



## Lesson 2- Front-End Development using Reactstrap

### Topics

- a. React UI Libraries
- b. ReactStrap Overview
- c. Installation
- d. Components in Reactstrap
- e. Form Validation
  - a. Yup
  - b. React Form

### Activity 2 – Front End Development

The objectives of this activity are to:

- a) Design the front-end of a React application using Reactstrap for responsive and modern UI components.
- b) Implement interactive elements and forms to enhance user experience.
- c) Use Reactstrap components like Button, Navbar, Form, Card, Collapse, and Input to build a structured, functional layout.
- d) Apply form validation using Yup and react-hook-form for robust input handling and validation.

**Note:** Follow the instructions carefully, as every task contributes to the completion of the final project. Make sure to complete each section thoroughly before moving on to the next, as the entire process is interconnected to achieve the final outcome.

#### Instructions:

#### Front-End Development Using Reactstrap

1. In Visual Code, open the folder for the postitapp created in the previous activity. Open terminal and change directory to the client folder. Execute command to start the app.

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS F:\React\FullStack\postitapp> cd client
PS F:\React\FullStack\postitapp\client> npm start

```

This is the expected output in the browser.



2. Install Reactstrap. Open another terminal. Execute command to install Reactstrap. Make sure to change to client directory.

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS F:\React\FullStack\postitapp> cd client
PS F:\React\FullStack\postitapp\client> npm install reactstrap
[#####.....] / reify:@types/react: sill audit bulk request {

```

3. Install Bootstrap and import in your **src/App.js**.

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Run `npm audit` for details.
PS F:\React\FullStack\postitapp\client> npm install bootstrap

up to date, audited 1559 packages in 2s

import "bootstrap/dist/css/bootstrap.min.css";

```

4. Let us use some Reactstrap components. The first component is the **Layout** Component. The Layout Component is a powerful mobile-first flexbox grid to build layouts of all shapes and sizes thanks to a twelve column system, six default responsive tiers.

ROW 1 - HEADER (Consistent component – visible always)
<p><b>ROW 2</b></p> <p>Row for rendering components depending on the menu clicked</p>
ROW 3 - FOOTER (Consistent component – visible always)

### App.js Wireframe

In App.js, write the following code to implement the wireframe using the Reactstrap Layout Components. You need to import the components that you will use such as the Container, Row and Col from reactstrap.

```
import "bootstrap/dist/css/bootstrap.min.css";
import "./App.css";
import Login from "./Components/Login";
import { BrowserRouter as Router, Routes, Route } from "react-router-dom";
import { Container, Row, Col } from "reactstrap"; //import the Reactstrap Components

const App = () => {
  return (
    <Container fluid>
      <Row>

        </Row>
      <Row>

        </Row>
      <Row>

        </Row>
      </Container>
    );
  }
};
```

```
};  
  
export default App;
```

5. Add the Col component required for each row and render the appropriate component based on the wireframe. In the second row add the `className="main"` and render the Home component in this row.

```
const App = () => {  
  return (  
    <Container fluid>  
      <Row>  
        <Header />  
      </Row>  
      <Row className="main">  
        <Home />  
      </Row>  
      <Row>  
        <Footer />  
      </Row>  
    </Container>  
  );  
};
```

This should be your initial output.





6. Next is to create the Layout for the Home component. Refer to the wireframe below.

User Component	SharePost Component
	Post Component

### Home.js Wireframe

Edit the **Home.js** file and create the Layout. Import the needed Reactstrap components.

```
import { Container, Row, Col } from "reactstrap"; //import the Reactstrap Components
```

Create the layout by using the Row and Col Components. Render the appropriate components in the respective columns based on the wireframe.

```
const Home = () => {
  return (
    <>
      <Row>
        <Col md={3}>
          <User />
        </Col>
        <Col md={9}>
          <SharePosts />
        </Col>
      </Row>
      <Row>
        <Col md={3}></Col>
        <Col md={9}>
          <Posts />
        </Col>
      </Row>
    </>
  );
};
```

This should be the output in the browser.

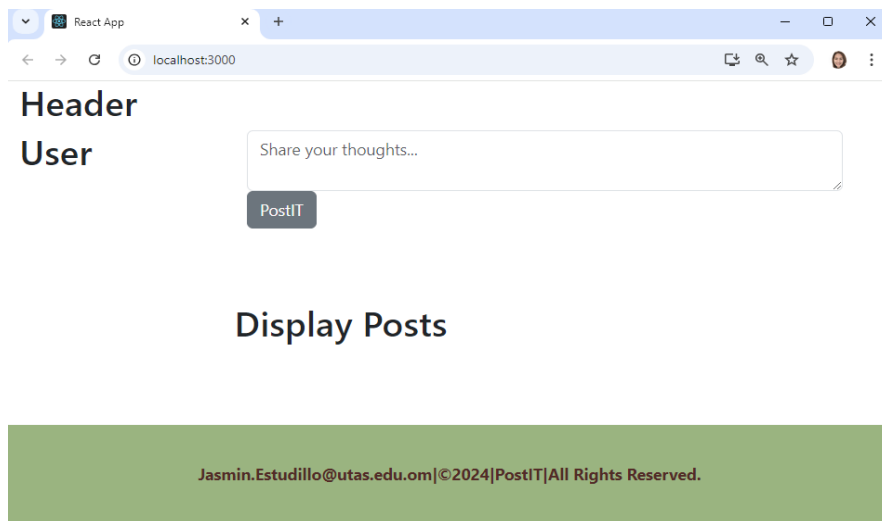


- Update the **SharePost.js** file and use the Form Components of Reactstrap. Import the form components of Reactstrap.

```
import {
  Button,
  Col,
  Label,
  Container,
  Row,
  FormGroup,
  Input,
} from "reactstrap";
```

- Create a row and a column. In the column add the Input and Button component.

```
const SharePosts = () => {
  return (
    <Container>
      <Row>
        <Col>
          <Input
            id="share"
            name="share"
            placeholder="Share your thoughts..."
            type="text"
          />
          <Button>PostIT</Button>
        </Col>
      </Row>
    </Container>
  );
};
```

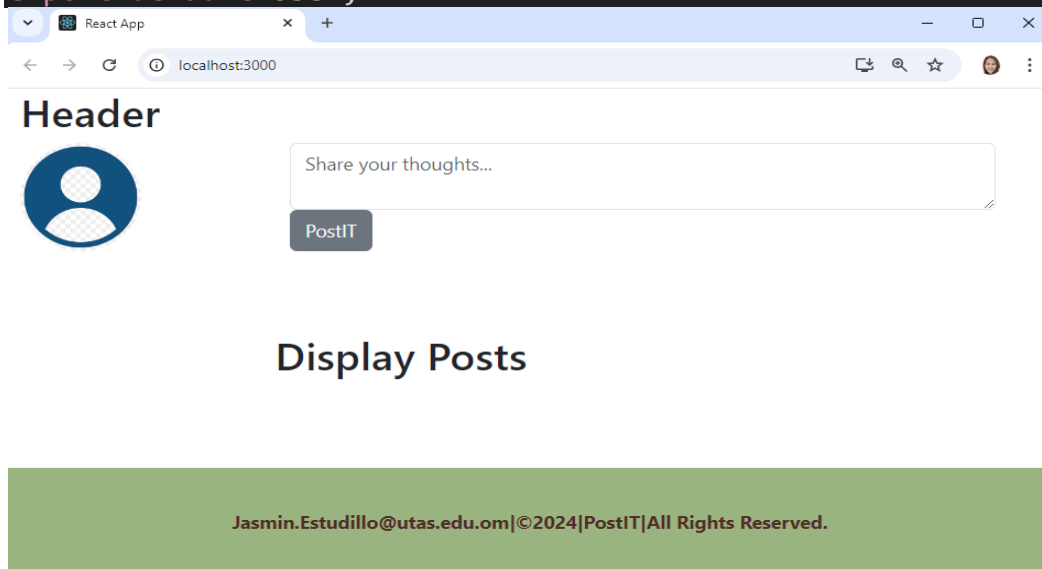


9. Update User.js and import the default image for the user. Display this image.

```
import user from "../Images/user.png";

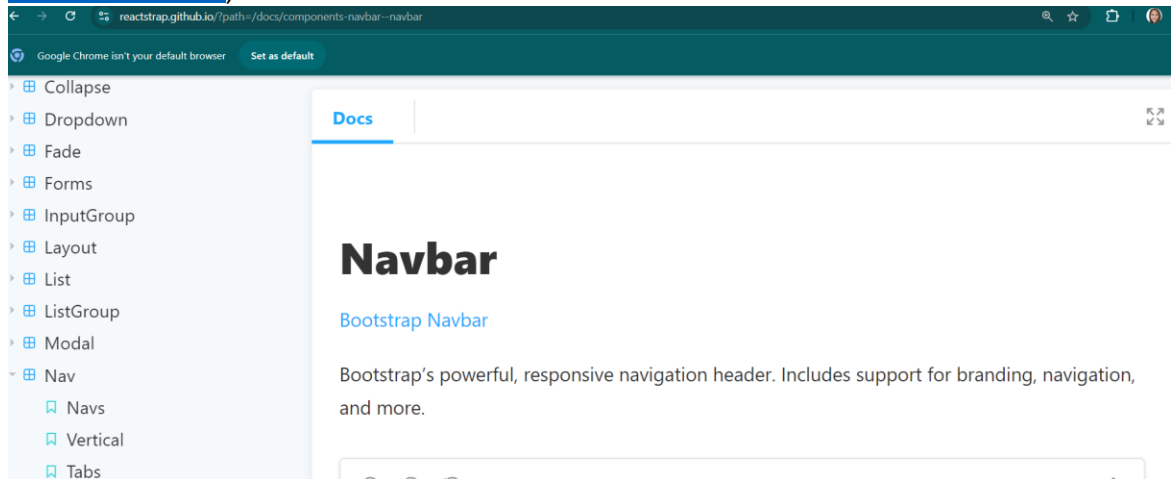
const User = () => {
  return (
    <div>
      <h1>User</h1>
      <img src={user} className="userImage" />
    </div>
  );
};

export default User;
```





10. Create the navigation by using the Reactstrap navigation components. You may refer to the documentation (<https://reactstrap.github.io/?path=/docs/components-navbar--navbar>).



11. Update Header.js. Import the Reactstrap components required for the navigation.

```
import {
  Navbar,
  Nav,
  NavItem,
  NavLink,
} from "reactstrap";
```

12. Write the code to create the navigation.

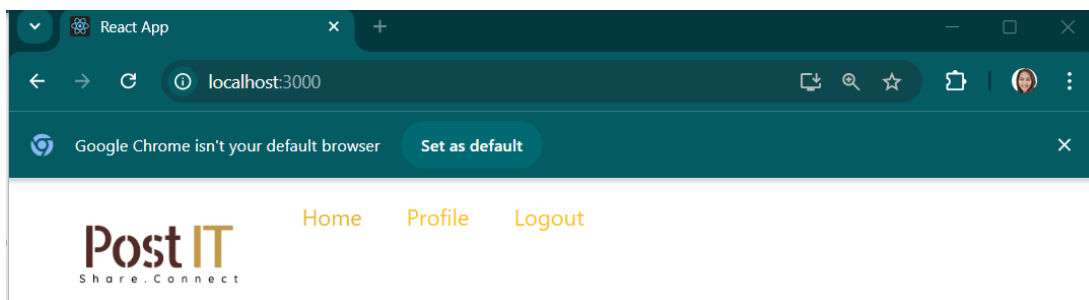
```
const Header = () => {

  return (
    <>
      <Navbar>
        <Nav>
          <NavItem>
            <NavLink active href="#">
              Home
            </NavLink>
          </NavItem>
          <NavItem>
            <NavLink href="#">Profile</NavLink>
          </NavItem>
          <NavItem>
            <NavLink href="#">Logout</NavLink>
          </NavItem>
        </Nav>
      </Navbar>
    </>
  );
};
```

13. In Header.js, import the image logo.png. Then, write the code to add new <Navitem> and display the image in that navitem.

```
import logo from "../Images/logo-t.png";
```

The navigation should display as the one below.



Write the code.

---



---



---



---

14. Update the App.css with the following CSS rules to design the header properly. Analyze and understand the CSS.

```
.header {
  display: flex;
  align-items: center; /* Vertically align items */
  padding: 10px;
  background-color: #f8f9fa;
}

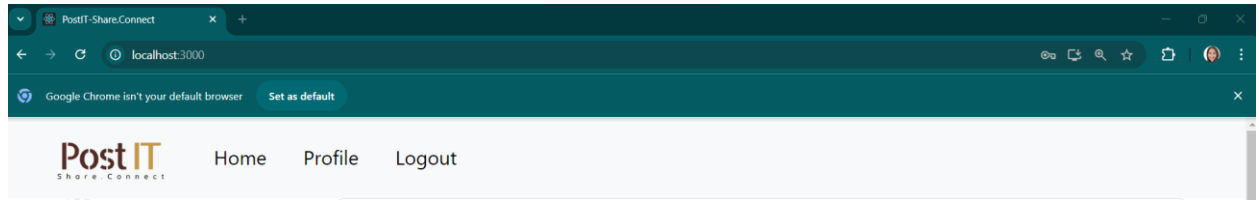
.nav {
  display: flex;
  align-items: center; /* Vertically align items */
}

.logo {
  height: 50px; /* Adjust the logo size as needed */
}

.nav a {
  text-decoration: none; /* Remove underline */
  color: #000;           /* Set text color */
  margin: 0 15px;        /* Add spacing between links */
}

.nav a:hover {
  color: #007bff; /* Change color on hover (optional) */
}
```

15. Apply the .header class to the <Navbar> element. The header will now appear as shown.



16. In **App.js**, create the client Routes using react-router-dom. Import components needed from react-router-dom.

```
import { BrowserRouter as Router, Routes, Route } from "react-router-dom";
```

17. Update the src/App.js and create the client routes using the react-router-dom components. The Routes will be responsible for rendering the appropriate component depending on the link activated. First add the <Router> component to encapsulate the routes. You can write it after the <Container> component.

```
const App = () => {
  return (
    <Container fluid>
      <Router>
        ...
      </Router>
    </Container>
  );
};
```

18. In the second row, write the codes to create the Routes. Each route specifies a path and the component (<Login />, <Home />, <Profile />, <Register />) that should be rendered when a user navigates to that path.

```
const App = () => {
  return (
    <Container fluid>
      <Router>
        <Row>
          <Header />
        </Row>
        <Row className="main">
          <Routes>
            <Route path="/" element={<Home />}></Route>
            <Route path="/login" element={<Login />}></Route>
            <Route path="/profile" element={<Profile />}></Route>
            <Route path="/register" element={<Register
  />}></Route>
          </Routes>

          </Row>
          <Row>
            <Footer />
          </Row>
        </Router>
      </Container>
    );
  };
};
```

19. Update the Header.js file to create links to correspond to the created routes. Import the <Link> component.

```
import { Link } from "react-router-dom";
```

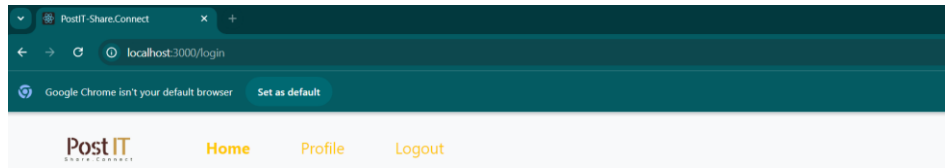
Then, use the <Link> component to create the link to the routes. Below is an example. Remove the <Navlink> and change it to <Link> of the react router. Create all the links to, home, profile and logout routes.

```
<NavItem>
  <Link to="/">Home</Link>
</NavItem>
```

Include the following CSS rules in App.css to remove the underline from the link:

```
.nav-link a{
  color: inherit;
  text-decoration: none;
}
```

20. Navigate to your <http://localhost:3000/login> route. You should see the default login page.

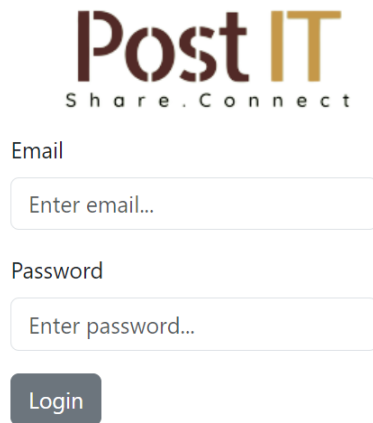


21. Update Login.js and create the layout for the login page. It consists of 3 rows with 1 column each.

```
const Login = () => {

  return (
    <div>
      <Container>
        <Form>
          <Row>
            <Col md={3}>
            </Col>
          </Row>
          <Row>
            <Col md={3}>
            </Col>
          </Row>
          <Row>
            <Col md={3}>
            </Col>
          </Row>
          <Row>
            <Col md={3}>
            </Col>
          </Row>
        </Form>
      </Container>
    </div>
  );
};
```

22. Refer to the Reactstrap documentation <https://reactstrap.github.io/?path=/docs/components-forms--form> to create the login page as shown.



The screenshot shows a login form for 'Post IT' with the tagline 'Share. Connect'. It includes an 'Email' field with a placeholder 'Enter email...', a 'Password' field with a placeholder 'Enter password...', and a 'Login' button.

Write also the code to display the logo in the first row.

23. Update Login.js, create a link to the register route. Make sure to import the Link component of the react-router-dom.

```
import { Link } from "react-router-dom";
```

.....

```

    <p className="smalltext">
      No Account? <Link to="/register">Sign Up
now.</Link>
    </p>

```

**Note:** Code to hide the Header when Login and Register component will be rendered will be done in the next topics.

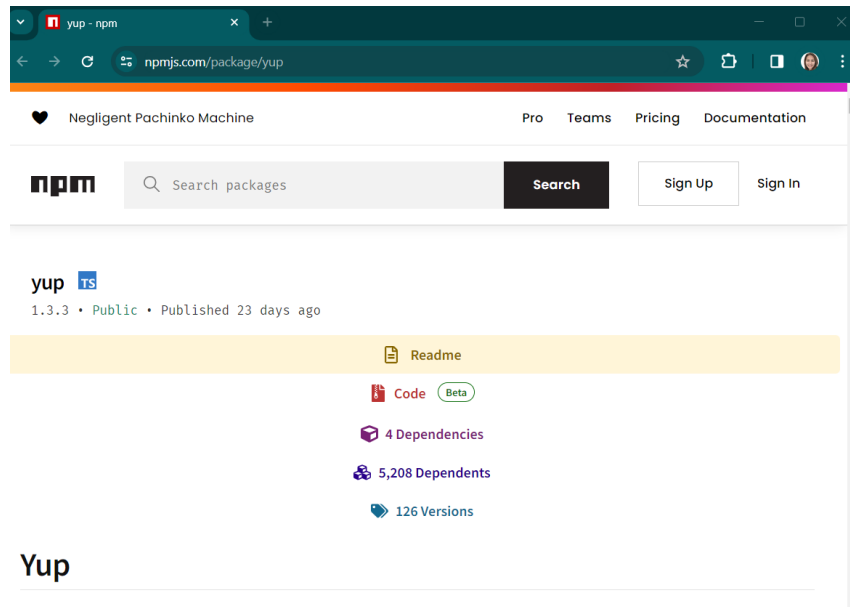
## Implement form validation using Yup and react-hook-form

24. Download the **Register.js** from E-learning or (*you can design your own Register component*) and save it in the Components folder.
25. Navigate to your <http://localhost:3000/register> route. The Register component should be rendered. In this register form, client side validation is required in order to ensure that data going to the database are filtered and in correct form as expected.

The screenshot shows a web browser window with the address bar displaying 'localhost:3000/register?'. The page has a dark teal header with the 'PostIT' logo and navigation links for 'Home', 'Profile', and 'Logout'. The main content area has a dark red background and features a white registration form. The form contains four input fields: 'Enter your name...', 'Enter your email...', 'Enter your password...', and 'Confirm your password...'. Below these fields is a blue 'Register' button. At the bottom of the page, a green footer bar contains the text 'Jasmin.Estudillo@utas.edu.om|©2024|PostIT|All Rights Reserved.'

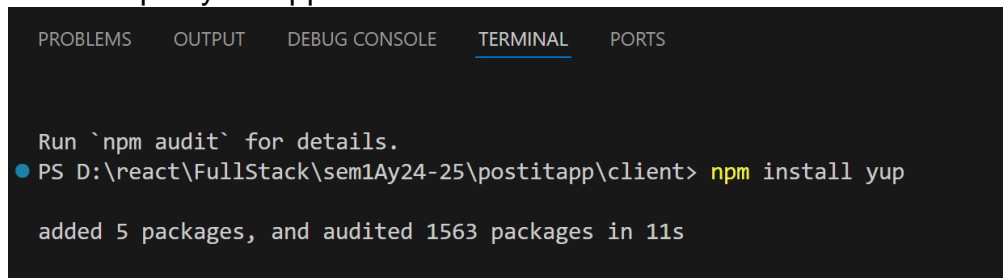
26. Using **Yup** for validation. According to the website of Yup:  
 “Yup is a schema builder for runtime value parsing and validation. Define a schema, transform a value to match, assert the shape of an existing value, or both. Yup schema are extremely expressive and allow modeling complex, interdependent validations, or value transformation.”





<https://www.npmjs.com/package/yup>

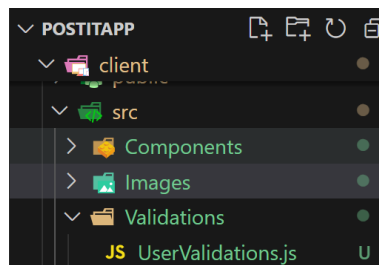
27. Install Yup in your application.



28. To use Yup, you need to create a Schema which is defined as:

*“Schema are comprised of parsing actions (transforms) as well as assertions (tests) about the input value. Validate an input value to parse it and run the configured set of assertions. Chain together methods to build a schema.”*

To organize validation schemas, **create a folder Validations in client/src.**  
Inside this folder, create a new file: **UserValidations.js.**



29. In this file, import all exports from the yup package. Create a variable for the validation schema by using the shape method of the yup object.

```
import * as yup from "yup"; //import all exports from the yup
export const userSchemaValidation = yup.object().shape({});
```

30. Then provide an object, which defines the details of the validation rules. The validation rules for each field in the schema are specified below. You can refer to the documentation of Yup for more information.

**name:**

Type: string

Validation: Required (cannot be empty)

**email:**

Type: string

Validation: Should be a valid email format, and it is required (cannot be empty)

**password:**

Type: string

Validation:

Minimum length: 4 characters

Maximum length: 20 characters

Required (cannot be empty)

**confirmPassword:**

Type: string

Validation:

Must match the value of the "password" field (using yup.ref("password"))

Required (cannot be empty)

To implement these rules:

```
import * as yup from "yup"; //import all from the yup

export const userSchemaValidation = yup.object().shape({
  name: yup.string().required("Name is required"),
  email: yup
    .string()
    .email("Not valid email format")
    .required("Email is required"),
  password: yup.string().min(4).max(20).required("Password is required"),
  confirmPassword: yup
    .string()
    .oneOf([yup.ref("password"), null], "Passwords Don't Match")
    .required(),
});
```

Note you need to export your schema so it can be accessed outside of the file.

31. In `src/Components/Register.js`, import the `UserValidation.js`.

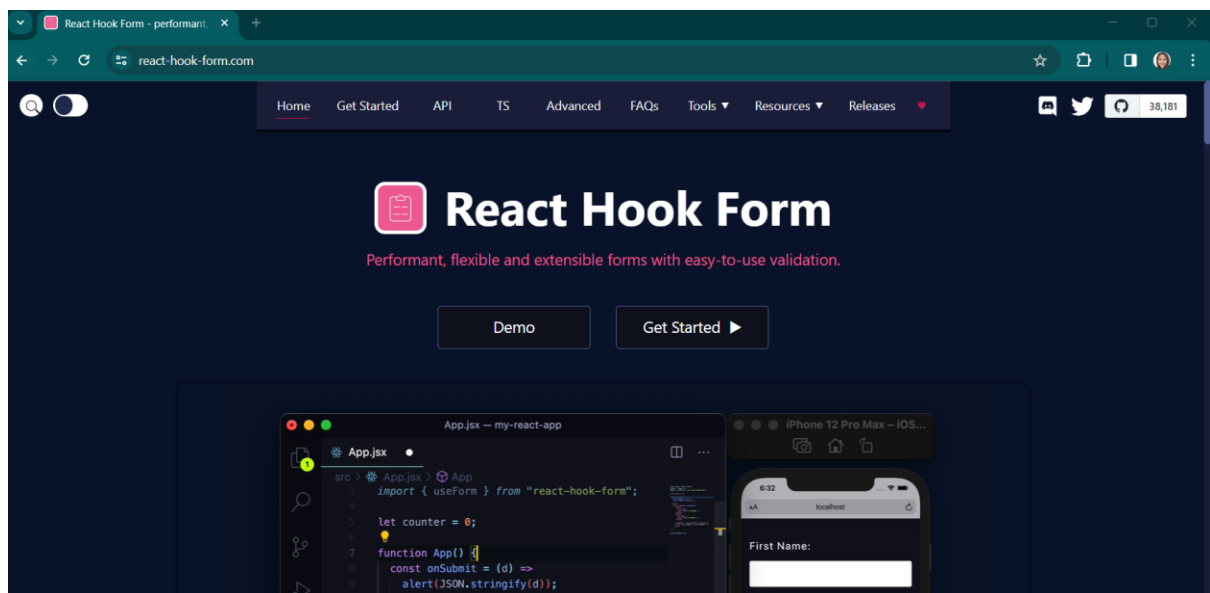
```
import { userSchemaValidation } from
"../Validations/UserValidations";
```

32. Next set up the react-hook-form. Install it in your application.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS D:\react\FullStack\sem1Ay24-25\postitapp\client> npm install react-hook-form
[.....] / idealTree: sill logfile start cleaning logs, removing 3 files
```

This is the website of react-hook-form.



<https://react-hook-form.com/>

33. Install the yupResolver. The yupResolver is a function from the react-hook-form library that helps integrate react-hook-form validation with Yup validation schemas.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS D:\react\FullStack\sem1Ay24-25\postitapp\client> npm install @hookform/resolvers
[.....] / idealTree: sill logfile start cleaning logs, removing 3 files
```

34. In **Register.js**, import the following:

```
import * as yup from "yup";
import { useForm } from "react-hook-form";
import { yupResolver } from "@hookform/resolvers/yup";
```

Use destructuring to create variables for specific properties of the object returned by the useForm hook. The useForm Hook helps manage form state, handling input registration, validation, and submission.

```
const Register = () => {

  //For form validation using react-hook-form
  const {
    register,
    handleSubmit, // Submit the form when this is called
    formState: { errors },
  } = useForm({
    resolver: yupResolver(userSchemaValidation), //Associate your
    //Yup validation schema using the resolver
  });

  // Handle form submission
  const onSubmit = (data) => {
    console.log("Form Data", data); // You can handle the form
    //submission here
  }
  return (
    ...
  )
}
```

### yupResolver:

This function is used as a resolver for the useForm hook to integrate Yup validation with the form. It takes the Yup schema (userSchemaValidation) and resolves it with the form state.

### Destructuring:

**register:** This function is used to register form inputs. It is typically used as a ref for each input element.

**handleSubmit:** This function is provided by useForm and is called when the form is submitted. It triggers the form validation and then, if successful, invokes the submission function.

**formState: { errors }:** This part deconstructs the errors property from the form state. It contains validation errors for each form field.

35. In the Form element, using the onSubmit event handler, invoke the function **submitForm** that was specified in the react-hook-form to perform the validations.

```
return (
  <Container fluid>
    ...
    <form className="div-form"
      onSubmit={handleSubmit(onSubmit)}>
    ...
  )
```

- **handleSubmit:** A method from react-hook-form that triggers validation and prevents the default form submission if there are validation errors.
- **onSubmit:** This function gets executed if validation succeeds. Form data submission or any further actions can be processed or handled in this function.

36. Then, register each of the input using the register function. The register function from react-hook-form is spread onto the input as JSX attributes using the spread operator ({...}). This associates the input with the form state and allows react-hook-form to track its value and handle validation.

Do the same for all the inputs.

```
<input
  type="text"
  className="form-control"
  id="name"
  placeholder="Enter your name..."
  {...register("name")}
/>
```

Display the errors:

```
<p className="error">{errors.name?.message}</p>
```

Test your validation:

The screenshot shows a web browser window with the URL `localhost:3000/register?`. The page has a dark red background. At the top, there is a navigation bar with the "Post IT" logo and links for "Home", "Profile", and "Logout". The main content area contains a white registration form. The form has four input fields: "Enter your name...", "Enter your email...", a password field with four dots, and another password field with three dots. Below each input field is a red error message: "Name is required", "Email is required", and "Passwords Don't Match". At the bottom of the form is a blue "Register" button.

37. In the `onSubmit` function that handles form submission, add a simple alert function for testing purposes.

```
// Handle form submission
const onSubmit = (data) => {
  console.log("Form Data", data); // You can handle the form
  submission here
  alert("Validation all good.")
}
```

38. You can also check the browser's console to see the data submitted.

