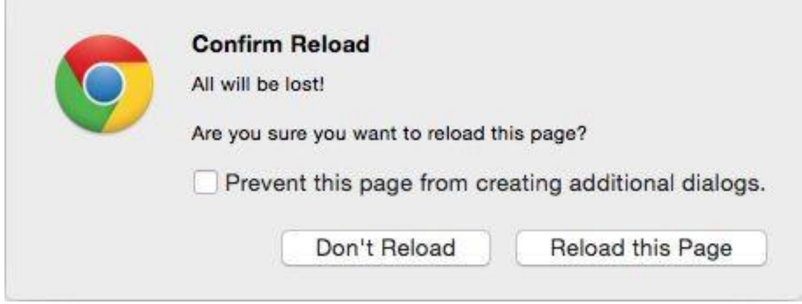
Single Page Applications and React Router

Web Apps and Browsers

- Web apps run in browsers (by definition)
- Users are use to browsing in browsers
  - Browser maintains a history of URLs visited
    - Back button - Go back in history to previous URL
    - Forward button - Go forward in history to next URL
  - Can move to a different page
    - Typing into location bar or forward/back buttons
    - Selecting a bookmarked URL
    - Page refresh operation
- Browser tosses the current JavaScript environment when navigating to a different page
  - Problematic for JavaScript frameworks: URL (and cookies) are the only information preserved

Problem with some web apps


Confirm Reload

All will be lost!

Are you sure you want to reload this page?

- Initial: pages served from web server
  - Each page had a URL and app switched between pages served by web server
- Early JavaScript apps: Website a single page/URL with the JavaScript

  Problem: Restart web app on navigation (Can lose a lot of work!)

  ```
  window.onbeforeunload = function(e) { return 'All will be lost!'; }
  ```
- Users expect app in browser to do the right thing:
  - Navigate with forward and back buttons, browser page history
  - Navigate away and come back to the app
  - Bookmark a place in the app
  - Copy the URL from the location bar and share it with someone
  - Push the page refresh button on the browser

Changing URL without page refresh

- Can change hash fragment in URL without reload

  `http://example.com`
  `http://example.com#fragment`
  `http://example.com?id=3535`
  `http://example.com?id=3535#fragment`

- HTML5 give JavaScript control of page reload
  - Browser History API - `window.history` - Change URL without reloading page

ReactJS support for SPA

- ReactJS has no opinion! Need 3rd party module.
- Example: React Router Version 5 https://v5.reactrouter.com/
  - Idea: Use URL to control conditional rendering
  - Newer version 6 is available using same concepts as v5 but slightly different syntax
- Various ways of encoding information in URL
  - In fragment part of the URL: `HashRouter`
  - Use HTML5 URL handler: `BrowserRouter`
- Import as a module:

```
import {HashRouter, Route, Link, Redirect} from 'react-router-
dom';
```

React Router v6

- Need to install the library: **`npm install react-router-dom`**
- Different routers in React Router DOM library
  - `BrowserRouter:` Handles routing by storing the routing context in the browser URL and implements backward/forward navigation with the inbuilt history stack
  - `HashRouter:` The `HashRouter` component doesn't send the current URL to the server by storing the routing context in the location hash (i.e., index.html#/profile)
  - `MemoryRouter`: Invisible router implementation that doesn't connect to an external location, such as the URL path or URL hash. The `MemoryRouter` stores the routing stack in memory but handles routing features like any other router

Creating routes with React Router v6

- The first component to import is `BrowserRouter`. It is used to wrap different routes. It uses the HTML5 history API to keep track of routes history in the React app.
- The next component to import from react-router-dom is the new `Routes`. It includes features like relative routing and linking, automatic route ranking, nested routes, and layouts.
- The last component from react-router-dom required is called `Route` and is responsible for rendering the UI of a React component.

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

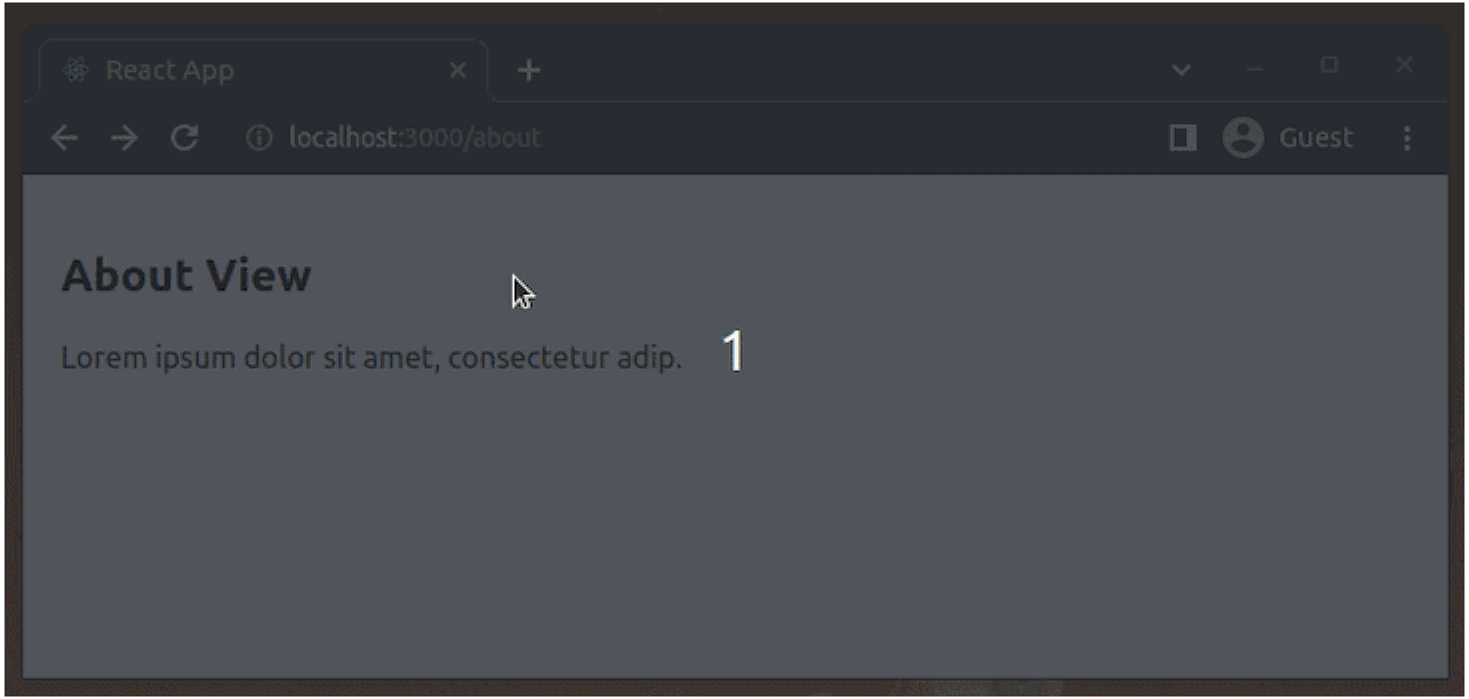
Building functional components

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

function Home() {
  return (
    <div style={{ padding: 20 }}>
      <h2>Home View</h2>
      <p>Lorem ipsum dolor sit amet, consectetur adip.</p>
    </div>
  );
}
function About() {
  return (
    <div style={{ padding: 20 }}>
      <h2>About View</h2>
      <p>Lorem ipsum dolor sit amet, consectetur adip.</p>
    </div>
  );
}
```

Building functional components

```
function NoMatch() {
  return (
    <div style={{ padding: 20 }}>
      <h2>404: Page Not Found</h2>
      <p>Lorem ipsum dolor sit amet, consectetur adip.</p>
  </div>
  );
}
function App() {
  return (
    <Router>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
      </Routes>
    </Router> );
}
export default App;
```

Building functional components



About View

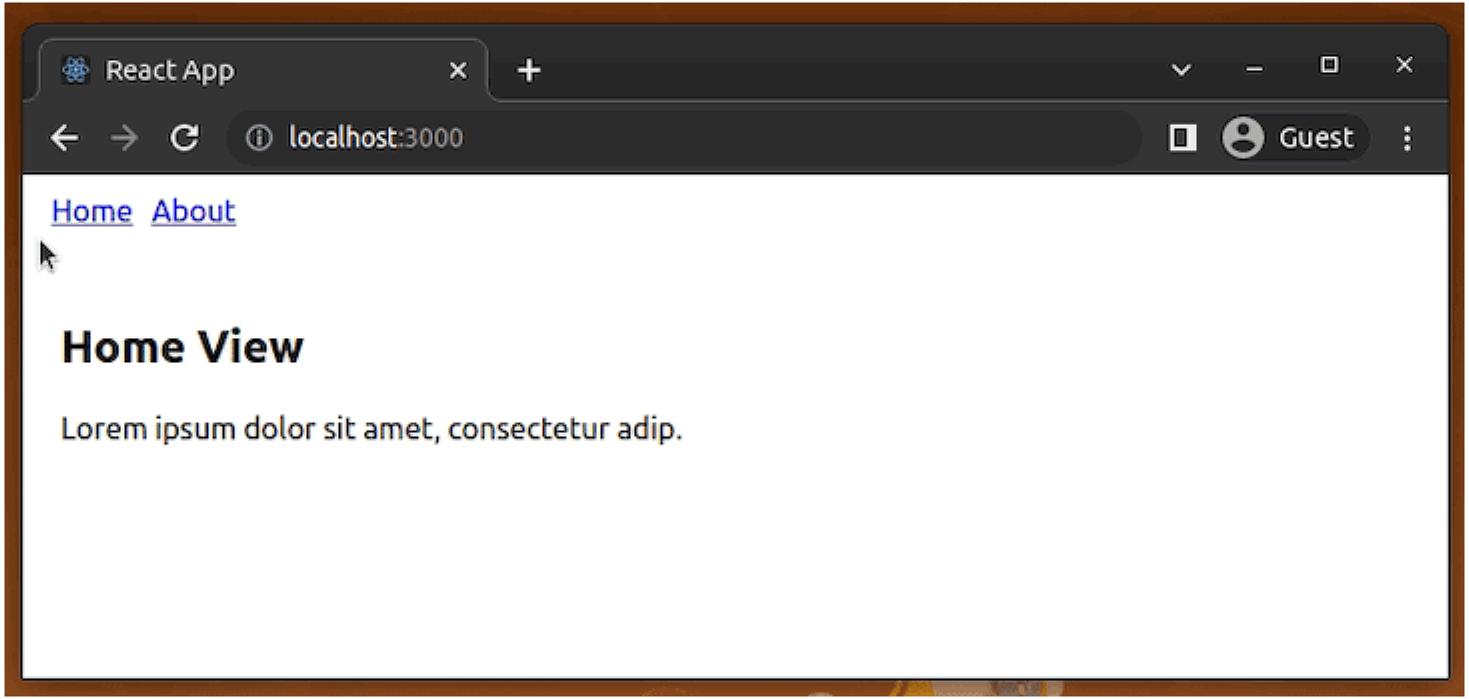
Lorem ipsum dolor sit amet, consectetur adip.

Adding a navigation menu

To avoid refreshing the webpages if using an <a> tag, the react-router-dom library provides the Link component

```
function App() {
  return (
    <Router>
      <nav style={{ margin: 10 }}>
        <Link to="/" style={{ padding: 5 }}> Home </Link>
        <Link to="/about" style={{ padding: 5 }}> About </Link>
      </nav>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
        <Route path="*" element={<NoMatch />} />
      </Routes>
    </Router>
  );
}
```

Adding a navigation menu



Home View

Lorem ipsum dolor sit amet, consectetur adip.

How to handle nested routes

- When routes are nested, a certain part of a webpage remains constant and only the child part of the webpage changes
  - In a simple blog, the title of the blog is always displayed, with a list of posts displayed beneath it.
  - When you click a post, the list of posts is replaced by the contents or the description of that specific post.
  - React Router library uses `Outlet` to render any matching children for a particular route with relative path definitions

```
import { BrowserRouter as Router, Routes, Route, Link, Outlet }

from 'react-router-dom';
```

How to handle nested routes

- To mimic a basic blog, let’s add some mock data in the App.js file.
    - Add the following BlogPosts constant to your App.js file’s beginning (after all imports)

```
const BlogPosts = {
    'first-blog-post': {
      title: 'First Blog Post’,
      description: 'Lorem ipsum dolor sit amet, consectetur adip.' },
    'second-blog-post': {
      title: 'Second Blog Post’,
      description: 'Hello React Router v6’
    }
};
```

How to handle nested routes

- Create a functional component called Posts, where a list of all posts is displayed.
    - The `Outlet` component definition will render child components based on nested routing definitions

```
function Posts() {
    return (
      <div style={{ padding: 20 }}>
        <h2>Blog</h2>
        <Outlet />
      </div>
    );
}
```

How to handle nested routes

- Define another component called PostLists
    - JavaScript `Object.entries()` method to return an array from the object `BlogPosts`

```
function PostLists() {
    return (
      <ul>
        {Object.entries(BlogPosts).map(([slug, { title }]) => (
          <li key={slug}>
            <h3>{title}</h3>
          </li> ))}
      </ul>
    );
}
```

How to handle nested routes

- Modify the routes in the `App` function component
  - The `index` prop for the `PostLists` route was used to specify the index of `/posts.` It will be rendered into their parent's Outlet at their parent's URL (like a default child route).

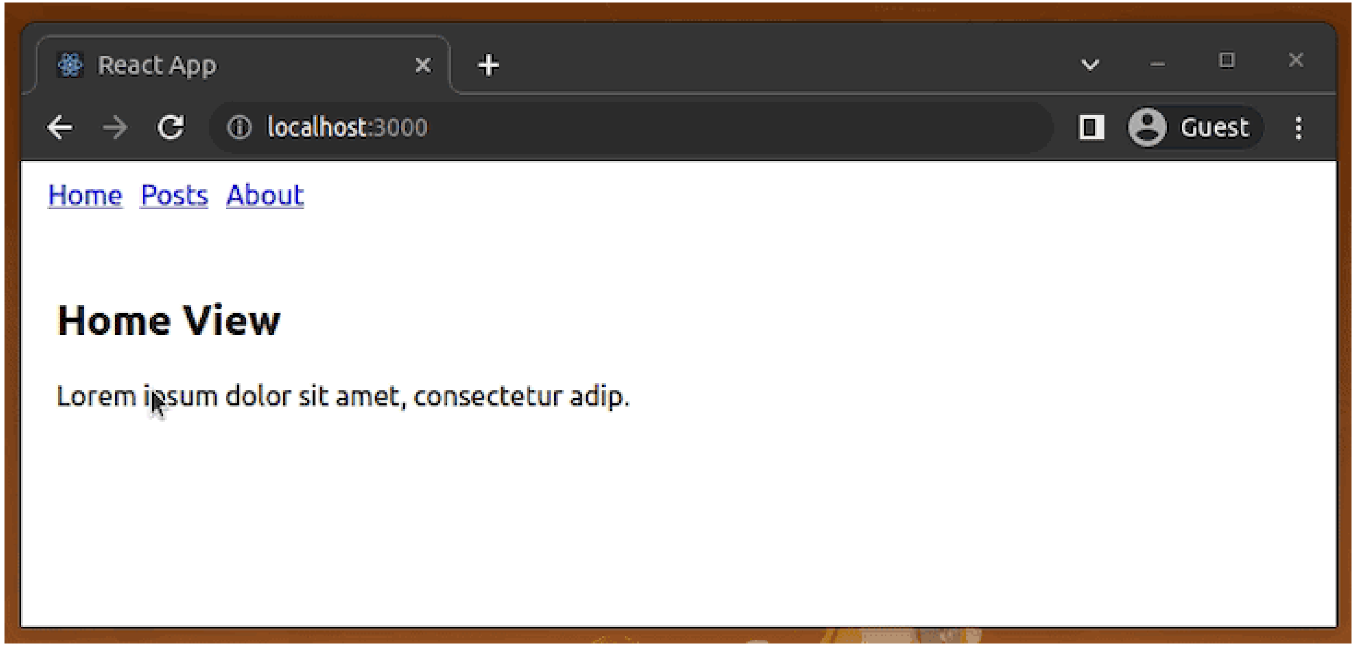
```
<Routes>
    <Route path="/" element={<Home />} />
    <Route path="/posts" element={<Posts />}>
      <Route index element={<PostLists />} />
    </Route>
    <Route path="/about" element={<About />} />
    <Route path="*" element={<NoMatch />} />
</Routes>
```

How to handle nested routes

- Update the navigation by adding a link to the Posts page
  - The `PostLists` child component was rendered within the `Posts` parent component via the library’s inbuilt `Outlet` component.

```
<nav style={{ margin: 10 }}>
    <Link to="/" style={{ padding: 5 }}> Home </Link>
    <Link to="/posts" style={{ padding: 5 }}> Posts </Link>
    <Link to="/about" style={{ padding: 5 }}> About </Link>
</nav>
```

How to handle nested routes



Home View

Lorem ipsum dolor sit amet, consectetur adip.

Accessing URL parameters and dynamic parameters of a route

- To visit the individual post by clicking the post title, wrap the title of each post within a `Link` component in the `PostsLists` component.
  - Define the path to each post using the `slug` of each post

```
function PostLists() {
    return (
      <ul>
        {Object.entries(BlogPosts).map(([slug, { title }]) => (
          <li key={slug}>
            <Link to={`/posts/${slug}`}> <h3>{title}</h3> </Link>
          </li>
        ))}
      </ul>
     );}
```

Accessing URL parameters and dynamic parameters of a route

- Create a new functional component called Post. This component is going to get the current slug of the post from `useParams` Hook

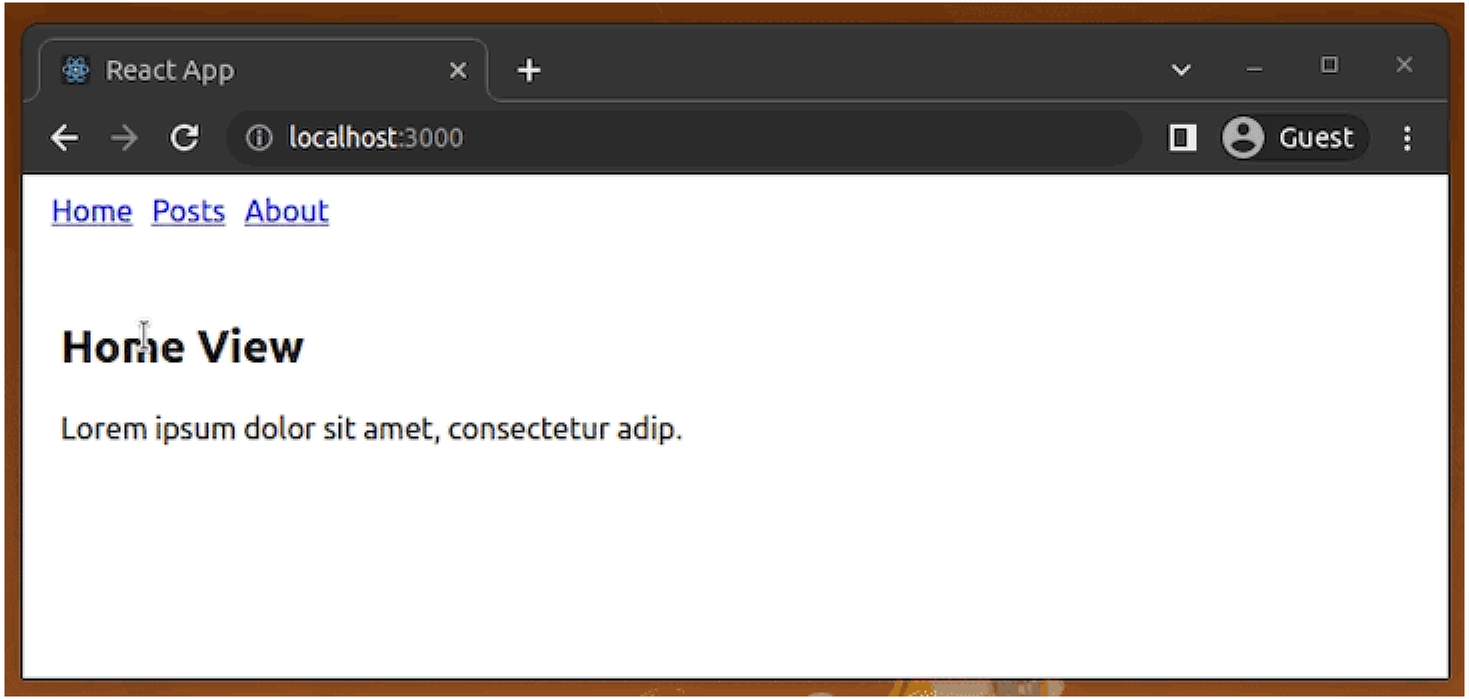
```
import {useParams} from react-router-dom

function Post() {

    const { slug } = useParams();

    const post = BlogPosts[slug];

    if(!post) {

      return <span>The blog post you've requested doesn't exist.</span>;  }

    const { title, description } = post;

    return ( <div style={{ padding: 20 }}> <h3>{title}</h3> <p>{description}
             </p> </div>

    );

}
```

Accessing URL parameters and dynamic parameters of a route

- Add a dynamic route called `:slug` in the `App` function component to render the contents of each post

```
<Route path="/posts" element={<Posts />}>
   <Route index element={<PostLists />} />
   <Route path=":slug" element={<Post />} />
</Route>
```

Accessing URL parameters and dynamic parameters of a route



Home View

Lorem ipsum dolor sit amet, consectetur adip.

How to protect routes

- Suppose we have a sample post statistics page that only authenticated admins can access

```
function Stats({ user }) {
    if(!user) {
      return ( <Navigate to="/login" replace/> );
    }
    return (
      <div style={{ padding: 20 }}>
        <h2>Stats View</h2>
        <p>Lorem ipsum dolor sit amet, consectetur adip.</p>
    </div>
    );
}
```

How to protect routes

```
function Login({ onLogin }) {
    const [creds, setCreds] = useState({});
    const navigate = useNavigate();
    function handleLogin() {
    // For demonstration purposes only.
      if(creds.username === 'admin' && creds.password === '123') {
        onLogin && onLogin({username: creds.username});
        navigate('/stats’);
    }
}
```

How to protect routes

```
function Login({ onLogin }) {
    const [creds, setCreds] = useState({});
    const navigate = useNavigate();
    function handleLogin() {
    // For demonstration purposes only.
      if(creds.username === 'admin' && creds.password === '123') {
        onLogin && onLogin({username: creds.username});
        navigate('/stats’);
      }
    }
```

How to protect routes

```
    return (
        <div style={{ padding: 10 }}> <br/>
        <span>Username:</span><br/>
        <input type="text" onChange={(e) => setCreds({...creds, username:
        e.target.value})}/><br/>
        <span>Password:</span><br/>
        <input type="password" onChange={(e) => setCreds({...creds, password:
        e.target.value})}/><br/><br/>
        <button onClick={handleLogin}>Login</button> </div>
    );
}
```

How to protect routes

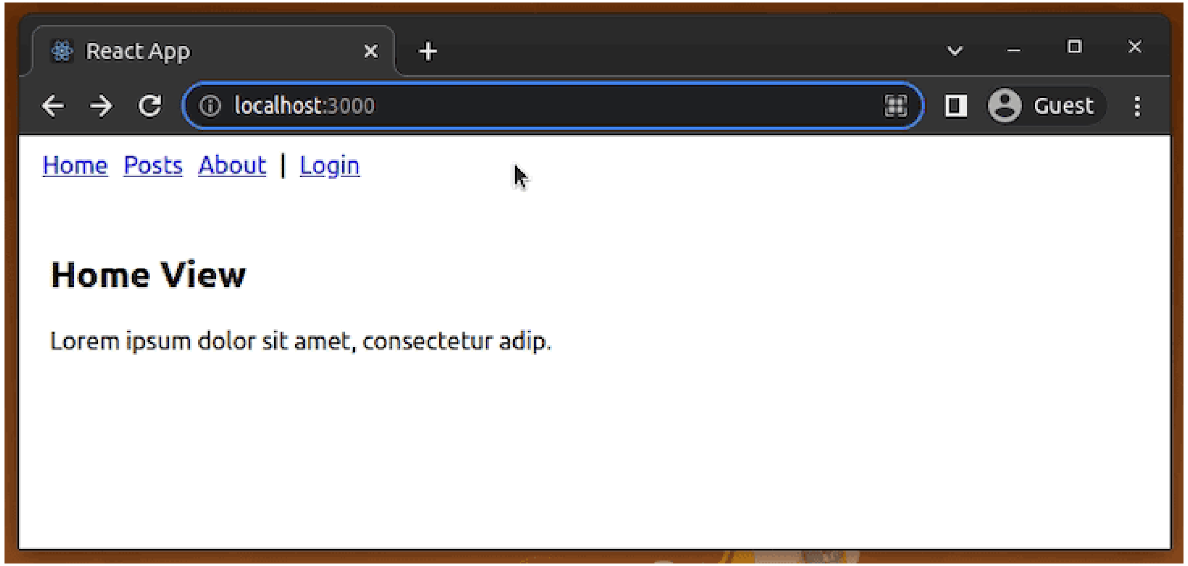
```
function AppLayout() {
  const [user, setUser] = useState();
  const navigate = useNavigate();
  function logOut() { setUser(null); navigate("/"); }
  return (
   <>
   <nav style={{ margin: 10 }}>
     <Link to="/" style={{ padding: 5 }}> Home </Link>
     <Link to="/posts" style={{ padding: 5 }}> Posts </Link>
     <Link to="/about" style={{ padding: 5 }}> About </Link>
     <span> | </span>
     { user && <Link to="/stats" style={{ padding: 5 }}> Stats </Link> }
     { !user && <Link to="/login" style={{ padding: 5 }}> Login </Link> }
     { user && <span onClick={logOut} style={{ padding: 5, cursor: 'pointer’}}>
          Logout </span> }
   </nav>
```

How to protect routes

```
                <Routes>
                  <Route path="/" element={<Home />} />
                  <Route path="/posts" element={<Posts />}>
                  <Route index element={<PostLists />} />
                  <Route path=":slug" element={<Post />} />
                  </Route> <Route path="/about" element={<About />} />
                  <Route path="/login" element={<Login onLogin={setUser}/>} />
                  <Route path="/stats" element={<Stats user={user}/>} />
                  <Route path="*" element={<NoMatch />} />
                </Routes>
              </>
          );
      }
```

How to protect routes

```
function App() {
  return (
    <Router>
      <AppLayout/>
    </Router>
  );
}
```



Home View

Lorem ipsum dolor sit amet, consectetur adip.