Hands-on Lab: Build the Profile Component

Skills Network

Estimated time needed: **90** minutes

Introduction

In this lab, you will manage profile information using the **Profile** card.

Objectives

After completing this lab, you will be able to:

- Create a component that displays profile information in a card
- Add the **Update** functionality for the Profile component.
- Display user-specific reports, which a user can view and download

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
    - Convert Static Pages to Dynamic React Components
    - Integrate Existing Functionality
    - Build the Appointment Booking Component, in particular the **Appointment Booking** component must be fully functional
    - Build the Notification Component
    - Build the Reviews Component

Exercise 1: Create the ProfileCard component

1. Create a **ProfileCard** folder within the **components** folder, which is in the **src** folder. Create the `ProfileCard.js` component and the `ProfileCard.css` file.

   src->components->ProfileCard->ProfileCard.js,ProfileCard.css

2. Implement the necessary elements within the component to display the user's profile information in a card format.

3. Create a drop-down in the Navbar to display a user's profile as an option.

4. **Take a screenshot** of the output and save it as **profilecard.png**.

▶ Click here to view the sample **drop-down for your Profile**.

5. Remember to perform `git add`, `git commit`, and `git push` all changes in your forked GitHub repository.

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

Exercise 2: Display the profile details

1. In case you exited and are re-entering the Skills Network lab environment:
   1. Open new terminal.
   2. Clone your React project's GitHub repository.

2. You can use the code block below to fetch information in `ProfileCard` component from database.

```
// Following code has been commented with appropriate comments for your reference.
// Import necessary modules from React and other files
import React, { useEffect, useState } from "react";
import { API_URL } from "../../config";
import { useNavigate } from "react-router-dom";
// Define a Function component called ProfileForm
const ProfileForm = () => {
  // Set up state variables using the useState hook
  const [userDetails, setUserDetails] = useState({});
  const [updatedDetails, setUpdatedDetails] = useState({});
  const [editMode, setEditMode] = useState(false);

  // Access the navigation functionality from React Router
  const navigate = useNavigate();

  // Use the useEffect hook to fetch user profile data when the component mounts or updates
  useEffect(() => {
    const authtoken = sessionStorage.getItem("auth-token");
    if (!authtoken) {
      navigate("/login");
    } else {
      fetchUserProfile();
    }
  }, [navigate]);
  // Function to fetch user profile data from the API
  const fetchUserProfile = async () => {
    try {
      const authtoken = sessionStorage.getItem("auth-token");
      const email = sessionStorage.getItem("email"); // Get the email from session storage
      if (!authtoken) {
        navigate("/login");
      } else {
        const response = await fetch(`${API_URL}/api/auth/user`, {
          headers: {
            "Authorization": `Bearer ${authtoken}`,
            "Email": email, // Add the email to the headers
          },
        });
        if (response.ok) {
          const user = await response.json();
          setUserDetails(user);
          setUpdatedDetails(user);
        } else {
          // Handle error case
          throw new Error("Failed to fetch user profile");
        }
      }
    } catch (error) {
      console.error(error);
      // Handle error case
    }
  };
  // Function to enable edit mode for profile details
  const handleEdit = () => {
    setEditMode(true);
  };
  // Function to update state when user inputs new data
  const handleInputChange = (e) => {
    setUpdatedDetails({
      ...updatedDetails,
      [e.target.name]: e.target.value,
    });
  };
  // Function to handle form submission when user saves changes
  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      const authtoken = sessionStorage.getItem("auth-token");
      const email = sessionStorage.getItem("email"); // Get the email from session storage
      if (!authtoken || !email) {
        navigate("/login");
        return;
      }
      const payload = { ...updatedDetails };
      const response = await fetch(`${API_URL}/api/auth/user`, {
        method: "PUT",
        headers: {
          "Authorization": `Bearer ${authtoken}`,
          "Content-Type": "application/json",
          "Email": email,
        },
        body: JSON.stringify(payload),
      });
      if (response.ok) {
        // Update the user details in session storage
        sessionStorage.setItem("name", updatedDetails.name);
        sessionStorage.setItem("phone", updatedDetails.phone);
        setUserDetails(updatedDetails);
        setEditMode(false);
        // Display success message to the user
        alert(`Profile Updated Successfully!`);
        navigate("/");
      } else {
        // Handle error case
        throw new Error("Failed to update profile");
      }
    } catch (error) {
      console.error(error);
      // Handle error case
    }
  };
  // Render the profile form with different sections based on edit mode
  return (
    <div className="profile-container">
      {editMode ? (
        <form onSubmit={handleSubmit}>
          <label>
            Email
            <input
              type="email"
              name="email"
              value={userDetails.email}
              disabled // Disable the email field
            />
          </label>
          {/* Create similar logic for displaying and editing name and phone from userDetails */}
          <button type="submit">Save</button>
        </form>
      ) : (
        <div className="profile-details">
          <h1>Welcome, {userDetails.name}</h1>
          {/* Implement code to display and allow editing of phone and email similar to above */}
          <button onClick={handleEdit}>Edit</button>
        </div>
      )}
    </div>
  );
};
// Export the ProfileForm component as the default export
export default ProfileForm;
```

3. When executing the code, you need to run `node index` after navigating to the `server` folder. Before implementing the code, consider the following points:

    1. For code line number 3, you need to install React Router using command **npm i react-router-dom**.

        Refer to the [Capstone Project Technical Reference]() reading for details about the React Router.

4. At line number 6, `userDetails` object has been created using `useState( )` hook with three properties called name, phone, and email that need to be displayed both in read-only mode and in the edit mode, like a toggle button.

5. For edit mode, code at line number 8 is used to create useState() hook for **editmode**.

6. Two methods have been created one to set displayed details in edit mode using handleEdit at line number 48 to and other to handle submit form details during changing profile details at line number 58.

7. Using these details, editMode has been created from line number 103-126, within return of ProfileCard component. You need to apply logic at line number 114 create logic for name and phone from userDetails and at 120, implement code to display detail of phone and email like above.

▶ Click here for a **hint** for line number 114.
▶ Click here for a **hint** for line number 120.

8. **Take a screenshot** of the integration and save it as **integration.png**. Also show details in ProfileCard component and save it as **profileform.png**. Also, take the screenshot of changed name as **profilename_change.png**

▶ Click here to view the **sample Profile form**.

9. Import the **ProfileCard** component into the relevant page or component where you want to display the reports.

Exercise 3: Style the ProfileCard

1. Apply appropriate CSS styles to the **ProfileCard** component to make it visually appealing and ensure it will fit well within the layout of your application.

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

2. **Take a screenshot** of the output and save it as **profile.png**.

Exercise 4: Create the ReportsLayout component

1. In case you exited and are re-entering the Skills Network lab environment:

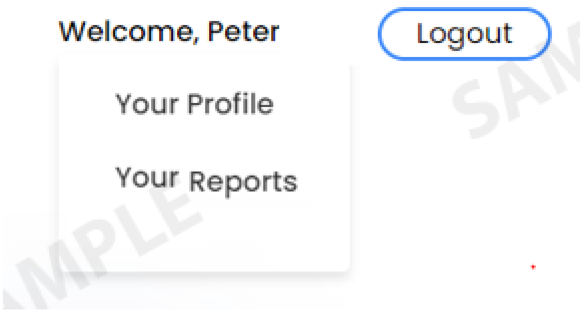
    1. Open new terminal.
    2. Clone your React project's GitHub repository.

2. Then, create the **ReportsLayout** folder within **components** folder in **src** folder. Create the ReportsLayout.js component and the ReportsLayout.css file.

    src->components->ReportsLayout->ReportsLayout.js,ReportsLayout.css

3. Define the **ReportsLayout** component using the React Function component syntax.

4. You can add drop-down item as “Your Reports” along with Profile, when a user clicks on its name.



5. Implement the necessary layout elements to display the ReportLayout components in the required format, such as a table, grid, or list.

▶ Click here to view the **sample Reports Layout**.

6. **Take a screenshot** of the report layout and save it as **reportlayout.png**.

7. Remember to perform `git add`, `git commit`, and `git push` all changes in your forked GitHub repository.

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

Exercise 5: Add functionality to view and download reports

1. In case you exited and are re-entering the Skills Network lab environment:

   1. Open new terminal.
   2. Clone your React project's GitHub repository.

2. Within the **ReportsLayout** component, implement the functionality to view reports by adding clickable elements or buttons that open the reports in a new tab or window.

3. You can create the report design using Figma. After creating the design, you can convert the Figma image to the pdf format. Then, save the file as **patient_report.pdf**. Make sure the pdf file is present in a public folder to ensure the file can be accessed from anywhere within the React project.

4. Implement functionality to download the reports. This can be achieved by adding download attribute in anchor tag.

5. **Take a screenshot** of the output and save it as **report.png**.

6. Remember to perform `git add`, `git commit` and `git push` all changes in your forked GitHub repository.

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

Exercise 6: Style the ReportsLayout component

1. In case you exited and are re-entering the Skills Network lab environment:

   1. Open new terminal.
   2. Clone your React project's GitHub repository.

2. Apply appropriate CSS styles to the ReportsLayout component to make it visually appealing and ensure it fits well within the layout of your application.

3. Consider using CSS frameworks or custom styles to enhance the user experience for the component.

4. **Take a screenshot** of the code snippet and save it as **style.png**

5. Remember to perform `git add`, `git commit`, and `git push` all changes in your forked GitHub repository.

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

Exercise 7: Integrate the ReportsLayout component

1. Import the **ReportsLayout** component into the relevant page or component where you want to display the reports.
2. Render the **ReportsLayout** component within the appropriate location in your application.

Exercise 8: Test and refine the components

1. Test the **ProfileCard** and **ProfileForm** components by updating the profile data, ensuring changes are reflected in the card and validations are applied.
2. Test the **ReportsLayout** component and verify that the report displays correctly and the view and download functionality works as expected.
3. Validate the behavior of the components in different scenarios, such as when the user has no reports or when encountering errors in the profile form.
4. **Take a screenshot** of the output and save it as **test_profile.png**.

Remember to continuously test, refine, and iterate on your components to ensure they work smoothly and meet your requirements.

Screenshot checklist

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

- *profilecard.png*
- *integration.png*
- *profileform.png*
- *profilename_change.png*
- *profile.png*
- *reportlayout.png*
- *report.png*
- *style.png*
- *test_profile.png*

Note about data management and persistence

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