CS628 Full-Stack Development – Web App

**HOS06A: React Router**

Created by Veerendra Jagatha on Aug 03, 2023

Review by Sam Chung on Aug 0, 2023

School of Technology & Computing (STC)

City University of Seattle (CityU)

**Before You Start**

* **Screenshots may be different from your environment.**
* The directory path shown in screenshots may be different from yours.
* Version numbers may not match the most current version at the time of writing. If given the option to choose between the stable release (long-term support) or the most recent, please select the **stable release** rather than the beta-testing version.
* There might be subtle discrepancies along with the steps. Please **use your best judgment** while going through this cookbook-style tutorial to complete each step.
* If you are not familiar with a terminal, command line, and bash scripts, check out this video: <https://youtu.be/Dp7uw9c6QH8>
* All the steps and concepts in this tutorial are from references, so if you encounter problems, please **try to read and compare the references to solve the problem**. If you still can't solve the problem, please contact your course TA.
* **Avoid copy-pasting code from the book or the GitHub repository**. Instead, type out the code yourself. Resort to copy-pasting only when you are stuck and find things not working as expected.
* Some steps may not be explained in detail. If you are not sure what to do:

1. Consult the resources from the course.
2. If you cannot solve the problem after a few tries (usually 15 -30 minutes), ask a TA for help.

#### **Readings and Examples:**

* Visit [CS 628 Repository for Examples](https://github.com/samchung0117/cs628-examples).
  + Select the related module.
  + Visit the README.md file.
  + Find examples for your practices.

**Learning Outcomes**

* Section 1: Accessing GitHub Codespaces
* Section 2: React Router using BrowserRouter
* Section 3: MemoryRouter
* Section 4: HashRouter
* Section 5: Nested Routing
* Section 6: Querying Parameters
* Section 7: Using NavLink
* Section 8: Redirect using Navigate
* Section 9: History Management
* Section 10: Pushing your work to GitHub

**Section 1: Accessing GitHub Codespaces**

Refer the steps from [TA Center](https://cityuseattle.github.io/docs/git/github_codepsace/) to get started with this week’s module GitHub Codespace.

Once the codespace is ready, in the terminal type the commands below to start a new react project and start it.

**>>npx create-react-app hos06**

**>> cd hos06**

**Note: After each section ensure to take the screenshots and save it in the module folder with the section number.**

**Section 2: React Router using BrowserRouter**

Reference:

* <https://reactrouter.com/en/main/router-components/browser-router#browserrouter>
* <https://reactrouter.com/en/main/components/routes>
* <https://reactrouter.com/en/main/route/route>
* <https://reactrouter.com/en/main/components/link#link>

React Router is a widely used library in the React ecosystem that facilitates efficient client-side routing for single-page applications (SPAs). It enables developers to create navigable user interfaces within a single HTML page, eliminating the need for full-page reloads. By defining routes and matching them to specific components, React Router dynamically renders the appropriate content based on Uniform Resource Locator (URL) changes, resulting in a seamless and fluid user experience. This library supports advanced features like nested routing, query parameter handling, and programmatic navigation, making it a powerful tool for managing client-side routing in React applications.

A <BrowserRouter> stores the current location in the browser's address bar using clean URLs and navigates using the browser's built-in history stack. “BrowserRouter” is typically used as the top-level router component in the application and serves as the entry point for defining “routes” and mapping them to specific components.

Before we begin, let us import the router module into our solution. Use the following commands in the “Terminal” to import the “react-router module” and start the development server.

**>>npm install react-router-dom**

**>> npm start**

Update your “index.css” file under the “src” folder with the styling code from the “style.css” from the “section2” in the “client-src-examples” in the “Module 06.”

Now we need to create components to accommodate routing in the application. Create three files named “Home.js”, “About.js” & “Details.js” and type the code from the “section2.”

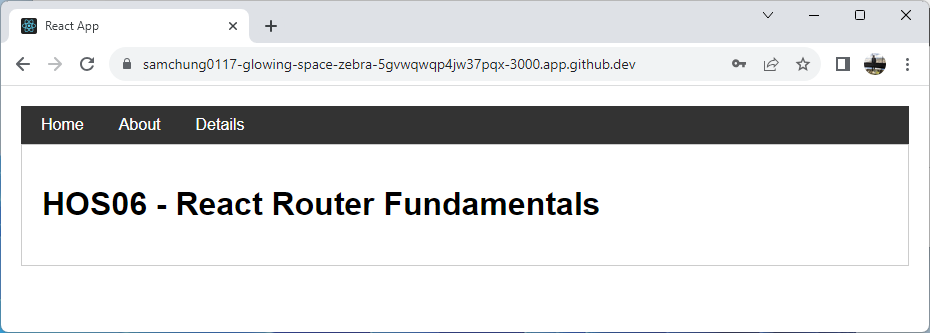
A screenshot of a computer program

Description automatically generated

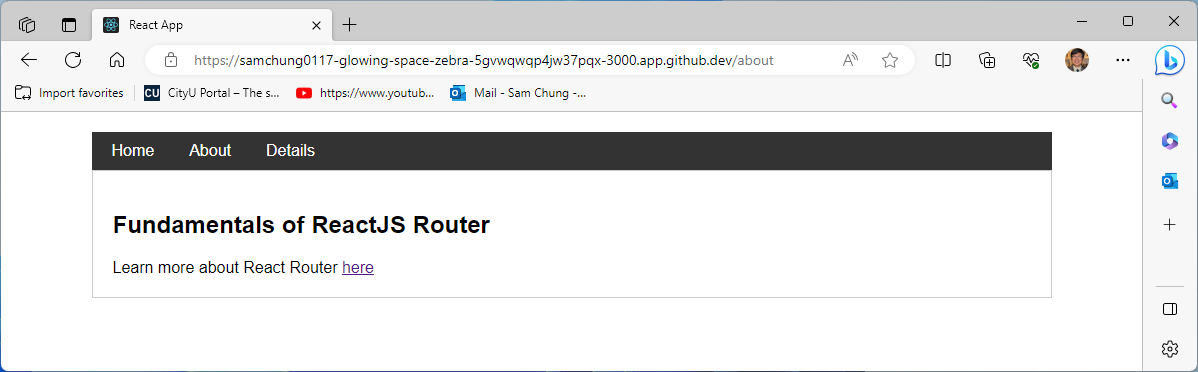
Now, incorporate the code provided in the "BrowserRouter.js" file from the examples folder into your "App.js" file. After making the necessary changes, refresh the development browser window to observe the updates we implemented.

The resulting screen should resemble the image below. You will notice the URL changing as you navigate using the “Link” component's "to" props.

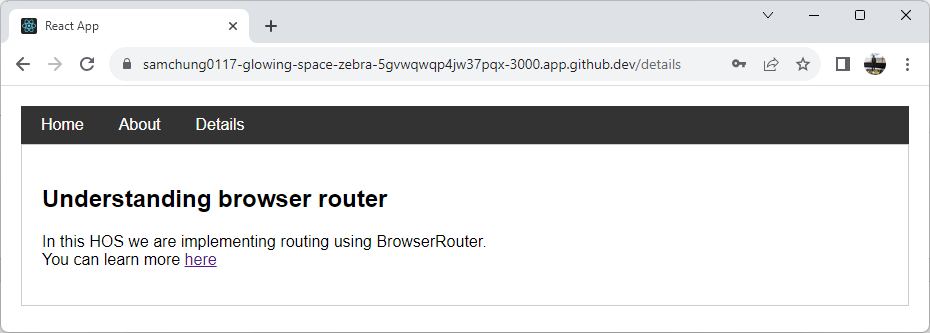
“<Link to="/">Home</Link>”



“<Link to="/about">About</Link>”



“<Link to="/details">Details</Link>”



**Section 3: MemoryRouter**

**Reference:**

* <https://reactrouter.com/en/main/router-components/memory-router#memoryrouter>

“MemoryRouter” stores URL changes in memory, not the user's browser. It keeps a history of URLs internally, bypassing the browser's address bar and disabling the back and forward buttons. This is beneficial for testing and non-browser environments, such as React Native, where browser-specific navigation features are not needed. It allows developers to simulate URL changes and test routing behavior without affecting the actual browser history.

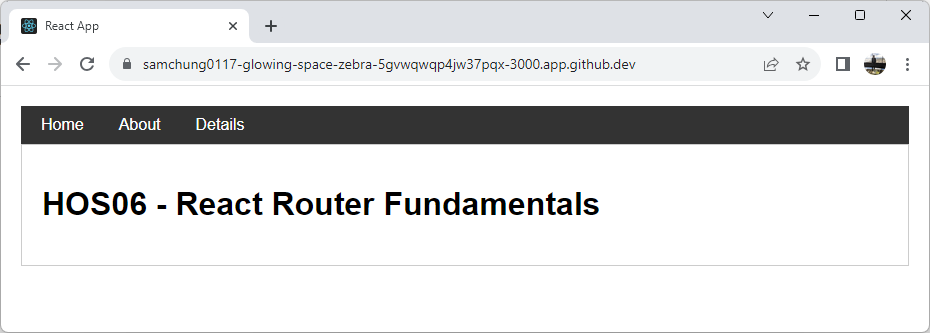
Update your “App.js” code as below to use “MemoryRouter” instead of “BrowserRouter.”

A screen shot of a computer program

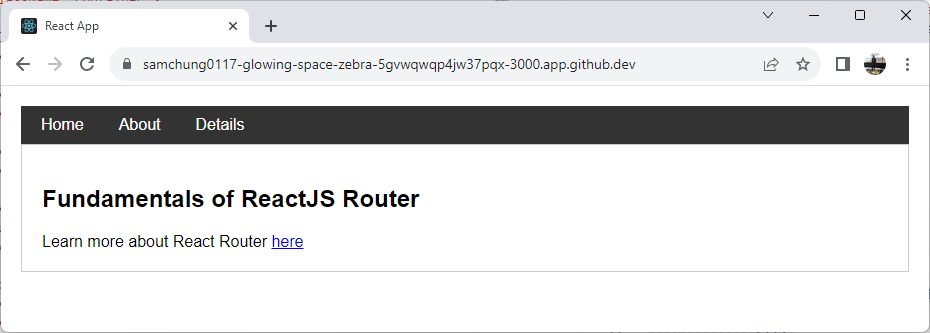
Description automatically generated

Refresh the browser window to see the changes. Try navigating to other pages in the application. You should see that the URL doesn’t change this time.

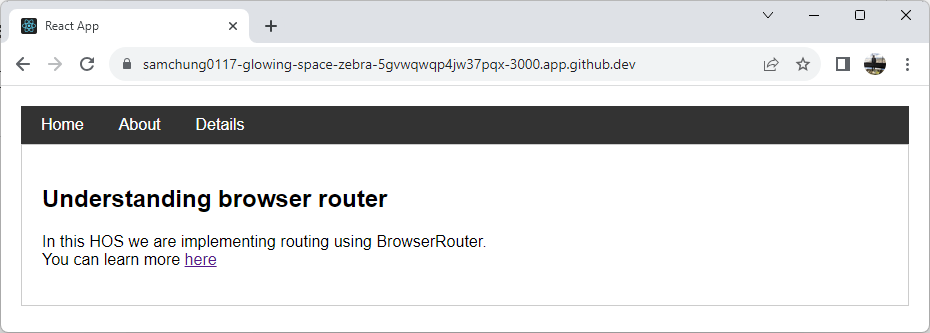
“<Link to="/">Home</Link>”



“<Link to="/about">About</Link>”



“<Link to="/details">Details</Link>”



**Section 4: HashRouter**

**Reference:**

* <https://reactrouter.com/en/main/router-components/hash-router#hashrouter>

<HashRouter> is a routing component provided by the “react-router-dom” that uses URL hashes to enable client-side navigation in single-page applications, particularly useful in scenarios where server-side handling of URL changes is not available, allowing for seamless route changes and browser back/forward button support. The server disregards the hash portion of the URL and always serves the “index.html” file for every request, leaving the handling of the hash value solely to the client-side application.

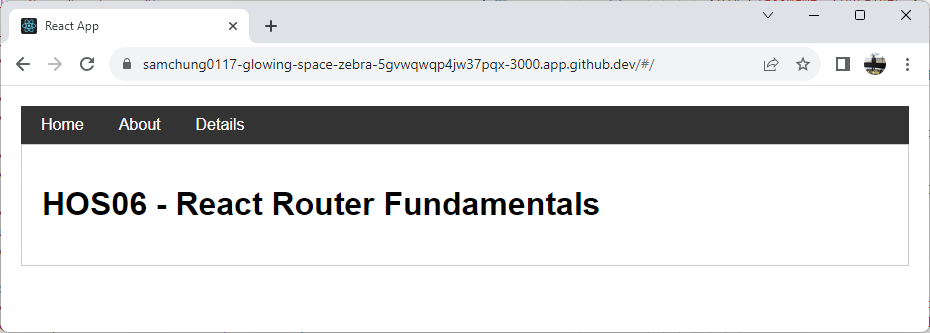
Update your “App.js” code as below to use “HashRouter” instead of “MemoryRouter.”

A screen shot of a computer program

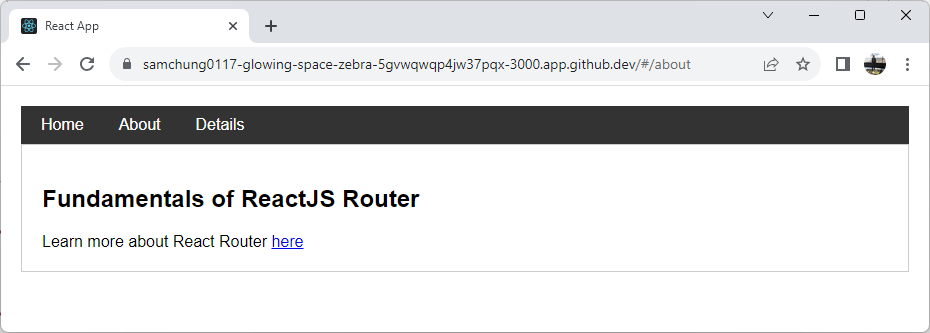
Description automatically generated

Refresh the browser window to notice the changes. Try navigating to other pages in the application. You should see # in the URL when routing to different pages.

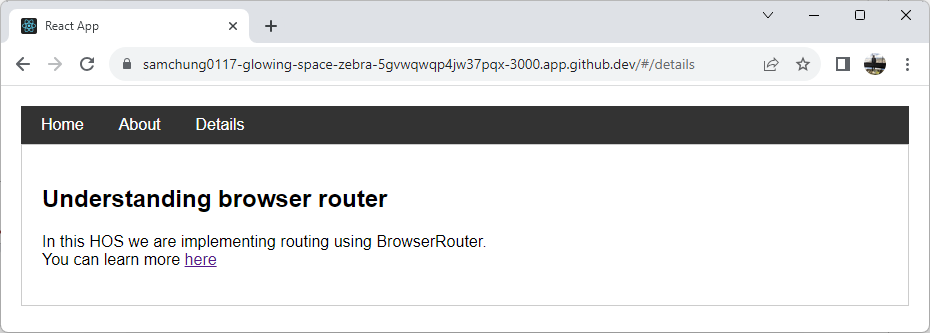
“<Link to="/">Home</Link>”



“<Link to="/about">About</Link>”



“<Link to="/details">Details</Link>”



**Section 5: Nested Routing**

Reference

* https://reactrouter.com/en/main/components/outlet#outlet

Nested routing in React Router defines routes within other routes, creating a hierarchical structure of components corresponding to specific URLs. With nested routing, you can render nested components based on the current URL path, allowing for more organized and modular code. Parent components can act as layouts, rendering common elements, while child components are swapped in and out based on the nested URL segments.

To accommodate nested routing in our application, let us create two more components, “Team.js” and “Member.js,” under the “src” folder.

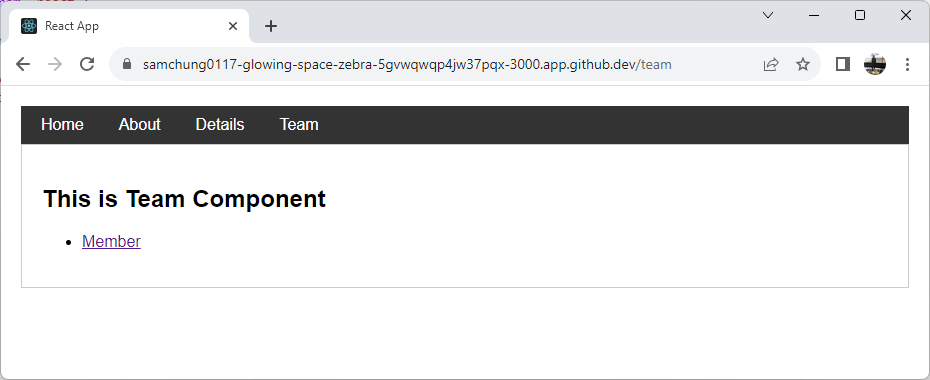
<Outlet> is a special component provided by the “react-router-dom” library that is used to render child routes within a parent route. You will find it under “Team.js.”

Type the code in Team and Member components from the “section5” folder. Update your “BrowserRouter” in “App.js” to match the following. You can get the entire “App.js” code from the samples folder.

A screenshot of a computer code

Description automatically generated

Refresh the dev browser, and now you will see a Team tab. Click on it and then click on Member to notice what happens.

“<Link to="/team">Team</Link>”  


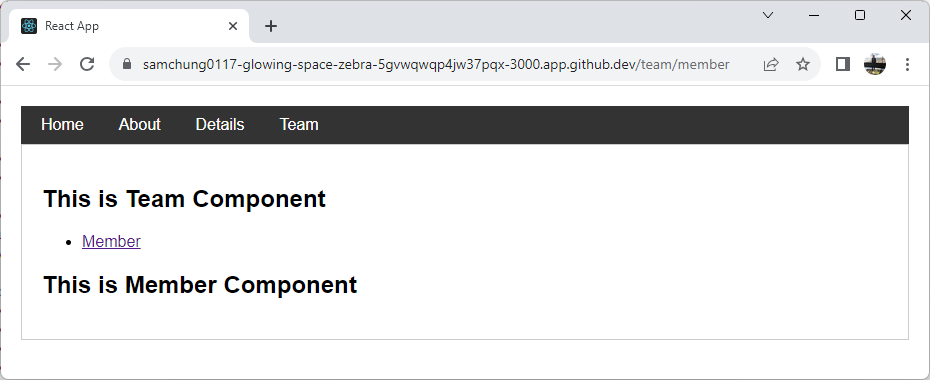
Here, “Member” is the child component rendered under the parent “Team” component using nested routing.

<Route path="/team" element={<Team />}>

{/\* Nested Routing \*/}

<Route path="member" element={<Member />} />

</Route>



**Section 6: Querying Parameters**

Reference:

* <https://reactrouter.com/en/main/hooks/use-params#useparams>

In React Router, “useParams” is a hook provided by the “react-router-dom” library that allows you to access and extract dynamic parameters from the URL. It is specifically used to retrieve values from route parameters defined in the route path.

When you define a route with a parameter placeholder in the path, such as **"/member/:id"**, the “useParams” hook allows you to extract the value of “**:id”** from the URL.

Let us see an example. Let’s modify our “Team” component to match the following,

A screenshot of a computer program

Description automatically generated

Update the “Member” component to query the parameters from the URL as shown below by using “useParams.”

A screen shot of a computer program

Description automatically generated

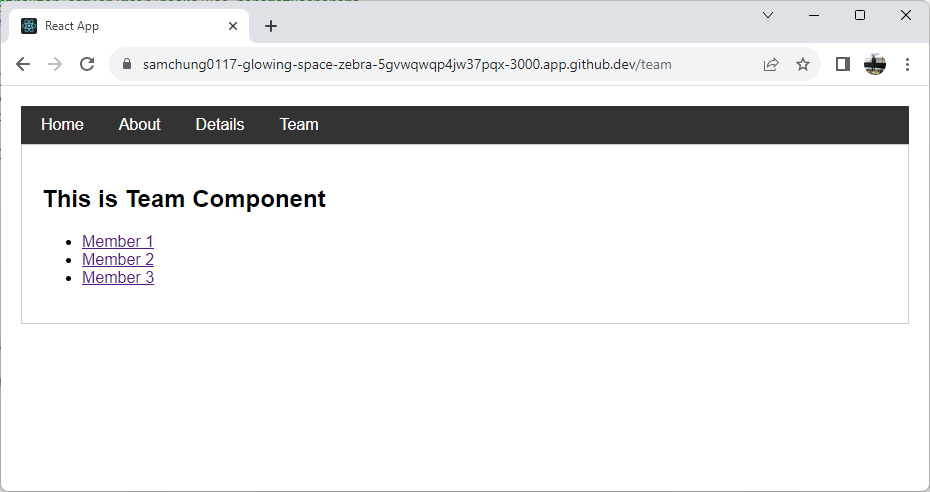
Let’s now update our “App.js” to accommodate the above routing changes,

A screen shot of a computer program

Description automatically generated

Now refresh the dev browser and navigate to the Team tab. Click on each “Member” and see what happens to the URL and the “Member” component.

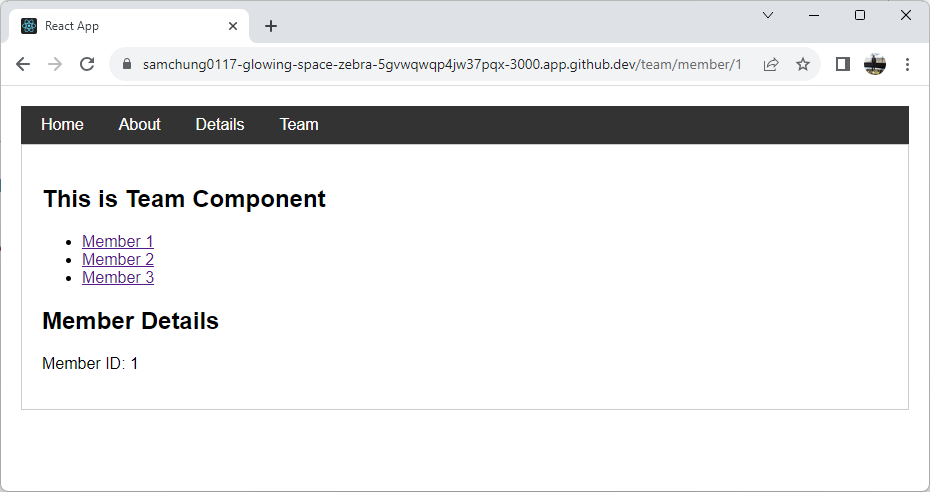
<Link to="/team">Team</Link>



<Route path="/team" element={<Team />}>

<Route path="member/:id" element={<Member />} />

</Route>



**Section 7: Using NavLink**

Reference:

* <https://reactrouter.com/en/main/components/nav-link#navlink>

In React Router, “NavLink” is a component provided by the “react-router-dom” library that is used to create navigation links within your application. It is like the regular Link component but comes with additional features specific to navigation.

The “NavLink” component allows you to define navigation links that can have an "active" state based on the current URL. When the user clicks on a “NavLink,” React Router automatically applies an "active" class to the link's rendered HTML element if the current URL matches the link's “to” prop.

Let us modify our “App.js” to use ‘NavLink” instead of Link,

A screenshot of a computer program

Description automatically generated

You can get the entire “App.js” code from “section7.”

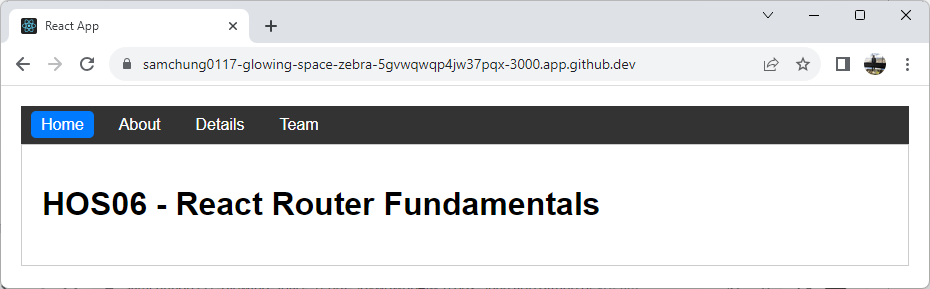
We now need to declare the active class in our “index.css” file. Match the styling code in the “index.css” file under the “src” folder with the “style.css” file under the hos-examples.

A screenshot of a computer code

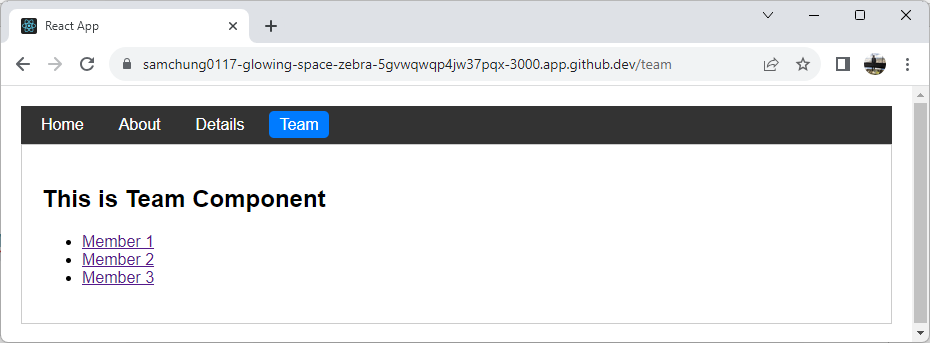
Description automatically generated

Now you will notice that the tab selected will get highlighted based on the styling we just defined.

<NavLink to="/" activeClassName="active">Home</NavLink>



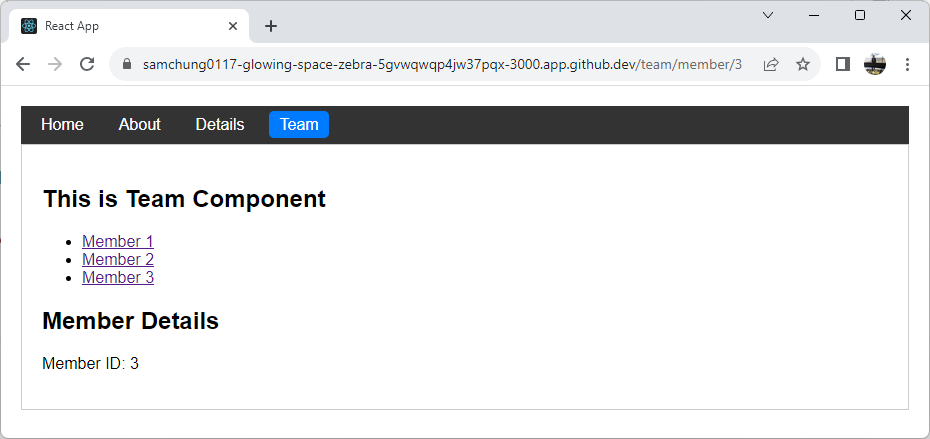
<NavLink to="/team" activeClassName="active">Team</NavLink>



<Route path="/team" element={<Team />}>

<Route path="member/:id" element={<Member />} />

</Route>



**Section 8: Redirect using Navigate**

Reference:

* <https://reactrouter.com/en/main/components/navigate#navigate>

In React Router, “Navigate” is a component provided by the “react-router-dom” library that is used to redirect users to a different route within the application programmatically. It allows you to navigate users to a specific URL without requiring any user interactions, such as clicking a link or button.

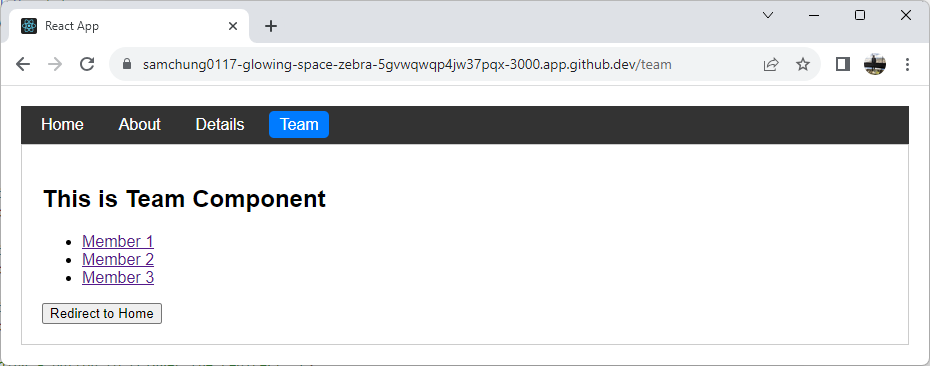
Let us implement “Navigate” under the “Team” component.

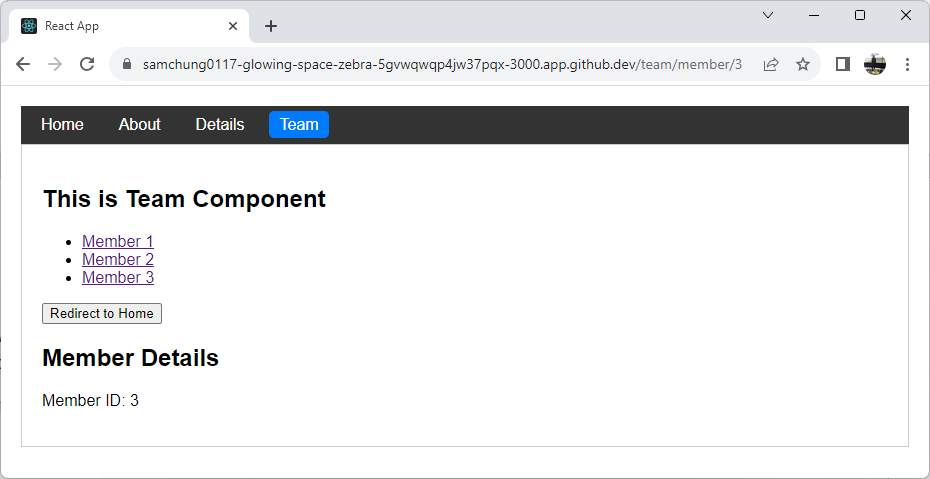
Update your “Team.js” to match the following code,

A screenshot of a computer program

Description automatically generated

When you click on the “Redirect” to “Home” button, you should be navigated to your “Home” component using the “Navigate.”





**Section 9: History Management**

Reference:

* <https://reactrouter.com/en/main/hooks/use-navigate#usenavigate>

The “useNavigate()” hook was introduced in React Router v6 as an alternative to the “useHistory()” hook used in earlier versions. With “useHistory(),” developers accessed the React Router history object and used push or replace methods for navigation. The hook enabled moving to specific URLs and navigating forward or backward in pages.

Let us update the “Team.js” to have the “Go Back” functionality and Redirect functionality to use the “useNavigate” hook.

A screenshot of a computer program

Description automatically generated

We will implement the “Go Forward” functionality in the “Details.js” file.

A screen shot of a computer code

Description automatically generated

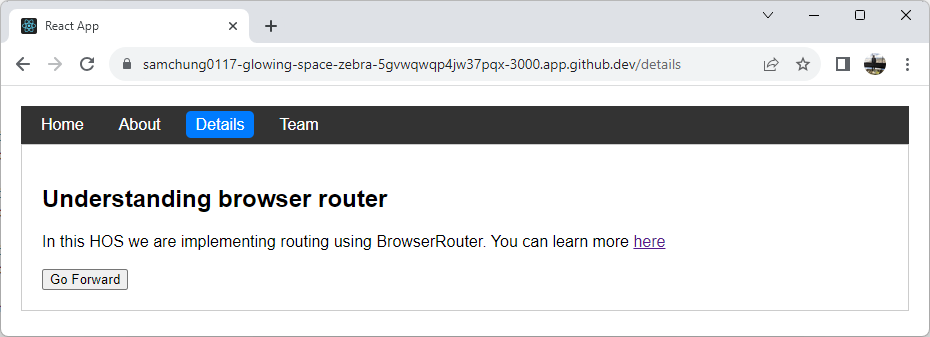
You can get all sample files from “section9.”

Now refresh the dev browser window to notice the changes. Play with the navigation.

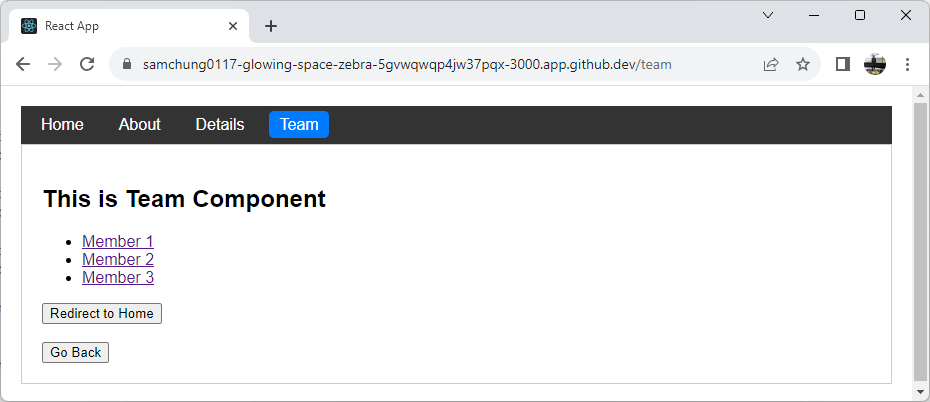
To observe the “go forward” functionality, navigate as below,

**>> Click the “Details” menu -> Click the “Team” menu -> Click the “Go Back” Button -> Click the “Go Forward” button.**

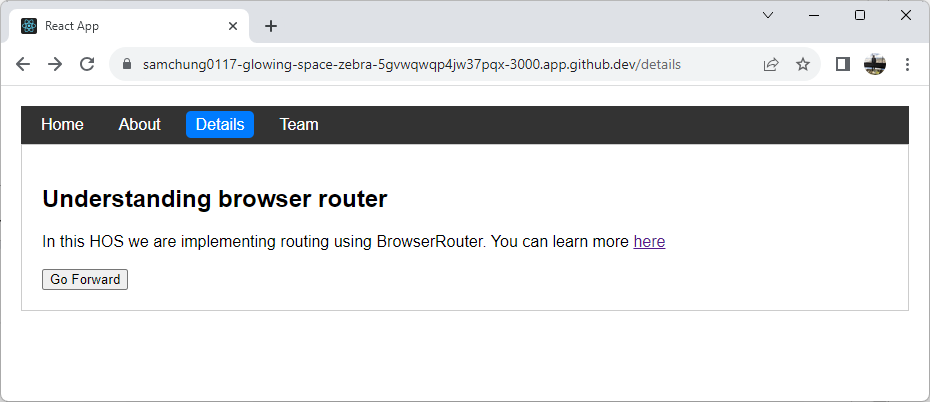
> “Details” menu



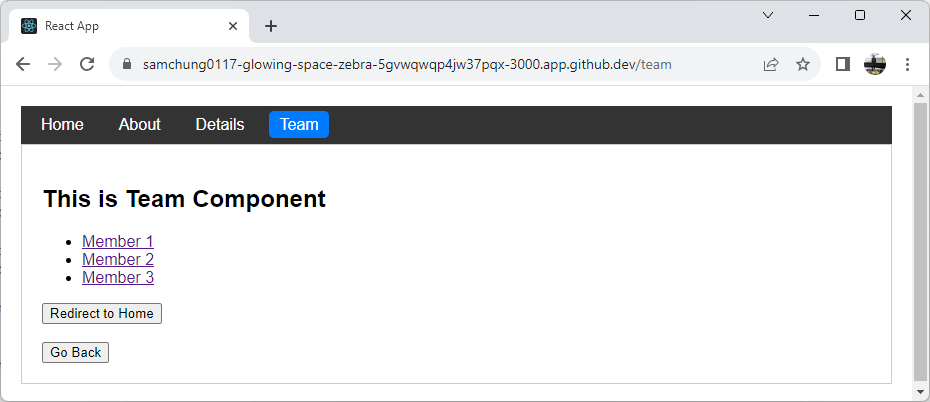
>”Team” menu



>”Go Back” button



>”Go Forward” button



**Section 10: Pushing your work to GitHub**

* 1. Go to Source Control on your GitHub Codespace and observe the pending changes.

Graphical user interface, text, application

Description automatically generated

* 1. Type the Message for your changes in the Message box on the top. For example,” **Submission for Module06 – Your Name**”
  2. Click on the dropdown beside the commit button and select **Commit & Push** to update the changes to your repository main branch.
  3. Select **Yes** when prompted.   
       
     Graphical user interface, application

     Description automatically generated  
       
       
     Following are the screenshots for learning outcome of HOS06  
     Section02  
     A close up of a computer screen

     Description automatically generatedA screenshot of a computer

     Description automatically generatedA screenshot of a computer

     Description automatically generated  
     Section03

A screen shot of a computer

Description automatically generated

A person holding a computer screen

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated  
Section04

A close up of a sign

Description automatically generated

A black and white screen with black text

Description automatically generated

A screenshot of a computer

Description automatically generated  
Section05

A close-up of a computer screen

Description automatically generated  
Section06

A screenshot of a computer

Description automatically generated

A white rectangular object with a black border

Description automatically generated

Section07

A screenshot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generatedA close-up of a computer screen

Description automatically generated  
  
Section 08

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Section09

A screen shot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generatedA white rectangular object with black border

Description automatically generatedA screenshot of a computer

Description automatically generated