Core concepts of React Hook Form:

1. **Import `useForm`:** This hook provides the core functionality for managing form state, validation, and submission.
2. **Destructure `register`, `handleSubmit`, and `errors`:**
   - `register`: Function to connect form inputs to React Hook Form, enabling validation and value tracking.
   - `handleSubmit`: Function to wrap your form submission handler. It automatically triggers validation and passes the form data if it's valid.
   - `errors`: An object that holds validation error messages for each field if validation fails.
3. **`register` input fields:** Use the spread syntax (`{...register("name", { required: true })}`) to register each input. You can pass validation rules as the second argument (e.g., `required: true`).
4. **Display validation errors:** Conditionally render error messages based on the `errors` object.
5. **Handle submission:** The `onSubmit` function is called when the form is submitted and the data is valid.

First, make a RegistrationForm component to be pulled into App.

Next, in the project, `npm i react-hook-form`

```
import React from 'react';
import { useForm } from 'react-hook-form';

function RegistrationForm() {
  const { register, handleSubmit, formState: { errors } } =
useForm();
  const onSubmit = (data) => console.log(data); // Handle
form submission

  return (
    <form onSubmit={handleSubmit(onSubmit)}>
      <div>
        <label htmlFor="name">Name:</label>
        <input type="text" id="name" {...register("name", {
required: true })} />
        {errors.name && <p>This field is required</p>}
      </div>
```

```
      <div>
        <label htmlFor="email">Email:</label>
        <input type="email"
id="email" {...register("email", {
          required: true,
          pattern: /^\S+@\S+$/i
        })} />
        {errors.email && <p>Invalid email address</p>}
      </div>

      <button type="submit">Submit</button>
    </form>
  );
}

export default RegistrationForm;
```

Enhancements:

```
npm install @mui/material @emotion/react @emotion/styled
```

Import TextField and Button, for example, and replace form elements with those:

```
import React from 'react';

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

import { TextField, Button } from '@mui/material';



function RegistrationForm() {

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

  const onSubmit = (data) => console.log(data); // Handle form
submission



  return (

    <form onSubmit={handleSubmit(onSubmit)}>
```

```
      <div>
        <label htmlFor="name">Name:</label>
        <TextField type="text" id="name" {...register("name",
{ required: true })} />
        {errors.name && <p>This field is required</p>}
      </div>

      <div>
        <label htmlFor="email">Email:</label>
        <TextField type="email" id="email" {...register("email",
{
          required: true,
          pattern: /^\S+@\S+$/i
        })} />
        {errors.email && <p>Invalid email address</p>}
      </div>

      <Button type="submit">Submit</Button>
    </form>
  );
}

export default RegistrationForm;
```

With React Form Hooks you can also:

Create complex validation rules: Use built-in validators, create custom ones, or integrate with third-party libraries like Yup.

Resolver: Integrate with popular schema validation libraries like Zod or Yup.

Yup provides a form validation schema. Install yup and resolvers; React Hook Form should already be installed.

```
npm install yup @hookform/resolvers
```

Yup validation schema (solution at `hookformyup`):

1. **Import Yup and Resolver:**
   - Import the necessary modules for Yup validation and integration with React Hook Form.
2. **Define Yup Schema:**
   - Create a `schema` object using Yup's schema builder. This defines the validation rules for each field.
   - We add `.required` with a custom error message for both `name` and `email`. Additionally, we use `.email` to validate the email format.
3. **Resolver Option:**
   - Pass the `yupResolver` function to the `resolver` option in `useForm`. This tells React Hook Form to use Yup for validation.
4. **Remove Inline Validation:**
   - Since Yup handles validation, you can remove the inline validation rules (`required`, `pattern`) from `register`.
5. **Display Yup Errors:**
   - Access the validation error messages through `errors.field.message` (e.g., `errors.name.message`).

Why Use Yup with React Hook Form?

- **Declarative Validation:** Yup schemas provide a clean and declarative way to define validation rules, making your form logic more readable and maintainable.
- **Complex Validation:** Yup easily handles complex validation scenarios like conditional validation, custom validation functions, and object/array validation.
- **Integration:** The `@hookform/resolvers/yup` package seamlessly integrates Yup with React Hook Form, streamlining the validation process.