Components

JsonEditor.tsx

tsx

```
import React, { useState } from 'react';
import { json } from 'json-lint';
import { useJsonValidation } from '../services/jsonService';

interface JsonEditorProps {
schema: any;
onChange: (schema: any) => void;
}

const JsonEditor: React.FC<JsonEditorProps> = ({ schema, onChange }) => {
const [jsonError, setJsonError] = useState<string | null>(null);
const { validateJson } = useJsonValidation();

const handleJsonChange = (jsonString: string) => {
try {
const json = JSON.parse(jsonString);
validateJson(json);
onChange(json);
setJsonError(null);
} catch (error) {
setJsonError(error.message);
}
};

return (
```

```
<div>
<textarea
value={JSON.stringify(schema, null, 2)}
onChange={(e) => handleJsonChange(e.target.value)}
/>
{jsonError && <div style={{ color: 'red' }}>{jsonError}</div>}
</div>
);
};
export default JsonEditor;
```

FormGenerator.tsx

tsx

```
import React from 'react';
import { useForm } from 'react-hook-form';
import { useSchema } from '../services/formService';
interface FormGeneratorProps {
schema: any;
}
const FormGenerator: React.FC<FormGeneratorProps> = ({ schema }) => {
const { register, handleSubmit, errors } = useForm();
const { generateForm } = useSchema();
const onSubmit = (data: any) => {
console.log(data);
};
return (
<form onSubmit={handleSubmit(onSubmit)}>
{generateForm(schema, register, errors)}
<button type="submit">Submit</button>
```

```
</form>
);
};
export default FormGenerator;
```

App.tsx

tsx

```
import React, { useState } from 'react';
import JsonEditor from './components/JsonEditor';
import FormGenerator from './components/FormGenerator';
const App: React.FC = () => {
const [schema, setSchema] = useState({});
return (
<div style={{ display: 'flex' }}>
<JsonEditor schema={schema} onChange={setSchema} />
<FormGenerator schema={schema} />
</div>
);
};
export default App;
```

Services:

Json Service.ts

```
import { json } from 'json-lint';
interface JsonValidationResult {
valid: boolean;
error: string | null;
}
```

```
const validateJson = (json: any): JsonValidationResult => {
try {
JSON.parse(JSON.stringify(json));
return { valid: true, error: null };
} catch (error) {
return { valid: false, error: error.message };
}
};
export const useJsonValidation = () => {
return { validateJson };
};
```

formService.ts

```
import { useForm } from 'react-hook-form';
interface FormSchema {
[key: string]: any;
}
const generateForm = (schema: FormSchema, register: any, errors: any) => {
return Object.keys(schema).map((key) => {
const field = schema[key];
return (
<div key={key}>
<label>{field.label}</label>
<input {...register(key)} type={field.type} />
{errors[key] && <div style={{ color: 'red' }}>{errors[key].message}</div>}
</div>
);
});
};
export const useSchema = () => {
```

return { generateForm };

};

Styles:

/* tailwind.css */

@tailwind base;

@tailwind components;

@tailwind utilities;

Tests:

tsx

// formGenerator.test.tsx

import React from 'react';

import { render, fireEvent, waitFor } from '@testing-library/react';

import FormGenerator from './FormGenerator';

describe('FormGenerator', () => {

it('renders form fields', () => {

const schema = {

name: { label: 'Name', type: 'text' },

email: { label: 'Email', type: 'email' },

};

const { getByLabelText } = render(<FormGenerator schema={schema} />);

expect(getByLabelText('Name')).toBeInTheDocument();

expect(get