Hands-on Lab: Convert Static Pages to Dynamic React Components

Estimated time needed: **45** minutes

Introduction

In this lab, you will convert static website pages to dynamic React components.

Objectives

After completing this lab, you will be able to:

- Convert the following static pages into dynamic React components:
    - Landing Page
    - Navigation Bar
    - Sign Up form
    - Login form
- Test the conversion

Prerequisites

- You should have completed the prerequisite courses, specially the **Developing Front-End Apps with React** course.
- You must have completed the following labs:
    - Design Website Layouts
    - Create a GitHub Repository for your Project
    - Build Static Website Layouts
    - Set up the React Environment

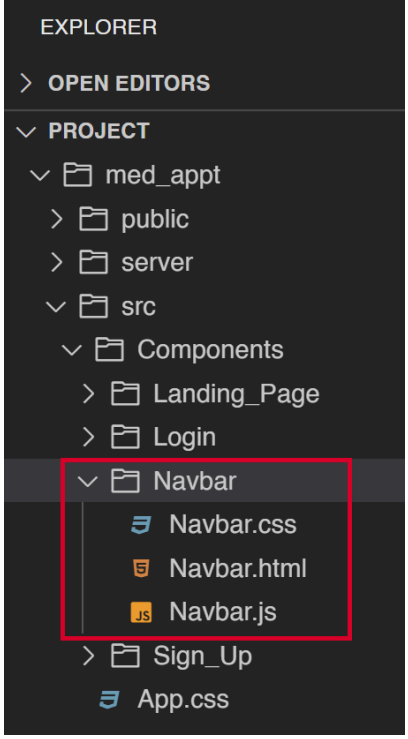
Project Scenario

Click **here** to review the project scenario.

Exercise 1: Create a React component

1. In the Skills Network lab environment, open a new terminal.
2. Clone your React project's GitHub repository.
3. Perform `npm install` in the React Project's **root** folder and in the **server** folder.
4. Navigate to the React project's **src** folder via the Explorer.
5. Create a folder named **Components** in the to start building the website components.
6. Move the following folders into the **Components** folder:
    - Landing_Page
    - Navbar
    - Login
    - Sign_up
7. Next, create a file named **Navbar.js** in the **Navbar** folder.

   A sample folder structure is displayed below:



8. **Take a screenshot** of the Navigation Bar structure and save it as **react_navbar_folder_struct.png**.

   Note: Replace **navbar** with the component name when saving screenshots of the folder structure for the other components.

9. Install the **ES7+ React/Redux/React-Native snippets** via the **Extensions** option on the Skills Network toolbar to access React snippets.

   Refer to the **Capstone Project Technical Reference** reading for details about ES7+ and other technical concepts.

10. Copy the entire HTML code inside `body` tag from **Navbar.html** to **Navbar.js** within the `div` tag of `return` for the function component.

11. Then, convert the code into JSX format, such as change `class` into `className`.

12. Import **Navbar.css** into the **Navbar.js** component using `import ./Navbar.css;`

13. Within the `return of` function component in the **App.js** file, remove all code starting from first `<div>` to the last `</div>`. Then, place `<> </>` inside return as JSX syntax.

14. You must:

    1. Install **react router dom** in the react project's root folder using `npm i react-router-dom` for creating routes.
    2. Then, include Navbar.js after **BrowserRouter** to render the Navigation Bar on every page of the website.

       Refer to the **Capstone Project Technical Reference** reading for details about React Router and other technical concepts.

    ▶ Click here for a **sample solution with routers**.

15. Test the **Navbar** component along with its CSS file:

    1. Perform `npm start` in the React project's root folder
    2. Launch the client-side application via **Launch Application**.

16. **Take a screenshot** of the output of the **Navbar** component and save it as **react_navbar_output.png**.

    Note: Replace **navbar** with the component name when saving screenshots of the test output.

17. Create a folder named as **Landing_Page**. Include **Landing_Page.js** and **Landing_page.css** within this folder. Convert the html file into React Function Component using JSX syntax. Import Landing_Page.css into Landing_Page.js to applt style.

▶ Click here for **exemplar solution code.**

18. You can make this Landing page as the Home route after you import the Landing_Page path.

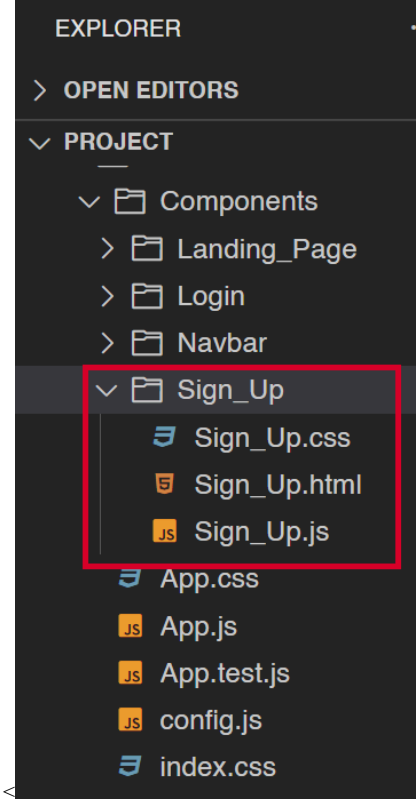
- Use this statement to import:
  `import Landing_page from './components/Landing_Page/Landing_page'};`
- Refer to the statement below to create route for this path.
  `<Route path="/" element={<Landing_Page/>}/>`

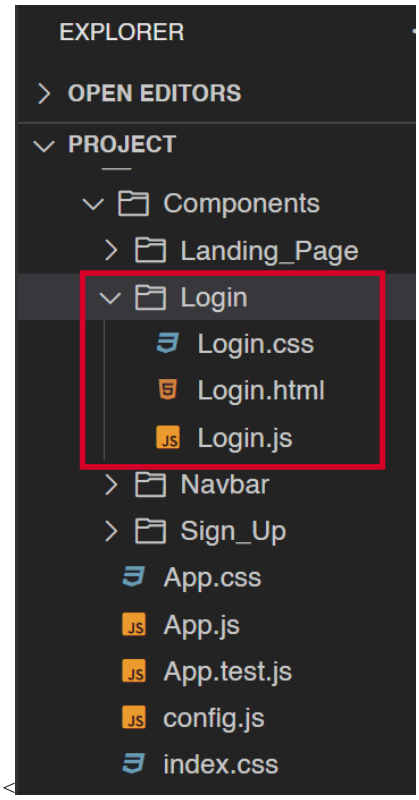
19. Perform `git config --global` to enter name and email, `git add`, `git commit`, and `git push` commands to update changes into your React project' GitHub repository for proper code management.

    Refer to the **Capstone Project Reference: Git Commands** reading for details about Git command syntax and use relevant to the project.

Exercise 2: Sign Up and Login components

1. Repeat steps 1 to 9 from **Exercise 1: Create a new React component** for the **Sign_Up** and **Login** forms within **Sign_Up** and **Login** folders to create the React components and the associated CSS files in the respective folders. The folder structure for **Sign_Up** and **Login** forms should look like below:





2. You must add validation for signup and login form to validate if user is filling the right details.

   Note: You can refer to the exemplar solution code for the Sign Up and Login forms to review the validation code as it has already been incorporated.

3. **Take screenshots** and save them as **signup_validation.png** and **login_validation.png**.

▶ Click here for a **hint**.

3. Connect the links to different components in the **Navbar** component in accordance with React guidelines and formats.
4. You must **take screenshots** of the folder structure for **Sign_Up** and **Login** and save them as **react_signup_folder_struct.png** and **react_login_folder_struct.png**, respectively.
5. You must also **take screenshots** for outputs of both sign up and login form and save them as **react_signup_output.png** and **react_login_output.png** respectively.
6. Test each component with the associated CSS file. Make sure that the components are working as expected by launching client-side application.
7. Perform `git add`, `git commit`, and `git push` commands to update changes into your React project' GitHub repository for proper code management.

   Refer to the **Capstone Project Reference: Git Commands** reading for details about Git command syntax and use relevant to the project.
8. You must create a folder structure for the website's Landing page similar to the **navbar** component.
9. Then, you need to integrate landing page of website with Navbar in such a way that Navbar is visible at the top of landing page.
10. Perform `git add`, `git commit`, and `git push` commands to update changes into your React project' GitHub repository for proper code management.

    Refer to the **Capstone Project Reference: Git Commands** reading for details about Git command syntax and use relevant to the project.

Exercise 3: Embed code in the Sign Up page to establish backend connectivity

1. Copy the code below and paste it in the Sign_Up.js file replacing all the existing content.

```
// Following code has been commented with appropriate comments for your reference.
import React, { useState } from 'react';
import './Sign_Up.css'
import { Link, useNavigate } from 'react-router-dom';
import { API_URL } from '../../config';
// Function component for Sign Up form
const Sign_Up = () => {
    // State variables using useState hook
    const [name, setName] = useState('');
    const [email, setEmail] = useState('');
    const [phone, setPhone] = useState('');
    const [password, setPassword] = useState('');
    const [showerr, setShowerr] = useState(''); // State to show error messages
    const navigate = useNavigate(); // Navigation hook from react-router
    // Function to handle form submission
    const register = async (e) => {
        e.preventDefault(); // Prevent default form submission
        // API Call to register user
        const response = await fetch(`${API_URL}/api/auth/register`, {
            method: "POST",
            headers: {
                "Content-Type": "application/json",
            },
            body: JSON.stringify({
                name: name,
                email: email,
                password: password,
                phone: phone,
            }),
        });
        const json = await response.json(); // Parse the response JSON
        if (json.authtoken) {
            // Store user data in session storage
            sessionStorage.setItem("auth-token", json.authtoken);
            sessionStorage.setItem("name", name);
            sessionStorage.setItem("phone", phone);
            sessionStorage.setItem("email", email);
            // Redirect user to home page
            navigate("/");
            window.location.reload(); // Refresh the page
        } else {
            if (json.errors) {
                for (const error of json.errors) {
                    setShowerr(error.msg); // Show error messages
                }
            } else {
                setShowerr(json.error);
            }
        }
    };
    // JSX to render the Sign Up form
    return (
        <div className="container" style={{marginTop:'5%'}}>
            <div className="signup-grid">
                <div className="signup-form">
                    <form method="POST" onSubmit={register}>
                        <div className="form-group">
                            <label htmlFor="email">Email</label>
                            <input value={email} onChange={(e) => setEmail(e.target.value)} type="email" name="email" id="email" className="form-control" placeh
                            {showerr && <div className="err" style={{ color: 'red' }}>{showerr}</div>}
                        </div>
                        {/* Apply similar logic for other form elements like name, phone, and password to capture user information */}
                    </form>
                </div>
            </div>
        </div>
        {/* Note: Sign up role is not stored in the database. Additional logic can be implemented for this based on your React code. */}
    );
}
export default Sign_Up; // Export the Sign_Up component for use in other components
```

What does the above code do and what you need to add in it?

1. The code contains various state variables that have been initialized using the **useState** hook for managing form data and user interface states.
2. At line number 13, the **navigate** variable has been declared using the **useNavigate** hook. You can use this hook for navigation.
3. At line 15, a function named **register** has been created that handles the form submission and API call from the server side. This will establish the database connection.
4. At line number 42, the **navigate** variable has been used to navigate to the Home page after the sign up is successful. You must implement logic to change the Navbar to display the **Logout** button and its functionality in your Navbar component.

Take a screenshot and save it as **logout_button.png**.

▶ Click here for a **sample solution** with a hint.

5. After this you need to extract the name of the user through email id before @ symbol and display at left side of the logout button. You can use above code as reference.

▶ Click here for a **sample screenshot**.

6. The code from line 59-63 provides a sample for creating an **email** form element using the useState hook at line number 8. You must create form elements for other Sign- Up fields using similar logic at line number 64.

   Note: Line number 63 which is part of database connectivity to check if email id already exist in database. You need not to include it with other elements.

▶ Click here for a **sample solution**.

7. With above steps user can easily register within this react application.

   Note: The code on line 61 verifies whether or not the email ID already exists in the database; if it does, an error is triggered. This code will be used only for the Sign Up component. To make sure database connection is working the mongoDB should be active. Then navigate inside server folder and you need to execute `node index`.

8. You can apply style sheet based on your website's theme.

9. Perform `git add`, `git commit`, and `git push` commands to update changes into your React project's GitHub repository for proper code management.

   Refer to the **Capstone Project Reference: Git Commands** reading for details about Git command syntax and use relevant to the project.

Exercise 4: Embed code in the Login page

1. Copy the code below and paste it in the **Login.js** file replacing all the existing content.

```
// Following code has been commented with appropriate comments for your reference.
import React, { useState, useEffect } from 'react';
// Apply CSS according to your design theme or the CSS provided in week 2 lab 2
import { Link, useNavigate } from 'react-router-dom';
import { API_URL } from '../../config';
const Login = () => {
  // State variables for email and password
  const [password, setPassword] = useState("");
  const [email, setEmail] = useState('');
  // Get navigation function from react-router-dom
  const navigate = useNavigate();
  // Check if user is already authenticated, then redirect to home page
  useEffect(() => {
    if (sessionStorage.getItem("auth-token")) {
      navigate("/");
    }
  }, []);
  // Function to handle login form submission
  const login = async (e) => {
    e.preventDefault();
    // Send a POST request to the login API endpoint
    const res = await fetch(`${API_URL}/api/auth/login`, {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({
        email: email,
        password: password,
      }),
    });
    // Parse the response JSON
    const json = await res.json();
    if (json.authtoken) {
      // If authentication token is received, store it in session storage
      sessionStorage.setItem('auth-token', json.authtoken);
      sessionStorage.setItem('email', email);
      // Redirect to home page and reload the window
      navigate('/');
      window.location.reload();
    } else {
      // Handle errors if authentication fails
      if (json.errors) {
        for (const error of json.errors) {
          alert(error.msg);
        }
      } else {
        alert(json.error);
      }
    }
  };
  return (
    <div>
      <div className="container">
        <div className="login-grid">
          <div className="login-text">
            <h2>Login</h2>
          </div>
          <div className="login-text">
            Are you a new member?
            <span>
              <Link to="/signup" style={{ color: '#2190FF' }}>
                Sign Up Here
              </Link>
            </span>
          </div>
          <br />
          <div className="login-form">
            <form onSubmit={login}>
              <div className="form-group">
                <label htmlFor="email">Email</label>
                {/* Input field for email */}
                <input
                  value={email}
                  onChange={(e) => setEmail(e.target.value)}
                  type="email"
                  name="email"
                  id="email"
                  className="form-control"
                  placeholder="Enter your email"
                  aria-describedby="helpId"
                />
              </div>
              {/* Input field for password */}
              // write logic code for password input box
              <div className="btn-group">
                {/* Login button */}
                <button type="submit" className="btn btn-primary mb-2 mr-1 waves-effect waves-light">
                  Login
                </button>
              </div>
            </form>
          </div>
        </div>
      </div>
    </div>
  )
}
```

```
export default Login;
```

What does the above code do and what you need to add in it?

1. The code contains various state variables that have been initialized using the **useState** hook for managing form data and user interface states.
2. At line number 13, the **navigate** variable has been declared using the **useNavigate** hook. You can use this hook for navigation.
3. At line 20, a function named **login** has been created that handles the form submission and API call from the server side. This will establish the database connection.
4. At line number 40, the **navigate** variable has been used to navigate to the Home page after the sign up is successful. You must need to check if logic to display the **Logout** button and extracted name from email is functional in your **Navbar** component or not.

▶ Click here for a **hint**.

5. The code from line 66-69 provides a sample for creating an **email** form element using the useState hook at line number 11. You must create form elements for password input element as well at line number 70.

▶ Click here for a **sample solution**.

6. You can apply style sheet based on your website's theme.
7. Perform `git config --global` for name and email, `git add`, `git commit`, and `git push` commands to update changes into your React project' GitHub repository for proper code management.

   Refer to the **Capstone Project Reference: Git Commands** reading for details about Git command syntax and use relevant to the project.

Screenshot checklist

You should have taken the following screenshots as part of this lab:

- *react_navbar_folder_struct.png*
- *react_navbar_output.png*
- *signup_validation.png*
- *login_validation.png*
- *logout_button.png*
- *react_signup_folder_struct.png*
- *react_login_folder_struct.png*
- *react_signup_output.png*
- *react_login_output.png*

Note about data management and persistence

This platform is not persistent. Make sure to commit your files to the repository before you leave the lab. When you revisit, you will need to clone the repository again.

To ensure the proper management and persistence of your data in a GitHub repository, it is crucial to follow a few essential steps:

- **Regular Updates:** Whenever you make changes or add new components to your project, it is essential to add, commit, and push the updates to your GitHub repository. This ensures that your latest work is safely stored and accessible to collaborators.
- **Session Persistence:** During an active session, your data remains accessible. However, it's important to note that if your session expires or you log out, you will need to clone the repository again to resume work.
- **Ignoring node modules:** When pushing data to GitHub, it's best practice to exclude the node modules folder from both your server and client directories. This folder contains external dependencies and can be quite large, making the repository heavy and slowing down the process. By adding it to the .gitignore file, you prevent it from being pushed to the repository, keeping your commits cleaner and more focused.

By adhering to these guidelines, you can maintain a well-organized and efficient GitHub repository, ensuring that your work is securely stored and easily accessible to you and your collaborators.

Author(s)

Richa Arora