!schema) {

return (

<div className="w-full lg:w-1/2 p-6 bg-gradient-to-r from-gray-100 via-gray-200 to-gray-300 rounded-lg shadow-xl">

<p className="text-center text-gray-500 font-semibold">Form preview will appear here...</p>

</div>

);

}

return (

<form className="w-full lg:w-1/2 p-6 bg-gradient-to-r from-green-100 via-blue-200 to-purple-300 rounded-lg shadow-xl space-y-6 transition-all duration-300 hover:shadow-2xl">

{schema.fields?.map((field: any, index: number) => (

<FormField key={index} field={field} />

))}

<button

type="submit"

className="w-full bg-gradient-to-r from-blue-500 to-teal-500 text-white py-3 px-6 rounded-lg shadow-md hover:from-blue-600 hover:to-teal-600 transition duration-300 ease-in-out transform hover:scale-105"

>

Submit

</button>

</form>

);

};

export default FormPreview;

**Src/Components/JSONEditor.tsx**

import React, { useRef, useEffect } from 'react';

interface JSONEditorProps {

value: string;

onChange: (value: string) => void;

error: string | null;

}

const JSONEditor: React.FC<JSONEditorProps> = ({ value, onChange, error }) => {

const textareaRef = useRef<HTMLTextAreaElement | null>(null);

// Auto-resize the textarea when the content changes (vertical resizing only)

useEffect(() => {

if (textareaRef.current) {

textareaRef.current.style.height = 'auto'; // Reset the height before recalculating

textareaRef.current.style.height = `${textareaRef.current.scrollHeight}px`; // Set height based on content

}

}, [value]);

return (

<div className="w-full lg:w-1/2 p-6">

<textarea

ref={textareaRef}

className="w-full min-h-[150px] border-2 border-gray-300 rounded-lg bg-gradient-to-r from-gray-100 via-white to-gray-200 focus:outline-none focus:ring-4 focus:ring-blue-400 focus:border-blue-500 resize-none overflow-x-auto text-right scrollbar-thin scrollbar-thumb-blue-400 scrollbar-track-gray-100"

value={value}

onChange={(e) => onChange(e.target.value)}

placeholder="Enter JSON Schema here..."

/>

{error && <p className="text-red-500 mt-2 text-sm">{error}</p>}

</div>

);

};

export default JSONEditor;

**Src/Utils/ValidateJSON.ts**

export const validateJSON = (json: string): boolean => {

try {

JSON.parse(json);

return true;

} catch {

return false;

}

};

**App.tsx**

import React, { useState } from 'react';

import JSONEditor from './components/JSONEditor';

import FormPreview from './components/FormPreview';

const App: React.FC = () => {

const [jsonSchema, setJsonSchema] = useState<string>('');

const [parsedSchema, setParsedSchema] = useState<any>(null);

const [error, setError] = useState<string | null>(null);

const handleSchemaChange = (newSchema: string) => {

setJsonSchema(newSchema);

try {

const parsed = JSON.parse(newSchema);

setParsedSchema(parsed);

setError(null);

} catch (err: any) {

setParsedSchema(null);

setError('Invalid JSON format');

}

};

return (

<div className="flex flex-col items-center min-h-screen p-6 bg-gradient-to-r from-teal-300 via-blue-300 to-purple-300">

{/\* Title with Gradient Background, Hover Effect, and Cursor Pointer \*/}

<h1

className="text-4xl lg:text-5xl font-bold text-white bg-gradient-to-r from-purple-500 via-pink-500 to-red-500 px-8 py-4 rounded-lg cursor-pointer hover:opacity-90 hover:shadow-2xl transition duration-300 text-center mb-8 mt-12"

onClick={() => alert('Dynamic Form Generator clicked!')}

>

Dynamic Form Generator

</h1>

{/\* Responsive Layout for Form Preview and JSON Editor \*/}

<div className="flex flex-col lg:flex-row w-full lg:w-3/4 gap-8 lg:gap-12 mb-12">

{/\* JSON Editor Section \*/}

<div className="w-full lg:w-1/2 bg-white p-6 rounded-xl shadow-lg transition-all duration-500 hover:shadow-2xl">

<JSONEditor

value={jsonSchema}

onChange={handleSchemaChange}

error={error}

/>

</div>

{/\* Form Preview Section \*/}

<div className="w-full lg:w-1/2 bg-white p-6 rounded-xl shadow-lg">

<FormPreview schema={parsedSchema} />

</div>

</div>

</div>

);

};

export default App;

**index.css**

@tailwind base;

@tailwind components;

@tailwind utilities;

**Index.tsx**

import React from 'react';

import ReactDOM from 'react-dom';

import App from './App';

import './index.css';

ReactDOM.render(

<React.StrictMode>

<App />

</React.StrictMode>,

document.getElementById('root')

);

**Interfaces.ts**

export interface Field {

label: string;

type: string;

name: string;

required?: boolean;

}

export interface Schema {

fields: Field[];

}

**setupTests.js**

// jest-dom adds custom jest matchers for asserting on DOM nodes.

// allows you to do things like:

// expect(element).toHaveTextContent(/react/i)

// learn more: https://github.com/testing-library/jest-dom

import '@testing-library/jest-dom';

**Package.json**

{

"name": "dynamic-form-generator",

"version": "0.1.0",

"private": true,

"dependencies": {

"@testing-library/jest-dom": "^5.17.0",

"@testing-library/react": "^13.4.0",

"@testing-library/user-event": "^13.5.0",

"@types/react": "^18.3.12",

"@types/react-dom": "^18.3.1",

"ajv": "^8.17.1",

"autoprefixer": "^10.4.20",

"monaco-editor": "^0.52.0",

"postcss": "^8.4.49",

"react": "^18.3.1",

"react-dom": "^18.3.1",

"react-scripts": "5.0.1",

"tailwindcss": "^3.4.15",

"typescript": "^5.6.3",

"web-vitals": "^2.1.4"

},

"scripts": {

"start": "react-scripts start",

"build": "react-scripts build",

"test": "react-scripts test",

"eject": "react-scripts eject"

},

"eslintConfig": {

"extends": [

"react-app",

"react-app/jest"

]

},

"browserslist": {

"production": [

">0.2%",

"not dead",

"not op\_mini all"

],

"development": [

"last 1 chrome version",

"last 1 firefox version",

"last 1 safari version"

]

}

}

**README.md**

**# Dynamic Form Generator**

This project is a dynamic form generator built with **\*\*React\*\***, **\*\*TypeScript\*\***, and **\*\*Tailwind CSS\*\***. It allows users to create customizable forms based on dynamic JSON data, preview the generated form, and edit the JSON configuration.

**## Features**

- **\*\*Dynamic Form Generation\*\***: Based on user-provided JSON configurations.

- **\*\*Form Preview\*\***: Allows users to preview the form before submission.

- **\*\*JSON Editor\*\***: Enables users to edit the form's JSON configuration.

- **\*\*Tailwind CSS\*\***: A utility-first CSS framework for styling the components.

- **\*\*TypeScript\*\***: For type safety and better development experience.

**## Project Structure**

dynamic-form-generator/ ├── public/ │ └── index.html # Main HTML file, entry point for the app. ├── src/ │ ├── assets/ # Static assets like images, fonts, etc. │ ├── components/ # React components │ │ ├── FormField.tsx # Renders dynamic form fields. │ │ ├── FormPreview.tsx # Previews the generated form. │ │ └── JSONEditor.tsx # Edits JSON configuration. │ ├── utils/ # Utility functions │ │ └── TSValidateJSON.ts # Validates JSON configuration. │ ├── interfaces/ # TypeScript interfaces │ │ └── FormTypes.ts # Interfaces for form data, fields, etc. │ ├── App.tsx # Main App component. │ ├── index.tsx # React entry point. │ ├── index.css # Global styles. │ └── setupTests.js # Testing setup. ├── .gitignore # Git ignore file. ├── package.json # Project metadata and dependencies. ├── package-lock.json # Lock file for consistent dependencies. ├── postcss.config.js # Tailwind CSS configuration. ├── tailwind.config.js # Tailwind CSS configuration. ├── tsconfig.json # TypeScript configuration. └── README.md # This file.

markdown

Copy code

**## Prerequisites**

Before running this project, make sure you have the following installed:

- **\*\*Node.js\*\***: A JavaScript runtime. Download it from [here](https://nodejs.org/).

- **\*\*npm\*\*** (comes with Node.js): A package manager for JavaScript.

**## Installation**

1. Clone this repository:

```bash

git clone https://github.com/your-username/dynamic-form-generator.git

Navigate to the project directory:

bash

Copy code

cd dynamic-form-generator

Install the dependencies:

bash

Copy code

npm install

Running the App

To start the app in development mode, run:

bash

Copy code

npm start

This will open the app in your default browser, usually at http://localhost:3000.

Building the App

To create a production build of the app, run:

bash

Copy code

npm run build

This will generate a build/ folder with optimized static files for production.

Testing

To run tests, use the following command:

bash

Copy code

npm test

You can add your own tests in the src/**\_\_tests\_\_**/ folder or modify existing ones.

Tailwind CSS Setup

This project uses Tailwind CSS for styling. If you need to customize the styles, you can modify the configuration in the tailwind.config.js file.

Tailwind is set up to process CSS through PostCSS. You can add custom styles in index.css and use Tailwind's utility classes throughout your components.

Contributing

Contributions are welcome! If you find a bug or have a suggestion for improvement, feel free to create an issue or submit a pull request.

How to Contribute:

Fork this repository.

Create a new branch (git checkout -b feature-name).

Make your changes.

Commit your changes (git commit -m 'Add feature').

Push to your branch (git push origin feature-name).

Open a pull request.

License

This project is licensed under the MIT License - see the LICENSE file for details.

Acknowledgements

React: A JavaScript library for building user interfaces.

TypeScript: A superset of JavaScript that adds types.

Tailwind CSS: A utility-first CSS framework for rapid UI development.

css

Copy code

You can copy-paste this content directly into your `README.md` file. It covers the essential aspects of your project and provides a guide for setting up, running, and contributing to the project.

**Tailwind.config.js**

/\*\* @type {import('tailwindcss').Config} \*/

module.exports = {

content: [],

theme: {

extend: {},

},

plugins: [],

}

**Tsconfig.json**

{

"compilerOptions": {

"target": "es5",

"lib": ["dom", "es2015"],

"jsx": "react",

"moduleResolution": "node",

"allowJs": true,

"esModuleInterop": true,  // Add this line

"strict": true,            // Optional: This enables strict type-checking

"skipLibCheck": true,      // Optional: Skips type checking of declaration files

"forceConsistentCasingInFileNames": true  // Optional: Ensures consistent casing in file names

},

"include": [

"src"

]

}