Requirement 2 - Build Frontend React App

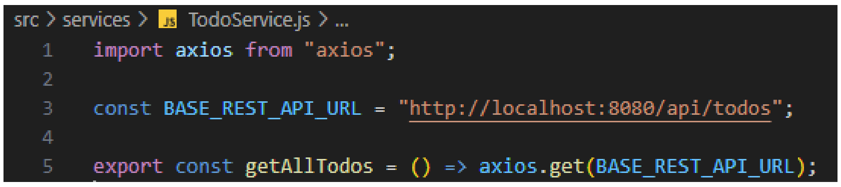
Build Frontend React App (Consume REST APIs) for Todo Management Module

User should able perform below operations:

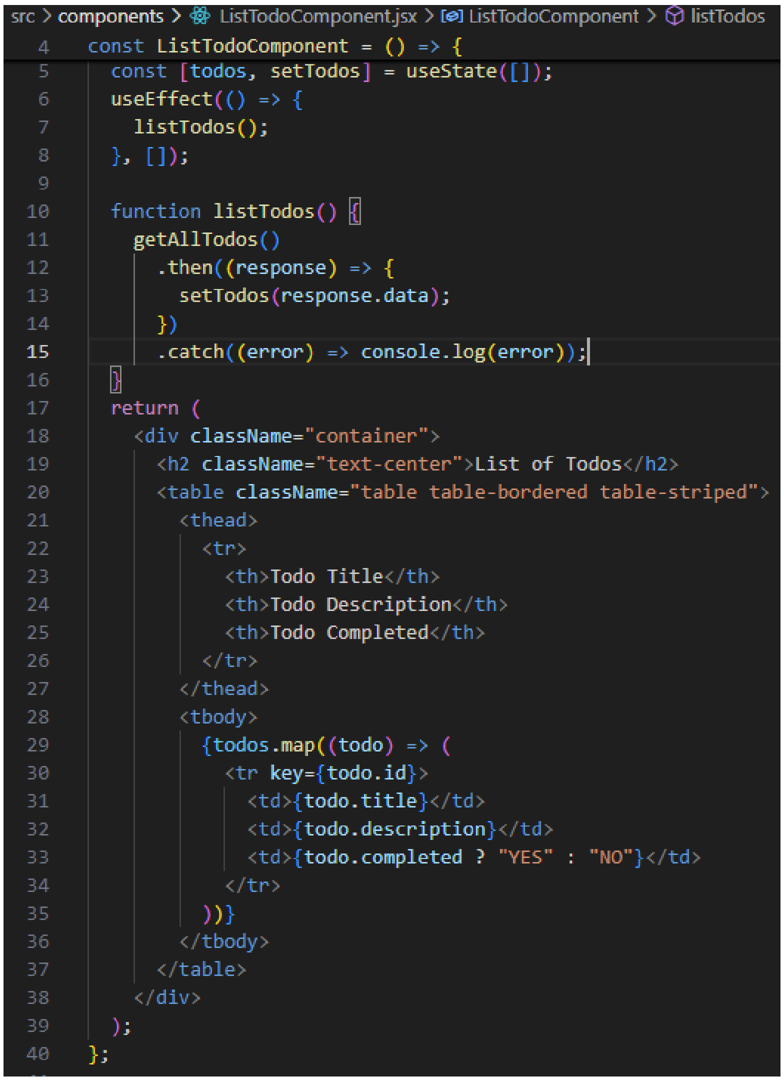
- Add new todo
- List all todos in a table
- Update particular todo
- Delete particular todo
- Mark todo as complete
- Mark todo as incomplete

Development Steps

1. Install **axios** Library
2. Create **TodoService.js** File
3. Write REST Client code to make a REST API call using **axios** API
4. Change **ListTodoComponent** to Display Response of the REST API (List of Todos)



```
import axios from "axios";

const BASE_REST_API_URL = "http://localhost:8080/api/todos";

export const getAllTodos = () => axios.get(BASE_REST_API_URL);
```

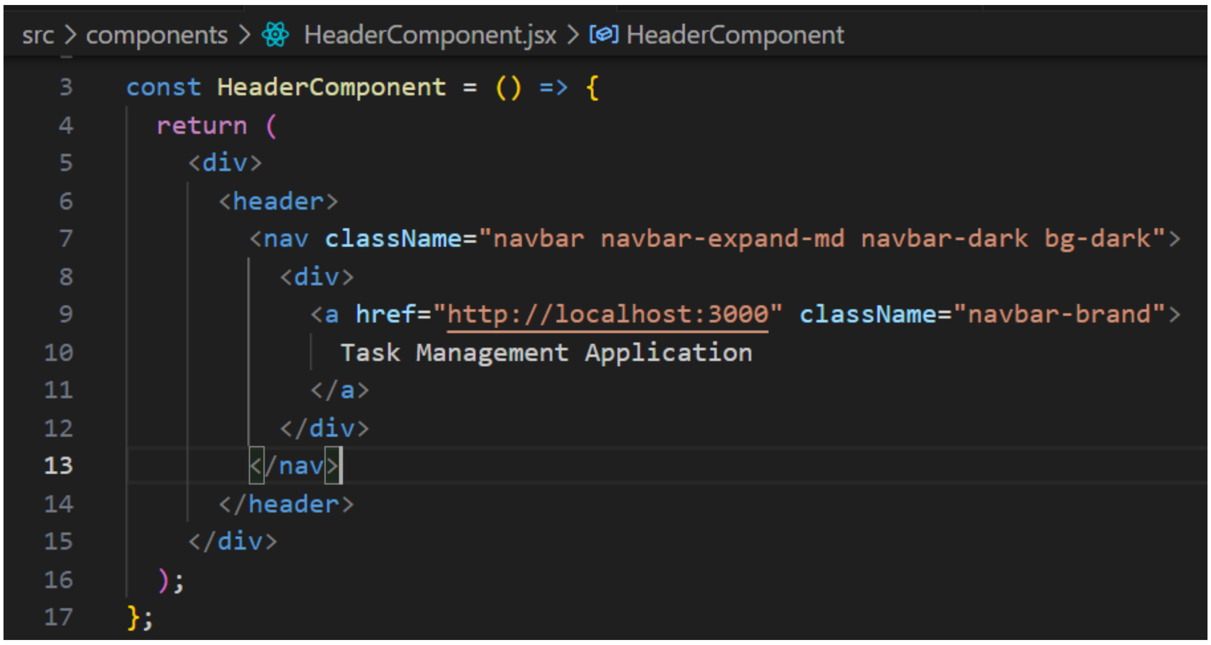

src > components > ListTodoComponent.jsx > ListTodoComponent > listTodos

```
const ListTodoComponent = () => {
  const [todos, setTodos] = useState([]);
  useEffect(() => {
    listTodos();
  }, []);

  function listTodos() {
    getAllTodos()
      .then((response) => {
        setTodos(response.data);
      })
      .catch((error) => console.log(error));
  }
  return (
    <div className="container">
      <h2 className="text-center">List of Todos</h2>
      <table className="table table-bordered table-striped">
        <thead>
          <tr>
            <th>Todo Title</th>
            <th>Todo Description</th>
            <th>Todo Completed</th>
          </tr>
        </thead>
        <tbody>
          {todos.map((todo) => (
            <tr key={todo.id}>
              <td>{todo.title}</td>
              <td>{todo.description}</td>
              <td>{todo.completed ? "YES" : "NO"}</td>
            </tr>
          ))}
        </tbody>
      </table>
    </div>
  );
};
```

Adding Header and Footer to React App

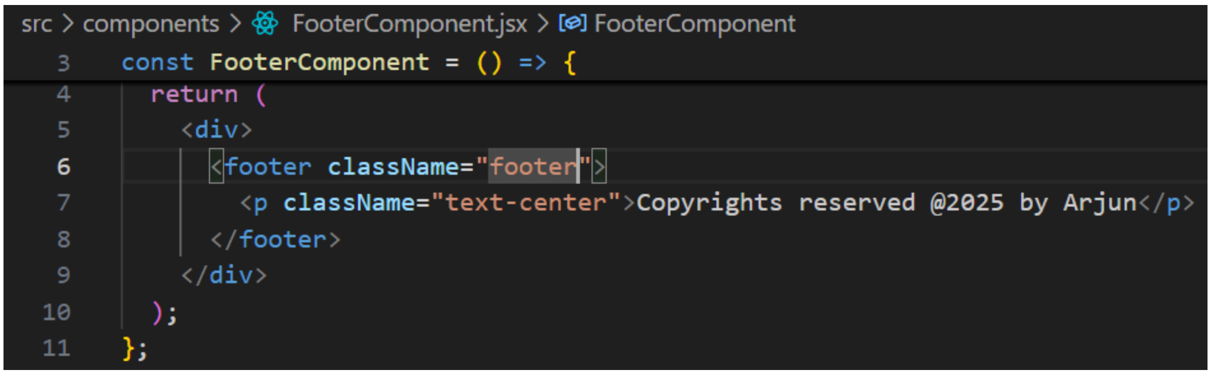
Development Steps

1. Create HeaderComponent (functional component)
2. Import and Use HeaderComponent in App Component
3. Create FooterComponent (functional component)
4. Import and Use FooterComponent in App Component



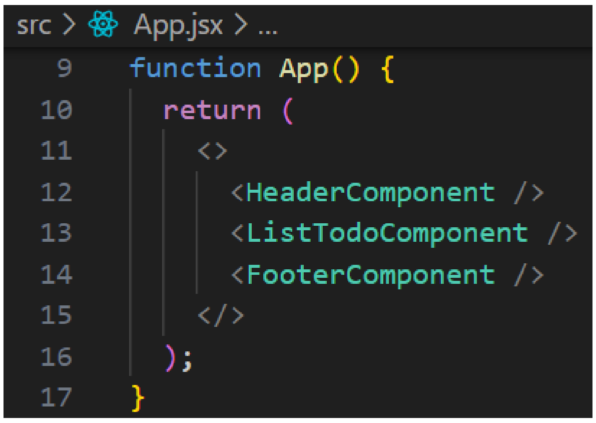
src > components > HeaderComponent.jsx > HeaderComponent

```jsx
const HeaderComponent = () => {
  return (
    <div>
      <header>
        <nav className="navbar navbar-expand-md navbar-dark bg-dark">
          <div>
            <a href="http://localhost:3000" className="navbar-brand">
              Task Management Application
            </a>
          </div>
        </nav>
      </header>
    </div>
  );
};
```



src > components > FooterComponent.jsx > FooterComponent

```jsx
const FooterComponent = () => {
  return (
    <div>
      <footer className="footer">
        <p className="text-center">Copyrights reserved @2025 by Arjun</p>
      </footer>
    </div>
  );
};
```



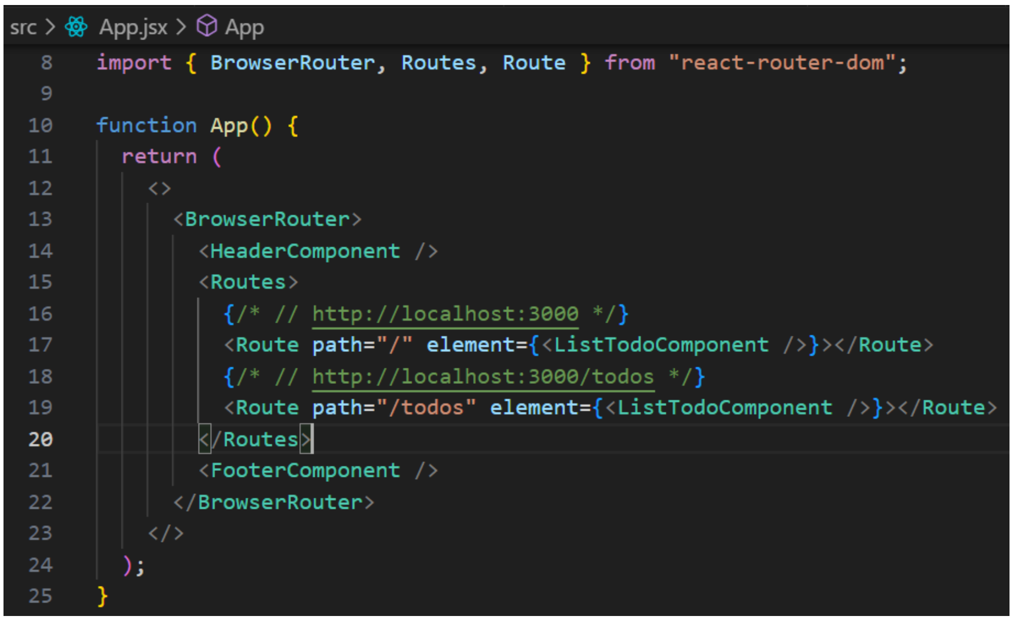
src > App.jsx > ...

```jsx
function App() {
  return (
    <>
      <HeaderComponent />
      <ListTodoComponent />
      <FooterComponent />
    </>
  );
}
```

Configure Routing in React App

Development Steps

1. Install **react-router-dom** library using NPM
2. Configure Routing in **App** Component
3. Configure Route for **ListTodoComponent**
4. Test Route for **ListTodoComponent**

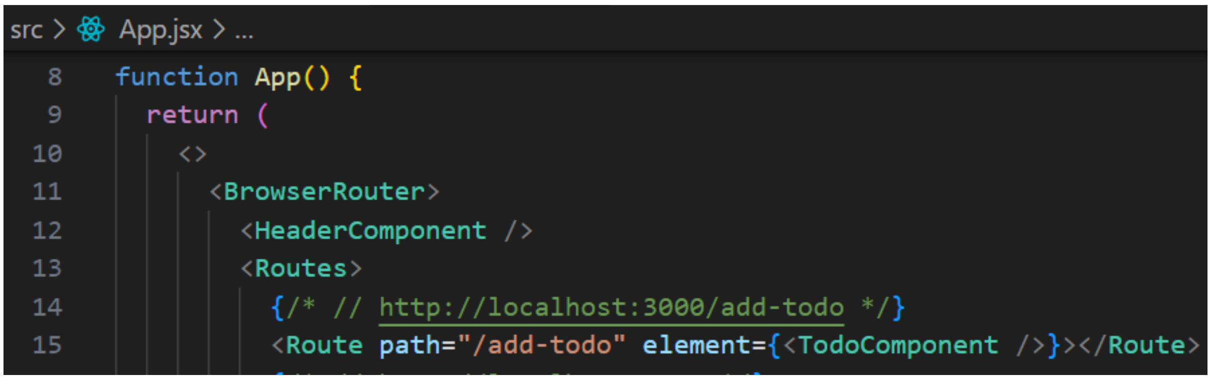

src > App.jsx > App

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

function App() {
  return (
    <>
      <BrowserRouter>
        <HeaderComponent />
        <Routes>
          {/* // http://localhost:3000 */}
          <Route path="/" element={<ListTodoComponent />}></Route>
          {/* // http://localhost:3000/todos */}
          <Route path="/todos" element={<ListTodoComponent />}></Route>
        </Routes>
        <FooterComponent />
      </BrowserRouter>
    </>
  );
}
```

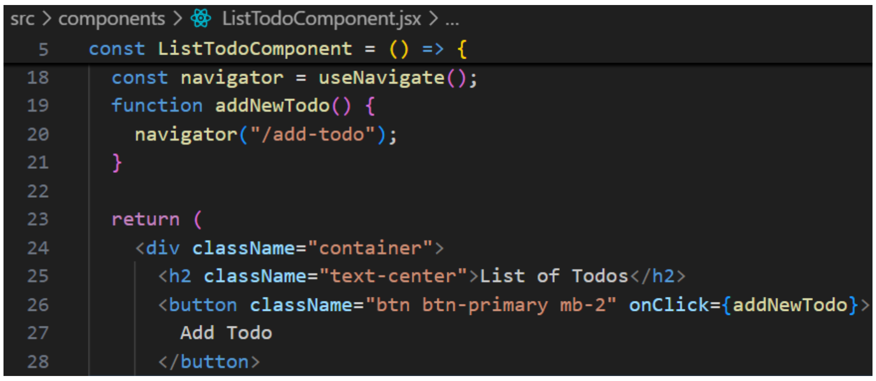
Create React TodoComponent

Development Steps

1. Create React Functional Component - **TodoComponent**
2. Add "**Add Todo**" button in **ListTodoComponent**
3. Configue Route for **TodoComponent**


src > App.jsx > ...

```jsx
function App() {
  return (
    <>
      <BrowserRouter>
        <HeaderComponent />
        <Routes>
          {/* // http://localhost:3000/add-todo */}
          <Route path="/add-todo" element={<TodoComponent />}></Route>
```

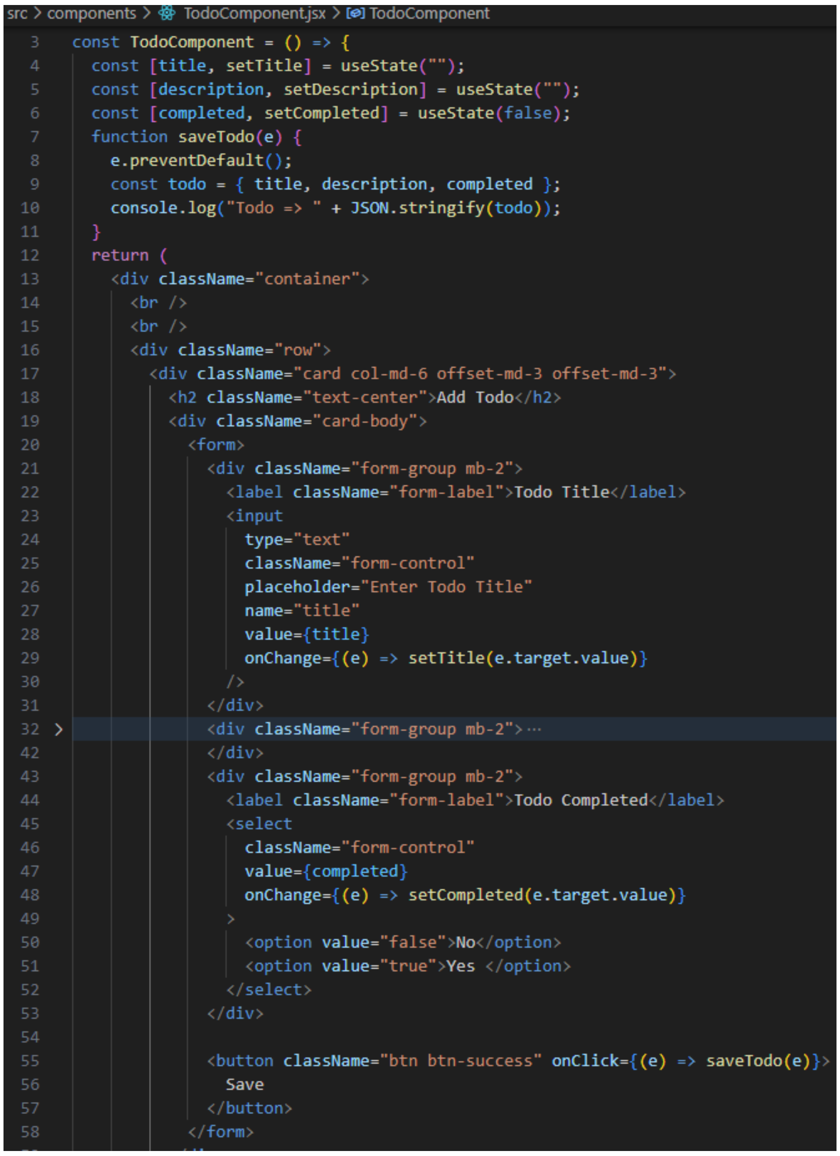


src > components > ListTodoComponent.jsx > ...

```
const ListTodoComponent = () => {
  const navigator = useNavigate();
  function addNewTodo() {
    navigator("/add-todo");
  }

  return (
    <div className="container">
      <h2 className="text-center">List of Todos</h2>
      <button className="btn btn-primary mb-2" onClick={addNewTodo}>
        Add Todo
      </button>
```

Development Steps

1. Define state variables (title, description and completed) in **TodoComponent** using **useState** Hook.
2. Design **Add Todo Form** using HTML and Bootstrap
3. Create JavaScript Function to handle onClick Event (Form submit)



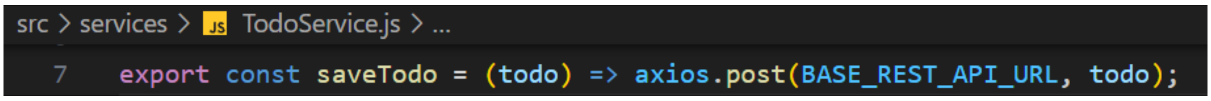
src > components > TodoComponent.jsx > TodoComponent

```
const TodoComponent = () => {
  const [title, setTitle] = useState("");
  const [description, setDescription] = useState("");
  const [completed, setCompleted] = useState(false);
  function saveTodo(e) {
    e.preventDefault();
    const todo = { title, description, completed };
    console.log("Todo => " + JSON.stringify(todo));
  }
  return (
    <div className="container">
      <br />
      <br />
      <div className="row">
        <div className="card col-md-6 offset-md-3 offset-md-3">
          <h2 className="text-center">Add Todo</h2>
          <div className="card-body">
            <form>
              <div className="form-group mb-2">
                <label className="form-label">Todo Title</label>
                <input
                  type="text"
                  className="form-control"
                  placeholder="Enter Todo Title"
                  name="title"
                  value={title}
                  onChange={(e) => setTitle(e.target.value)}
                />
              </div>
              <div className="form-group mb-2">…
              </div>
              <div className="form-group mb-2">
                <label className="form-label">Todo Completed</label>
                <select
                  className="form-control"
                  value={completed}
                  onChange={(e) => setCompleted(e.target.value)}
                >
                  <option value="false">No</option>
                  <option value="true">Yes </option>
                </select>
              </div>

              <button className="btn btn-success" onClick={(e) => saveTodo(e)}>
                Save
              </button>
            </form>
```

Connect React App to Add Todo REST API

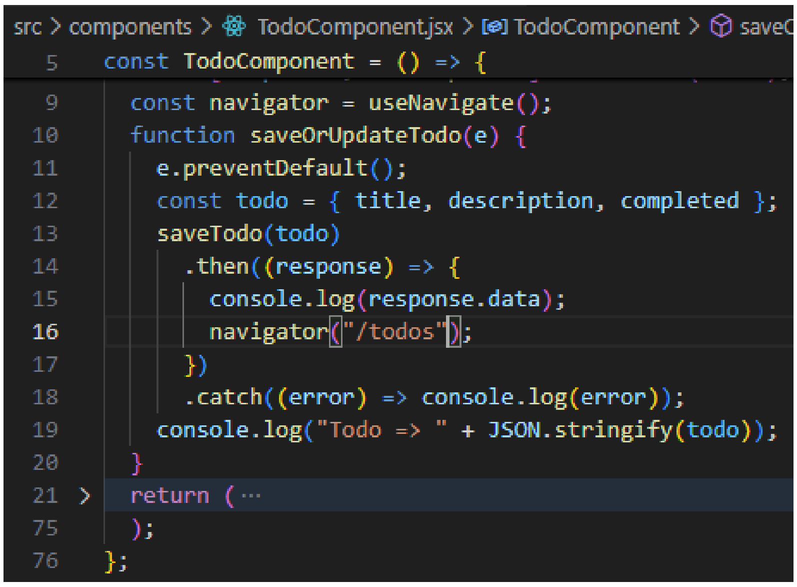
Development Steps

1. In **TodoService**, write a code to call **Add Todo REST API** using axios.
2. Change **TodoComponent** to call **TodoService** method
3. Navigate to List Todos Page After Form Submission Done



src > services > TodoService.js > ...

```
export const saveTodo = (todo) => axios.post(BASE_REST_API_URL, todo);
```



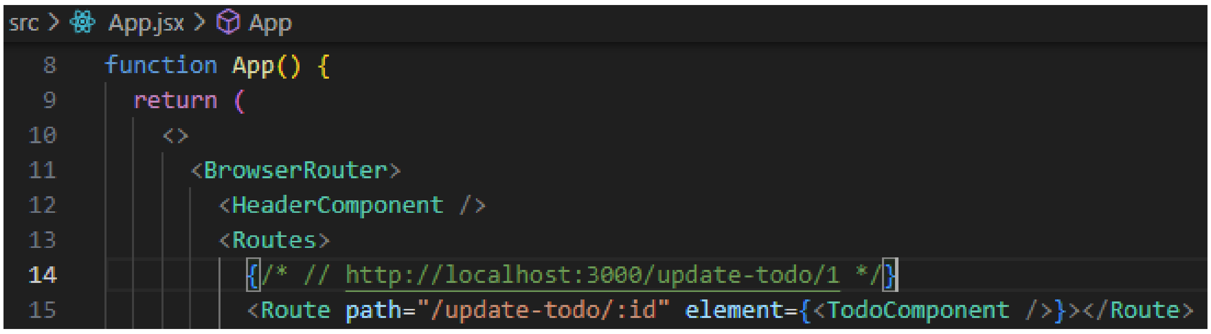
src > components > TodoComponent.jsx > TodoComponent > saveC

```
const TodoComponent = () => {
  const navigator = useNavigate();
  function saveOrUpdateTodo(e) {
    e.preventDefault();
    const todo = { title, description, completed };
    saveTodo(todo)
      .then((response) => {
        console.log(response.data);
        navigator("/todos");
      })
      .catch((error) => console.log(error));
    console.log("Todo => " + JSON.stringify(todo));
  }
  return (…
  );
};
```

Update Todo Feature

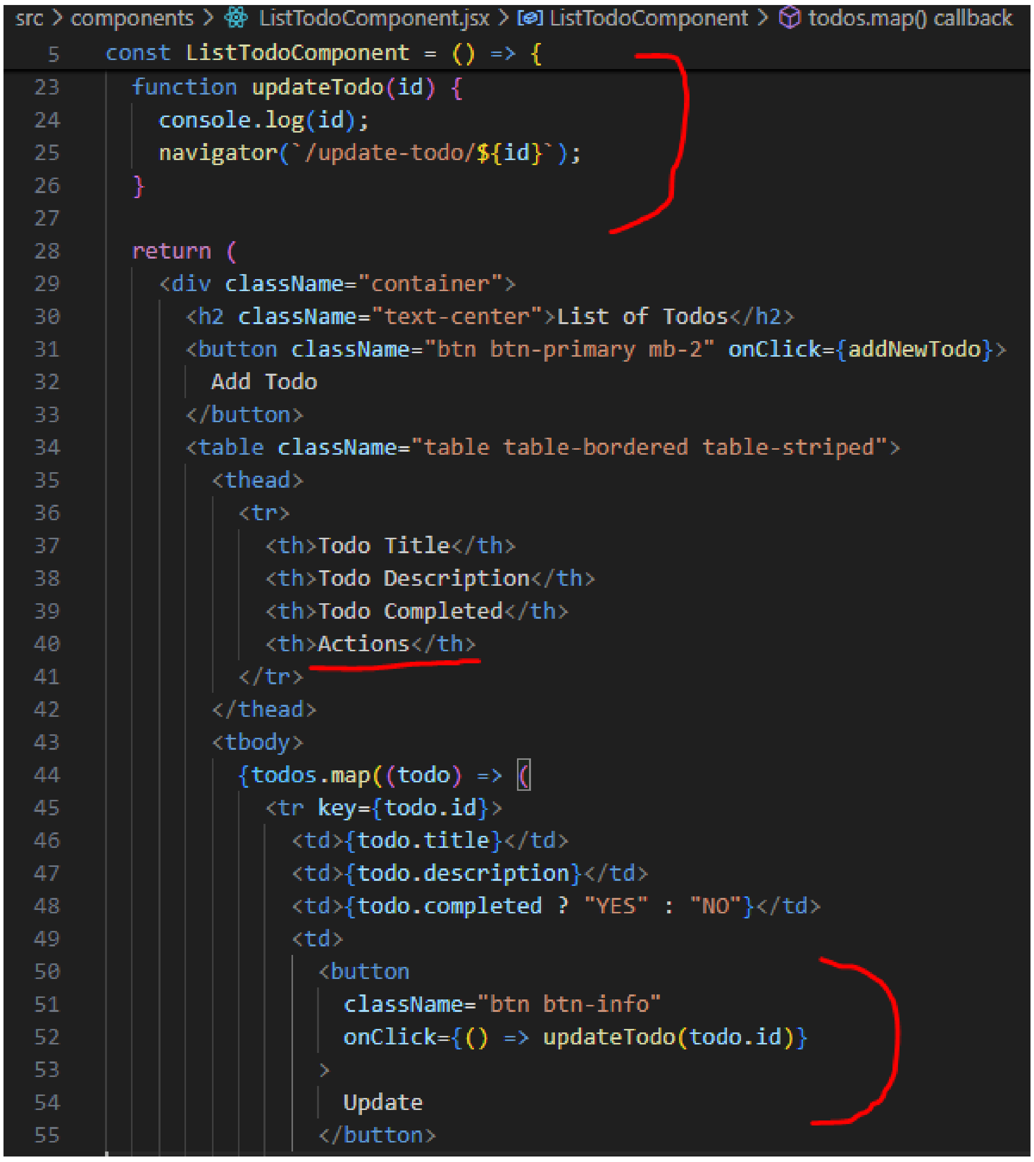
Development Steps

1. Add **Update** button to list todos page
2. Add Route for Update Todo in **App** component
3. Change Page Title Dynamically (**TodoComponent** supports both Add and Update)



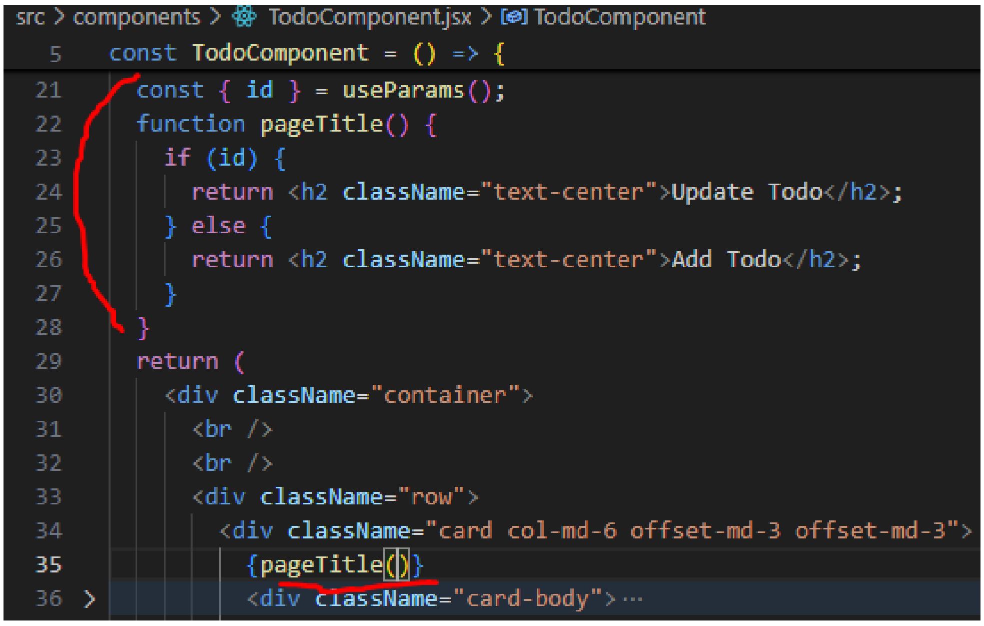
src > App.jsx > App

```
function App() {
  return (
    <>
      <BrowserRouter>
        <HeaderComponent />
        <Routes>
          {/* // http://localhost:3000/update-todo/1 */}
          <Route path="/update-todo/:id" element={<TodoComponent />}></Route>
```



src > components > ListTodoComponent.jsx > ListTodoComponent > todos.map() callback

```
const ListTodoComponent = () => {
  function updateTodo(id) {
    console.log(id);
    navigator(`/update-todo/${id}`);
  }

  return (
    <div className="container">
      <h2 className="text-center">List of Todos</h2>
      <button className="btn btn-primary mb-2" onClick={addNewTodo}>
        Add Todo
      </button>
      <table className="table table-bordered table-striped">
        <thead>
          <tr>
            <th>Todo Title</th>
            <th>Todo Description</th>
            <th>Todo Completed</th>
            <th>Actions</th>
          </tr>
        </thead>
        <tbody>
          {todos.map((todo) => (
            <tr key={todo.id}>
              <td>{todo.title}</td>
              <td>{todo.description}</td>
              <td>{todo.completed ? "YES" : "NO"}</td>
              <td>
                <button
                  className="btn btn-info"
                  onClick={() => updateTodo(todo.id)}
                >
                  Update
                </button>
```



src > components > TodoComponent.jsx > TodoComponent

```
const TodoComponent = () => {
  const { id } = useParams();
  function pageTitle() {
    if (id) {
      return <h2 className="text-center">Update Todo</h2>;
    } else {
      return <h2 className="text-center">Add Todo</h2>;
    }
  }
  return (
    <div className="container">
      <br />
      <br />
      <div className="row">
        <div className="card col-md-6 offset-md-3 offset-md-3">
          {pageTitle()}
          <div className="card-body">…
```

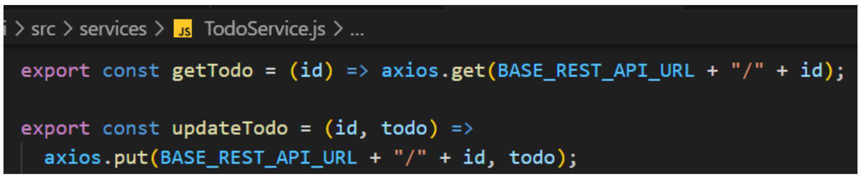
Connect React App to Get Todo & Update Todo REST API

Development Steps

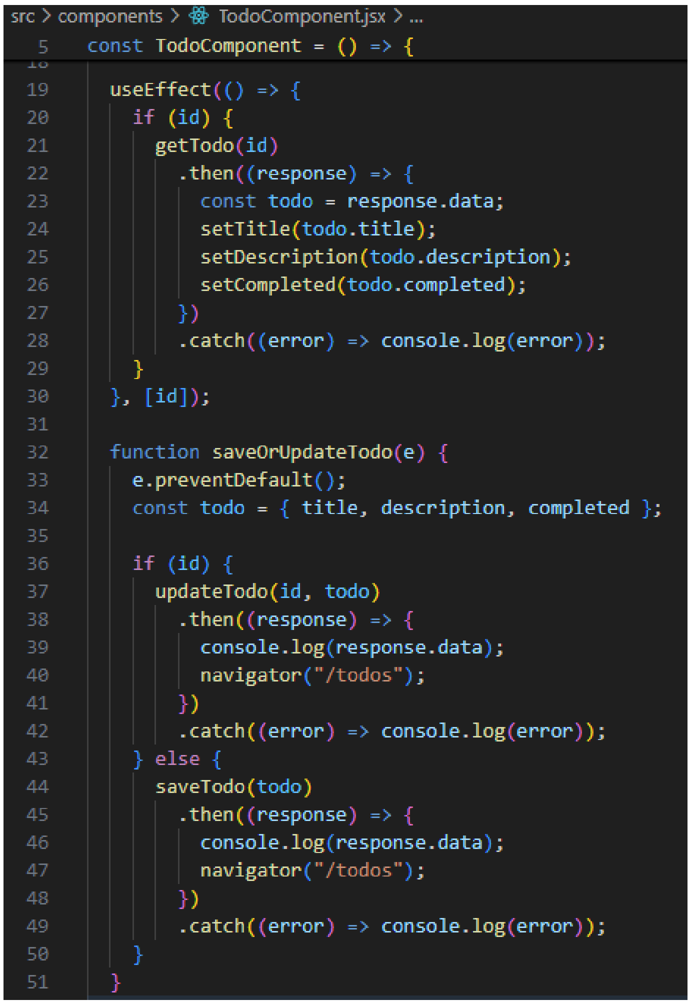
1. In **TodoService**, write a code to call **Get Todo REST API** using axios.
2. Use **useEffect** hook to populate the Todo data in the form for update

Development Steps

1. In **TodoService**, write a code to call **Update Todo REST API** using axios.
2. Change **TodoComponent.saveOrUpdateTodo()** method to perform both add and update todo operations

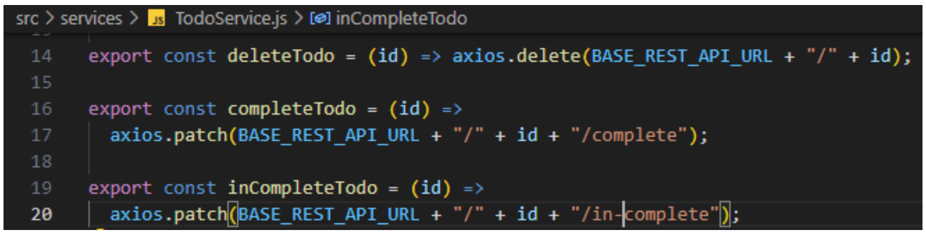


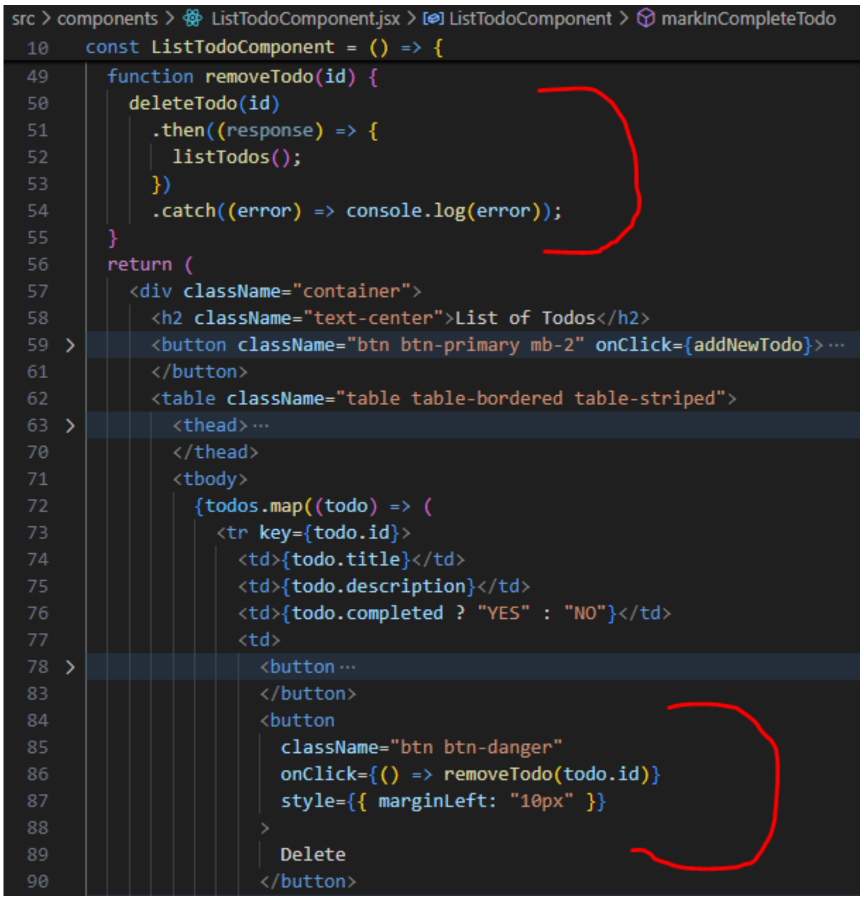
```
> src > services > TodoService.js > ...
export const getTodo = (id) => axios.get(BASE_REST_API_URL + "/" + id);

export const updateTodo = (id, todo) =>
  axios.put(BASE_REST_API_URL + "/" + id, todo);
```



```
src > components > TodoComponent.jsx > ...
const TodoComponent = () => {

  useEffect(() => {
    if (id) {
      getTodo(id)
        .then((response) => {
          const todo = response.data;
          setTitle(todo.title);
          setDescription(todo.description);
          setCompleted(todo.completed);
        })
        .catch((error) => console.log(error));
    }
  }, [id]);

  function saveOrUpdateTodo(e) {
    e.preventDefault();
    const todo = { title, description, completed };

    if (id) {
      updateTodo(id, todo)
        .then((response) => {
          console.log(response.data);
          navigator("/todos");
        })
        .catch((error) => console.log(error));
    } else {
      saveTodo(todo)
        .then((response) => {
          console.log(response.data);
          navigator("/todos");
        })
        .catch((error) => console.log(error));
    }
  }
```

Implement Delete, Complete, Incomplete Todo Feature


```
src > services > TodoService.js > inCompleteTodo
export const deleteTodo = (id) => axios.delete(BASE_REST_API_URL + "/" + id);

export const completeTodo = (id) =>
  axios.patch(BASE_REST_API_URL + "/" + id + "/complete");

export const inCompleteTodo = (id) =>
  axios.patch(BASE_REST_API_URL + "/" + id + "/in-complete");
```


```
src > components > ListTodoComponent.jsx > ListTodoComponent > markInCompleteTodo
const ListTodoComponent = () => {
  function removeTodo(id) {
    deleteTodo(id)
      .then((response) => {
        listTodos();
      })
      .catch((error) => console.log(error));
  }
  return (
    <div className="container">
      <h2 className="text-center">List of Todos</h2>
      <button className="btn btn-primary mb-2" onClick={addNewTodo}>…
      </button>
      <table className="table table-bordered table-striped">
        <thead>…
        </thead>
        <tbody>
          {todos.map((todo) => (
            <tr key={todo.id}>
              <td>{todo.title}</td>
              <td>{todo.description}</td>
              <td>{todo.completed ? "YES" : "NO"}</td>
              <td>
                <button…
                </button>
                <button
                  className="btn btn-danger"
                  onClick={() => removeTodo(todo.id)}
                  style={{ marginLeft: "10px" }}
                >
                  Delete
                </button>
```

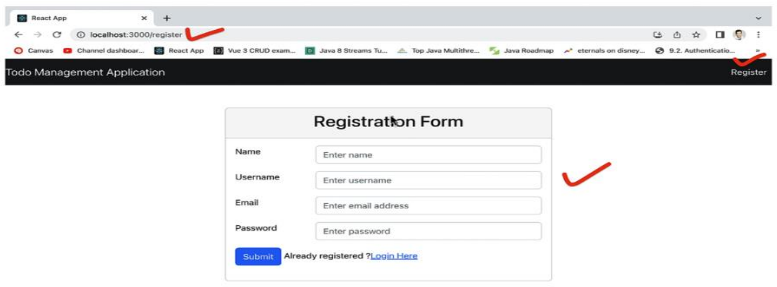
Requirement 5 - Register and Login Implementation in React App

Build Frontend React App (Consume REST APIs) for Registration and Login Module

User should able perform below operations:

- Register to Todo App
- Login to Todo App using Registered Credentials
- Logout from Todo App

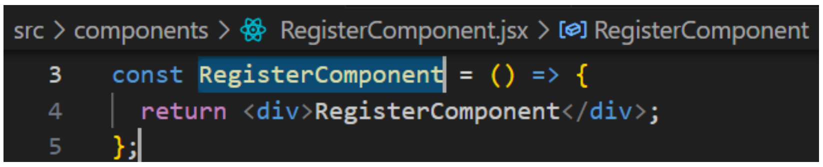
User Registration Feature



Development Steps

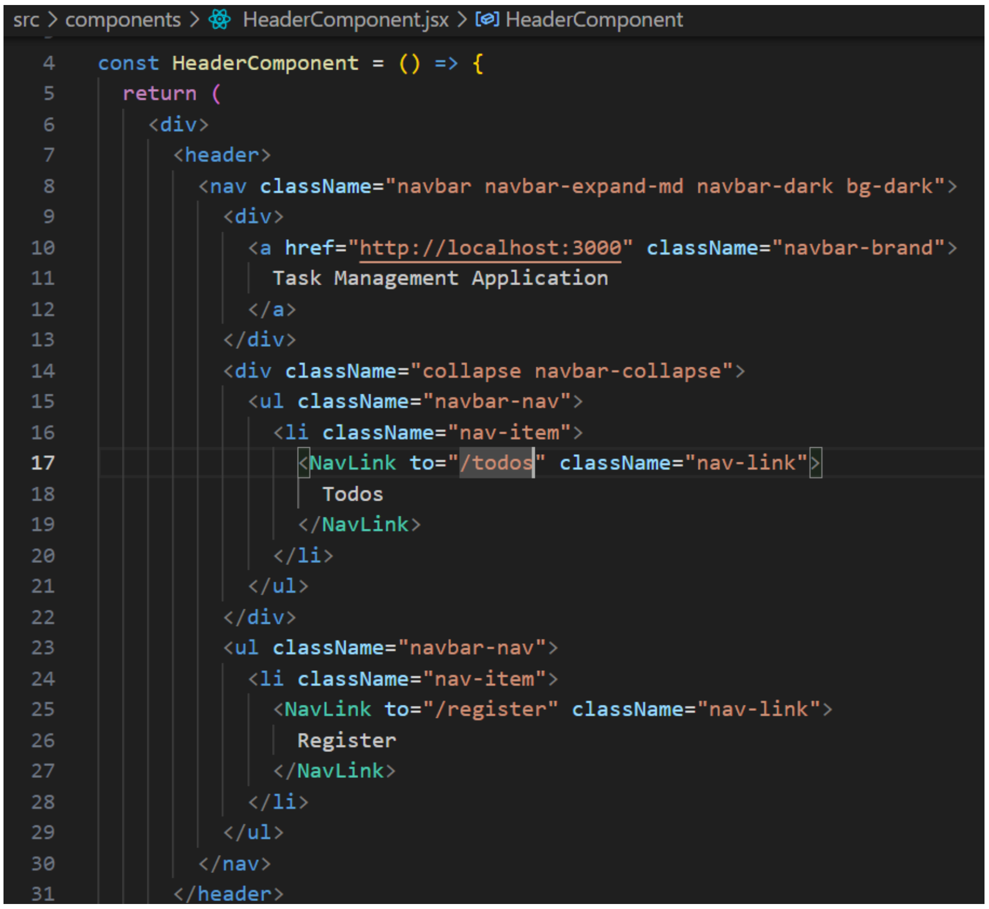
1. Create functional component - **RegisterComponent**
2. Configure Route for **RegisterComponent**
3. Add **Register** Button to Header

Whenever we want to build navigation bar or tabs in header, we can use NavLink component from react-router-dom library. This component help us to select the currently selected tab.



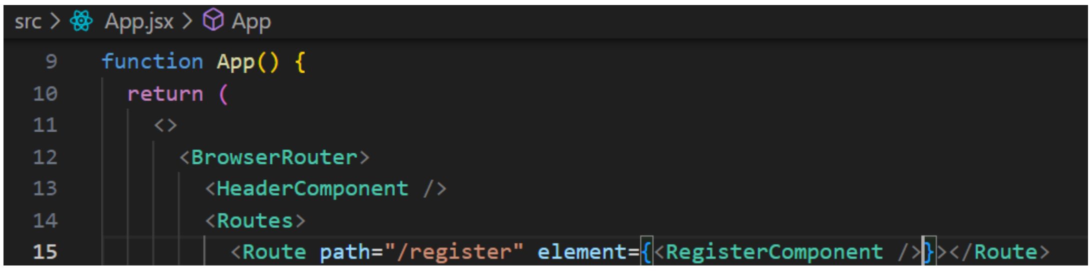
src > components > RegisterComponent.jsx > RegisterComponent

```
const RegisterComponent = () => {
  return <div>RegisterComponent</div>;
};
```



src > components > HeaderComponent.jsx > HeaderComponent

```
const HeaderComponent = () => {
  return (
    <div>
      <header>
        <nav className="navbar navbar-expand-md navbar-dark bg-dark">
          <div>
            <a href="http://localhost:3000" className="navbar-brand">
              Task Management Application
            </a>
          </div>
          <div className="collapse navbar-collapse">
            <ul className="navbar-nav">
              <li className="nav-item">
                <NavLink to="/todos" className="nav-link">
                  Todos
                </NavLink>
              </li>
            </ul>
          </div>
          <ul className="navbar-nav">
            <li className="nav-item">
              <NavLink to="/register" className="nav-link">
                Register
              </NavLink>
            </li>
          </ul>
        </nav>
      </header>
```



```
src > App.jsx > App
function App() {
  return (
    <>
      <BrowserRouter>
        <HeaderComponent />
        <Routes>
          <Route path="/register" element={<RegisterComponent />}></Route>
```

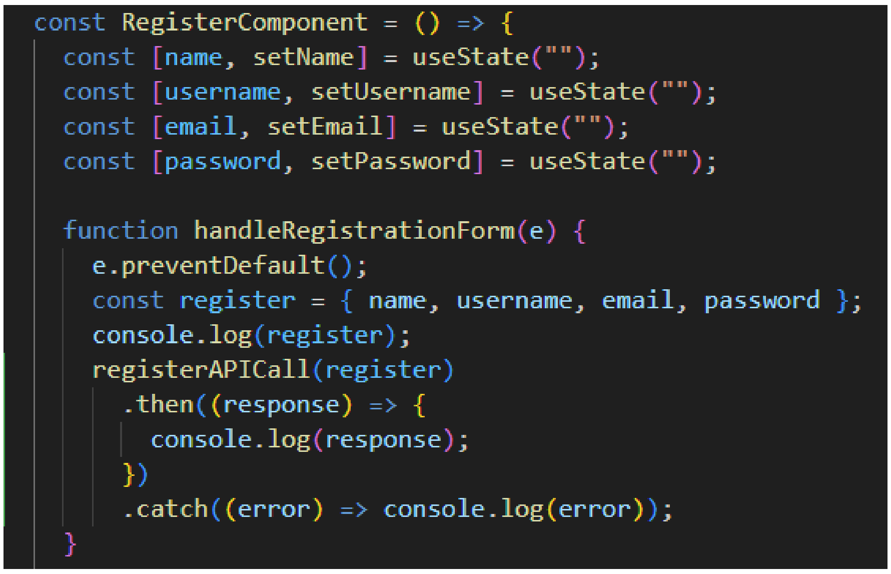
<u>**User Registration Form Handling**</u>

Development Steps

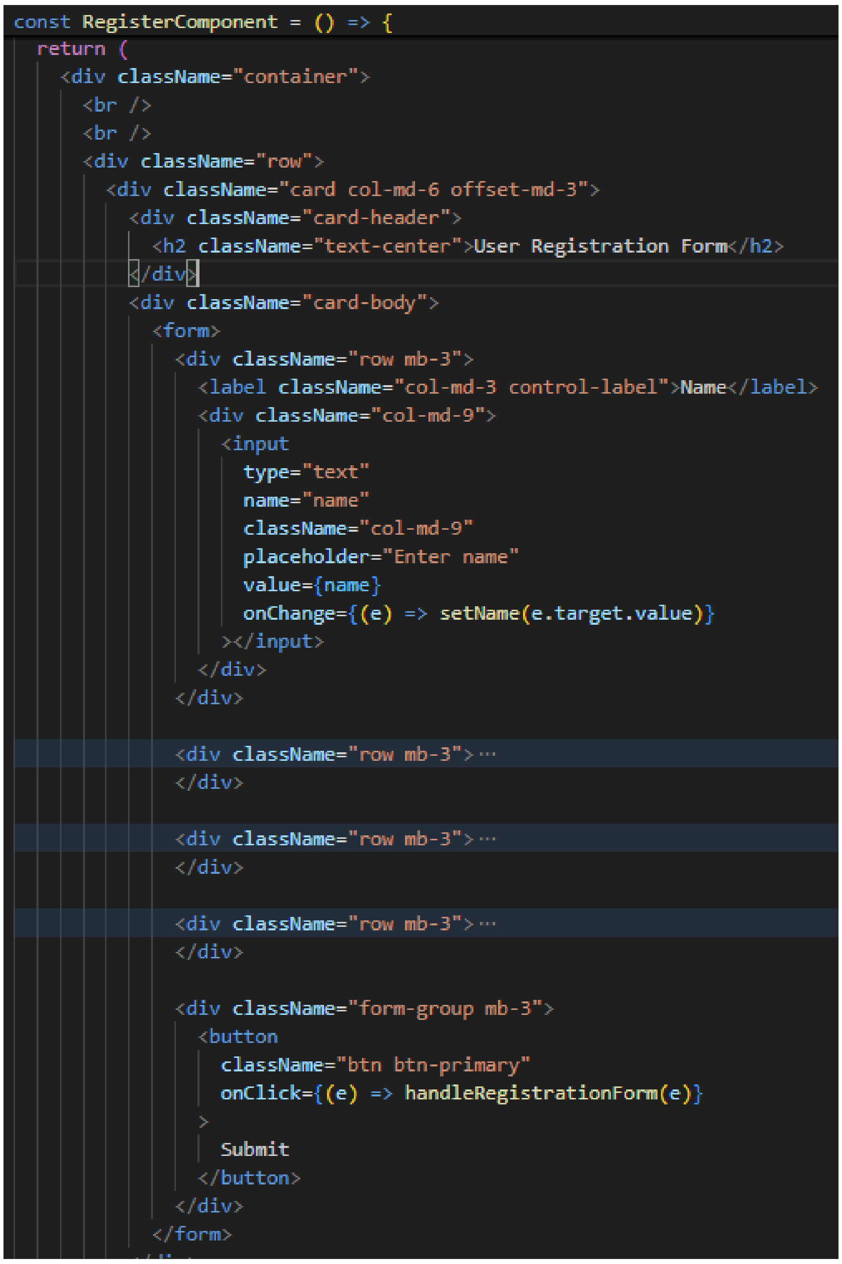
1. Define state variables (**name, username, email and password**) in **RegisterComponent** using **useState** Hook
2. Design **User Registration Form** using HTML and Bootstrap
3. Create JavaScript Function to handle onClick Event (Form submit)

Development Steps

1. Create a **AuthService.js** File
2. In **AuthService**, write a code to call **Register REST API** using **axios**
3. Change **RegisterComponent.handleRegisterForm()** method to call **AuthService** method

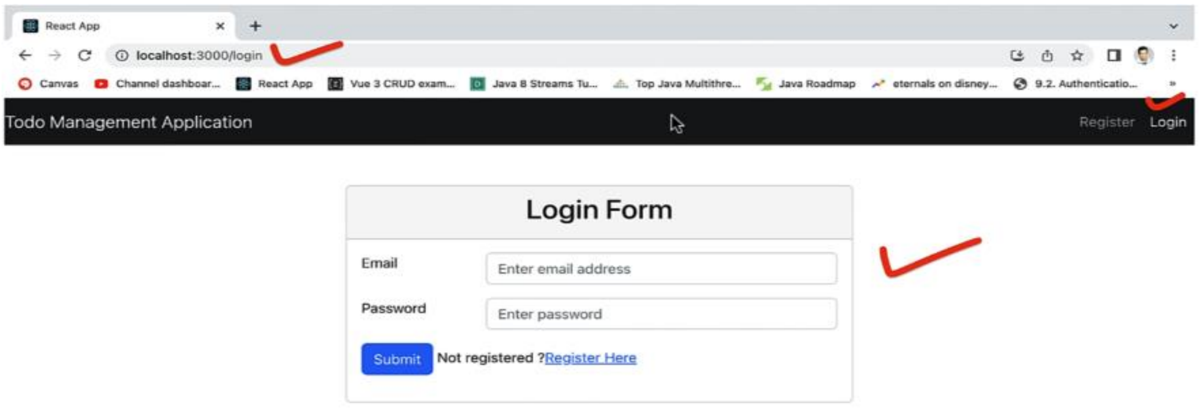


```
const RegisterComponent = () => {
  const [name, setName] = useState("");
  const [username, setUsername] = useState("");
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");

  function handleRegistrationForm(e) {
    e.preventDefault();
    const register = { name, username, email, password };
    console.log(register);
    registerAPICall(register)
      .then((response) => {
        console.log(response);
      })
      .catch((error) => console.log(error));
  }
```



```
const RegisterComponent = () => {
  return (
    <div className="container">
      <br />
      <br />
      <div className="row">
        <div className="card col-md-6 offset-md-3">
          <div className="card-header">
            <h2 className="text-center">User Registration Form</h2>
          </div>
          <div className="card-body">
            <form>
              <div className="row mb-3">
                <label className="col-md-3 control-label">Name</label>
                <div className="col-md-9">
                  <input
                    type="text"
                    name="name"
                    className="col-md-9"
                    placeholder="Enter name"
                    value={name}
                    onChange={(e) => setName(e.target.value)}
                  ></input>
                </div>
              </div>

              <div className="row mb-3">⋯
              </div>

              <div className="row mb-3">⋯
              </div>

              <div className="row mb-3">⋯
              </div>

              <div className="form-group mb-3">
                <button
                  className="btn btn-primary"
                  onClick={(e) => handleRegistrationForm(e)}
                >
                  Submit
                </button>
              </div>
            </form>
```

User Login Feature



Login Form

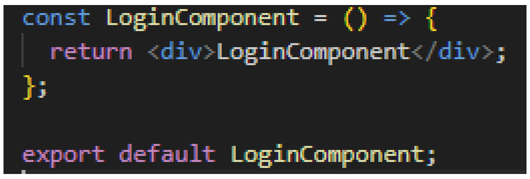
Email [Enter email address]

Password [Enter password]

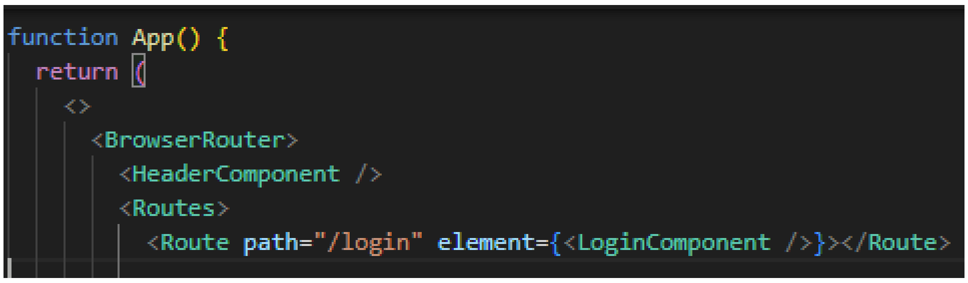
Submit Not registered ?Register Here

Development Steps

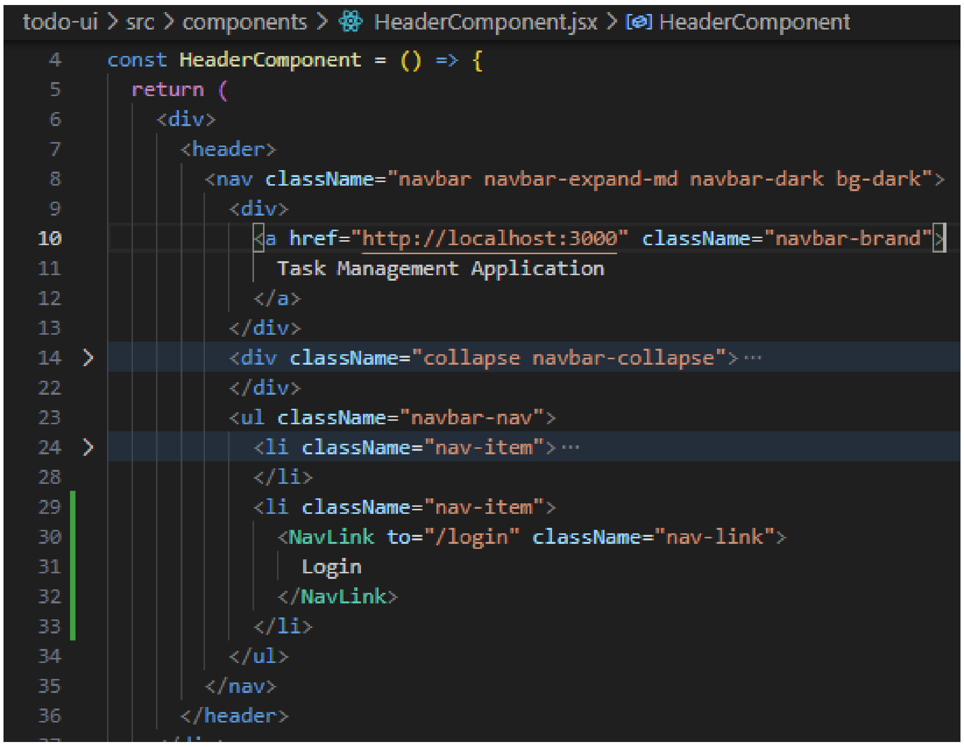
1. Create functional component - **LoginComponent**
2. Configure Route for **LoginComponent**
3. Add **Login** Button to Header



```
const LoginComponent = () => {
  return <div>LoginComponent</div>;
};

export default LoginComponent;
```



```
function App() {
  return (
    <>
      <BrowserRouter>
        <HeaderComponent />
        <Routes>
          <Route path="/login" element={<LoginComponent />}></Route>
```



todo-ui > src > components > HeaderComponent.jsx > HeaderComponent

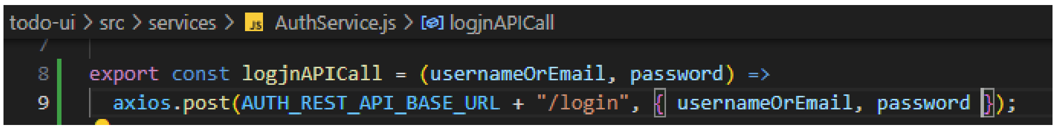
```
const HeaderComponent = () => {
  return (
    <div>
      <header>
        <nav className="navbar navbar-expand-md navbar-dark bg-dark">
          <div>
            <a href="http://localhost:3000" className="navbar-brand">
              Task Management Application
            </a>
          </div>
          <div className="collapse navbar-collapse">···
          </div>
          <ul className="navbar-nav">
            <li className="nav-item">···
            </li>
            <li className="nav-item">
              <NavLink to="/login" className="nav-link">
                Login
              </NavLink>
            </li>
          </ul>
        </nav>
      </header>
```

Development Steps

1. Define state variables (username and password) in **LoginComponent** using **useState** Hook
2. Design User **Login Form** using HTML and Bootstrap
3. Create JavaScript Function to handle onClick Event (Form submit)

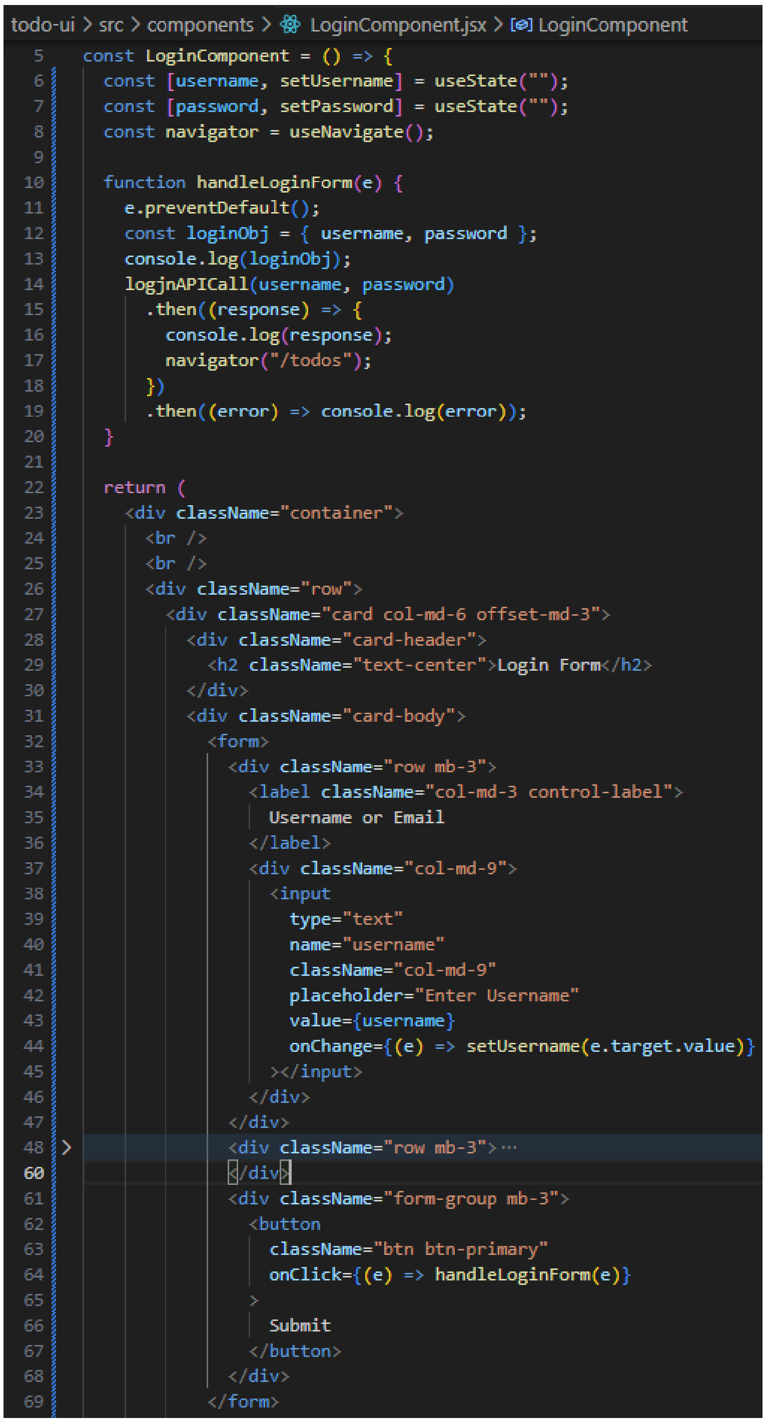
Development Steps

1. In **AuthService**, write a code to call **Login REST API** using axios
2. Change **LoginComponent.handleLoginForm()** method to call **AuthService** method
3. Navigate to **List of Todos** Page After Login Form Submission Done



todo-ui > src > services > AuthService.js > logjnAPICall

```
export const logjnAPICall = (usernameOrEmail, password) =>
  axios.post(AUTH_REST_API_BASE_URL + "/login", { usernameOrEmail, password });
```

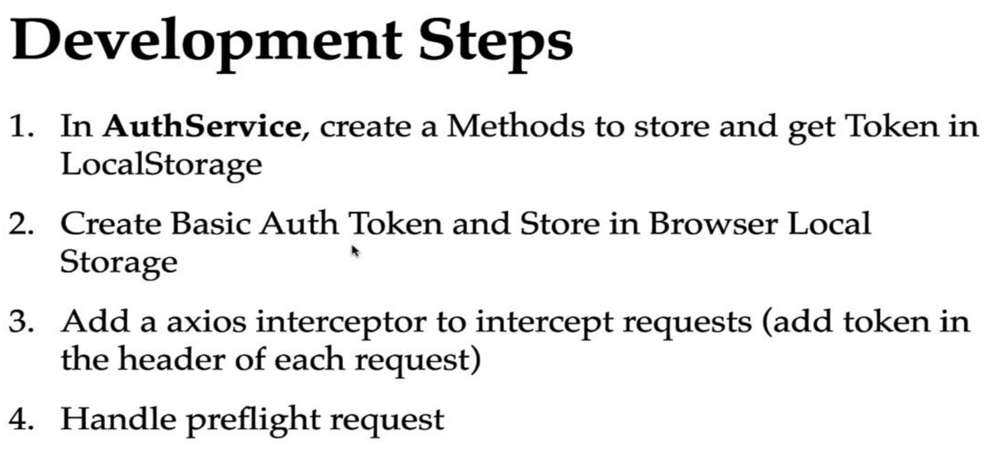


todo-ui > src > components > LoginComponent.jsx > LoginComponent

```
const LoginComponent = () => {
  const [username, setUsername] = useState("");
  const [password, setPassword] = useState("");
  const navigator = useNavigate();

  function handleLoginForm(e) {
    e.preventDefault();
    const loginObj = { username, password };
    console.log(loginObj);
    logjnAPICall(username, password)
      .then((response) => {
        console.log(response);
        navigator("/todos");
      })
      .then((error) => console.log(error));
  }

  return (
    <div className="container">
      <br />
      <br />
      <div className="row">
        <div className="card col-md-6 offset-md-3">
          <div className="card-header">
            <h2 className="text-center">Login Form</h2>
          </div>
          <div className="card-body">
            <form>
              <div className="row mb-3">
                <label className="col-md-3 control-label">
                  Username or Email
                </label>
                <div className="col-md-9">
                  <input
                    type="text"
                    name="username"
                    className="col-md-9"
                    placeholder="Enter Username"
                    value={username}
                    onChange={(e) => setUsername(e.target.value)}
                  ></input>
                </div>
              </div>
              <div className="row mb-3">⋯
              </div>
              <div className="form-group mb-3">
                <button
                  className="btn btn-primary"
                  onClick={(e) => handleLoginForm(e)}
                >
                  Submit
                </button>
              </div>
            </form>
```

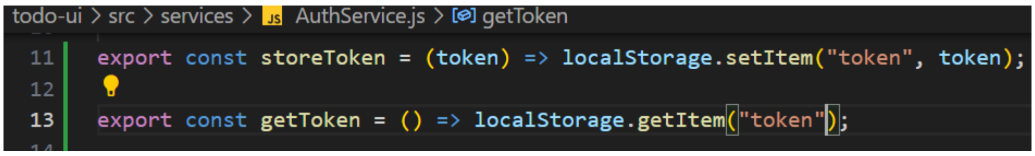
Implement Baisc Authentication in React App

Basically each request should have authorization header having username and password



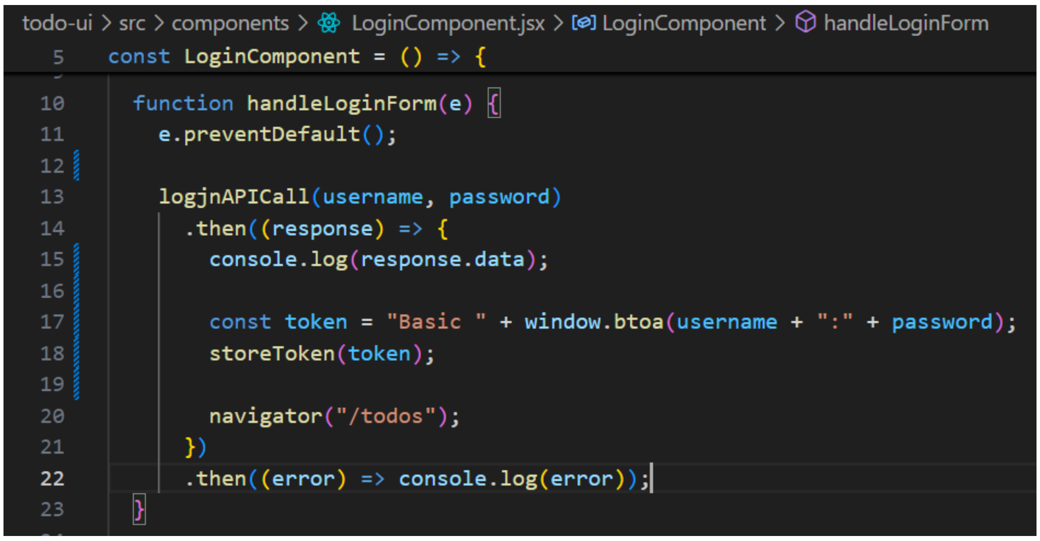
Development Steps

1. In **AuthService**, create a Methods to store and get Token in LocalStorage
2. Create Basic Auth Token and Store in Browser Local Storage
3. Add a axios interceptor to intercept requests (add token in the header of each request)
4. Handle preflight request



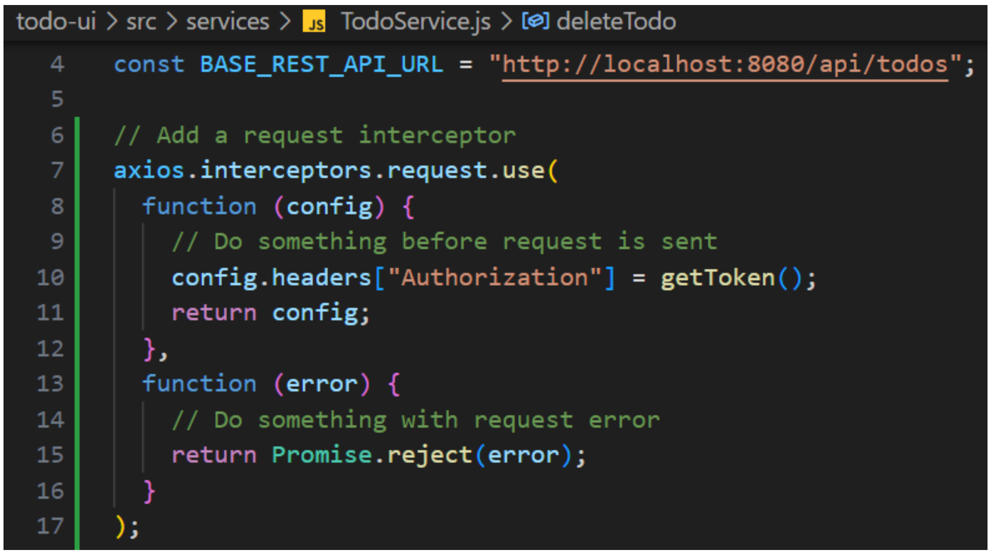
todo-ui > src > services > AuthService.js > getToken

```
export const storeToken = (token) => localStorage.setItem("token", token);

export const getToken = () => localStorage.getItem("token");
```



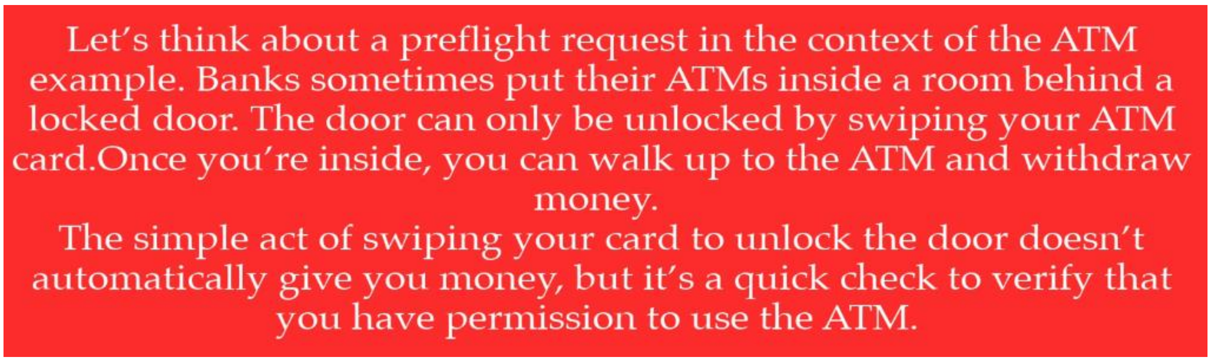
todo-ui > src > components > LoginComponent.jsx > LoginComponent > handleLoginForm

```
const LoginComponent = () => {

  function handleLoginForm(e) {
    e.preventDefault();

    logjnAPICall(username, password)
      .then((response) => {
        console.log(response.data);

        const token = "Basic " + window.btoa(username + ":" + password);
        storeToken(token);

        navigator("/todos");
      })
      .then((error) => console.log(error));
  }
```

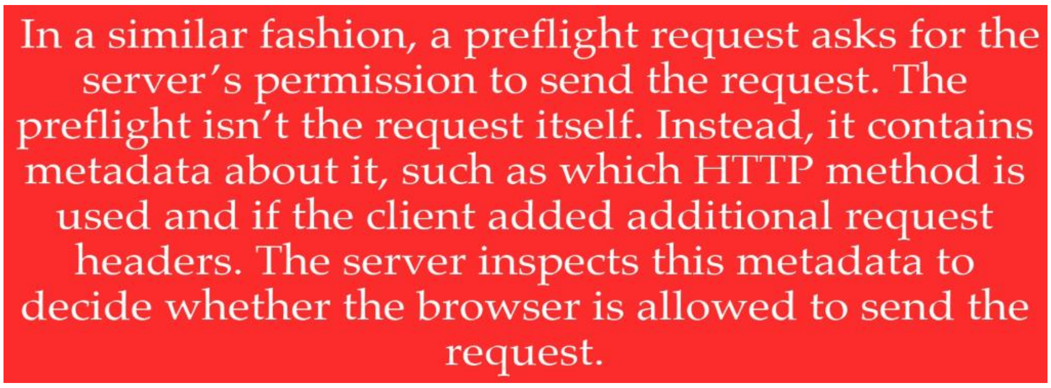


todo-ui > src > services > TodoService.js > deleteTodo

```
const BASE_REST_API_URL = "http://localhost:8080/api/todos";

// Add a request interceptor
axios.interceptors.request.use(
  function (config) {
    // Do something before request is sent
    config.headers["Authorization"] = getToken();
    return config;
  },
  function (error) {
    // Do something with request error
    return Promise.reject(error);
  }
);
```

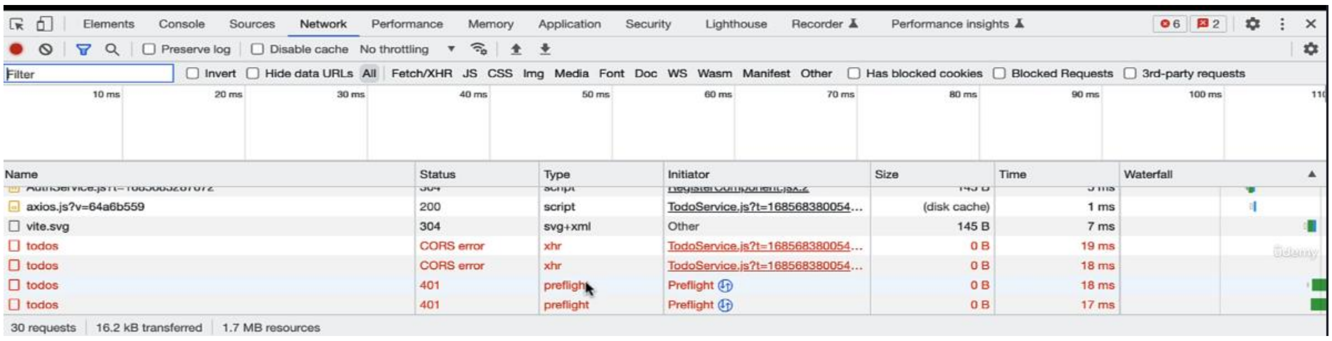


Let's think about a preflight request in the context of the ATM example. Banks sometimes put their ATMs inside a room behind a locked door. The door can only be unlocked by swiping your ATM card.Once you're inside, you can walk up to the ATM and withdraw money.
The simple act of swiping your card to unlock the door doesn't automatically give you money, but it's a quick check to verify that you have permission to use the ATM.



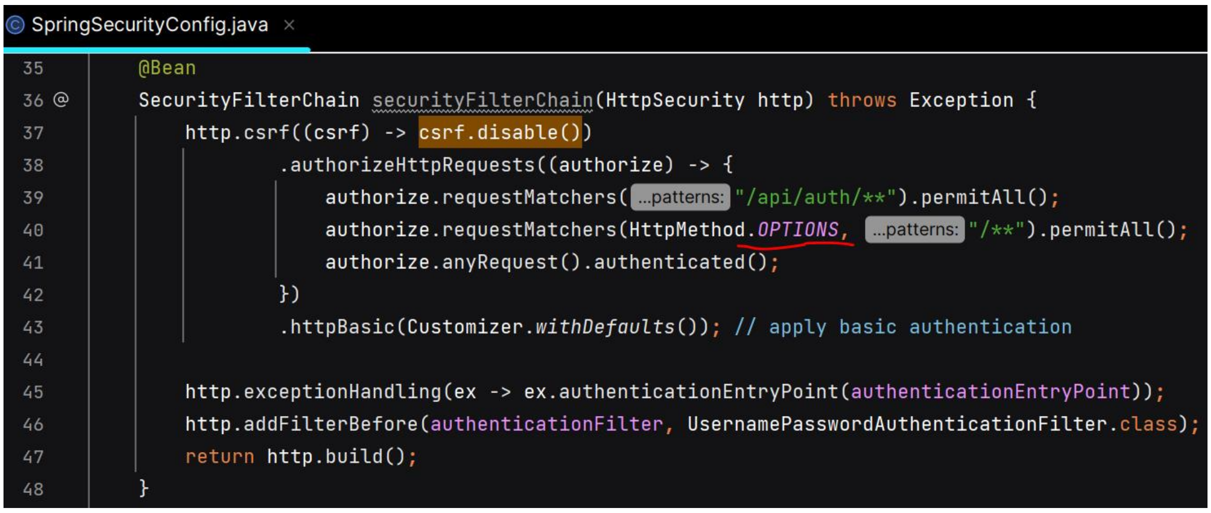
In a similar fashion, a preflight request asks for the server's permission to send the request. The preflight isn't the request itself. Instead, it contains metadata about it, such as which HTTP method is used and if the client added additional request headers. The server inspects this metadata to decide whether the browser is allowed to send the request.

Inorder to handle preflight request, we need to add logic in Spring security to give permission to preflight requests. (to all OPTIONS request)



Name	Status	Type	Initiator	Size	Time	Waterfall
axios.js?v=64a6b559	200	script	TodoService.js?t=168568380054...	(disk cache)	1 ms	
vite.svg	304	svg+xml	Other	145 B	7 ms	
todos	CORS error	xhr	TodoService.js?t=168568380054...	0 B	19 ms	
todos	CORS error	xhr	TodoService.js?t=168568380054...	0 B	18 ms	
todos	401	preflight	Preflight	0 B	18 ms	
todos	401	preflight	Preflight	0 B	17 ms	

30 requests | 16.2 kB transferred | 1.7 MB resources



SpringSecurityConfig.java

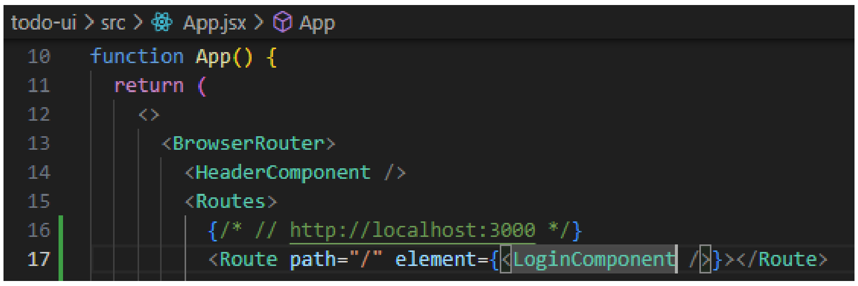
```java
    @Bean
    SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http.csrf((csrf) -> csrf.disable())
                .authorizeHttpRequests((authorize) -> {
                    authorize.requestMatchers(...patterns: "/api/auth/**").permitAll();
                    authorize.requestMatchers(HttpMethod.OPTIONS, ...patterns: "/**").permitAll();
                    authorize.anyRequest().authenticated();
                })
                .httpBasic(Customizer.withDefaults()); // apply basic authentication

        http.exceptionHandling(ex -> ex.authenticationEntryPoint(authenticationEntryPoint));
        http.addFilterBefore(authenticationFilter, UsernamePasswordAuthenticationFilter.class);
        return http.build();
    }
```

Display the Links in Header of application as Per User Authentication

Development Steps

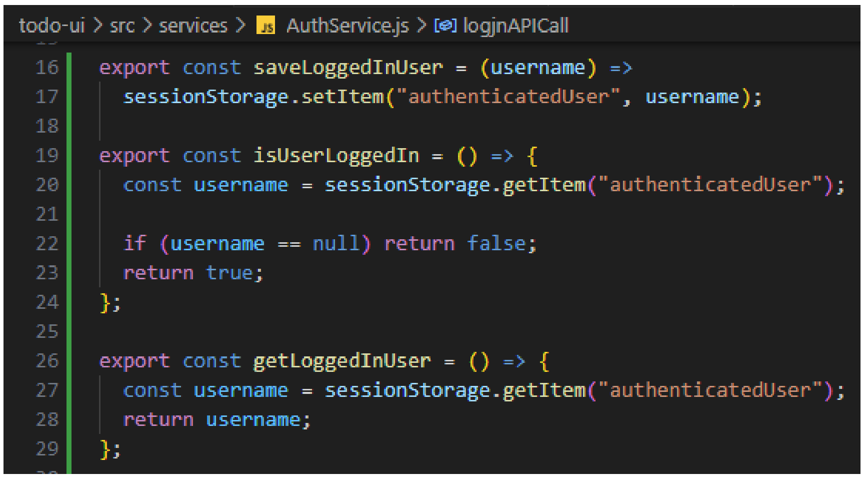
1. Create **saveLoggedInUser(), isUserLoggedIn(), getLoggedInUser()** methods
2. Change the Header to Display Proper Links

Local storage doesn't have expiration date but session storage do have expiration. Whenever we close the browser tab then all the data in session storage gets lost.



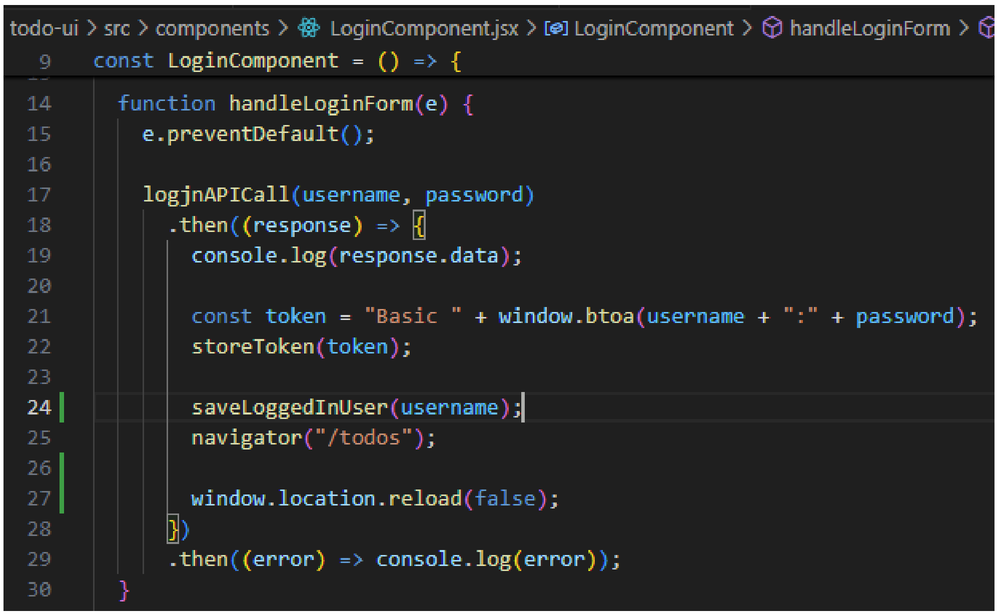
todo-ui > src > App.jsx > App

```
function App() {
  return (
    <>
      <BrowserRouter>
        <HeaderComponent />
        <Routes>
          {/* // http://localhost:3000 */}
          <Route path="/" element={<LoginComponent />}></Route>
```



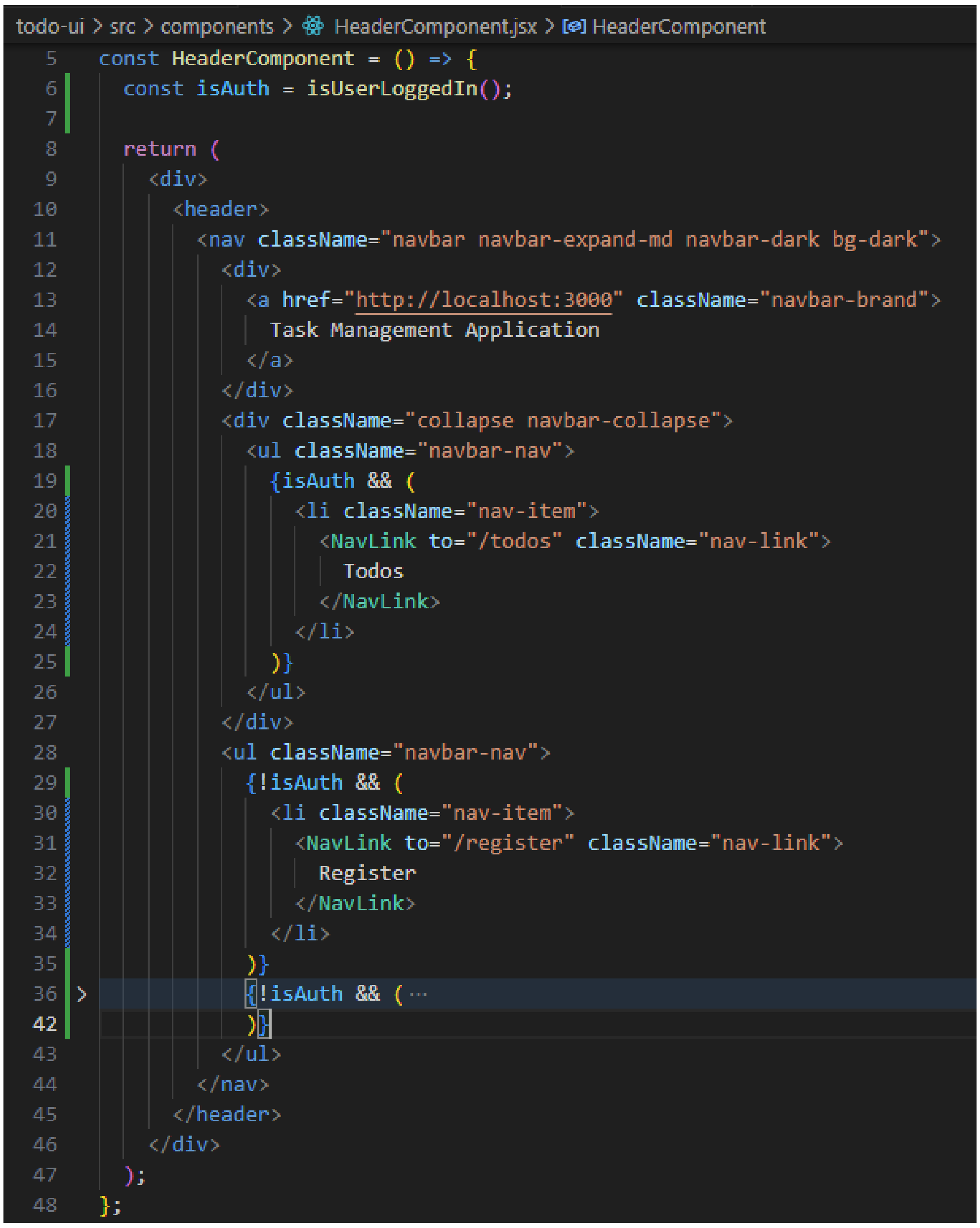
todo-ui > src > services > AuthService.js > logjnAPICall

```
export const saveLoggedInUser = (username) =>
  sessionStorage.setItem("authenticatedUser", username);

export const isUserLoggedIn = () => {
  const username = sessionStorage.getItem("authenticatedUser");

  if (username == null) return false;
  return true;
};

export const getLoggedInUser = () => {
  const username = sessionStorage.getItem("authenticatedUser");
  return username;
};
```



todo-ui > src > components > LoginComponent.jsx > LoginComponent > handleLoginForm >

```
const LoginComponent = () => {

  function handleLoginForm(e) {
    e.preventDefault();

    logjnAPICall(username, password)
      .then((response) => {
        console.log(response.data);

        const token = "Basic " + window.btoa(username + ":" + password);
        storeToken(token);

        saveLoggedInUser(username);
        navigator("/todos");

        window.location.reload(false);
      })
      .then((error) => console.log(error));
  }
```

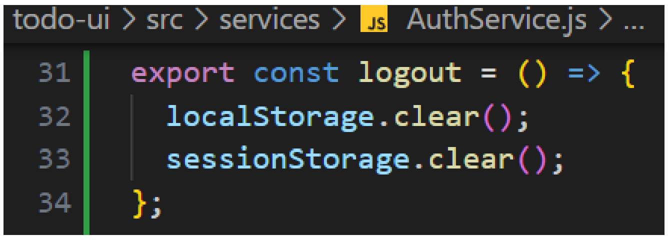

todo-ui > src > components > HeaderComponent.jsx > HeaderComponent

```
const HeaderComponent = () => {
  const isAuth = isUserLoggedIn();

  return (
    <div>
      <header>
        <nav className="navbar navbar-expand-md navbar-dark bg-dark">
          <div>
            <a href="http://localhost:3000" className="navbar-brand">
              Task Management Application
            </a>
          </div>
          <div className="collapse navbar-collapse">
            <ul className="navbar-nav">
              {isAuth && (
                <li className="nav-item">
                  <NavLink to="/todos" className="nav-link">
                    Todos
                  </NavLink>
                </li>
              )}
            </ul>
          </div>
          <ul className="navbar-nav">
            {!isAuth && (
              <li className="nav-item">
                <NavLink to="/register" className="nav-link">
                  Register
                </NavLink>
              </li>
            )}
            {!isAuth && (…
            )}
          </ul>
        </nav>
      </header>
    </div>
  );
};
```

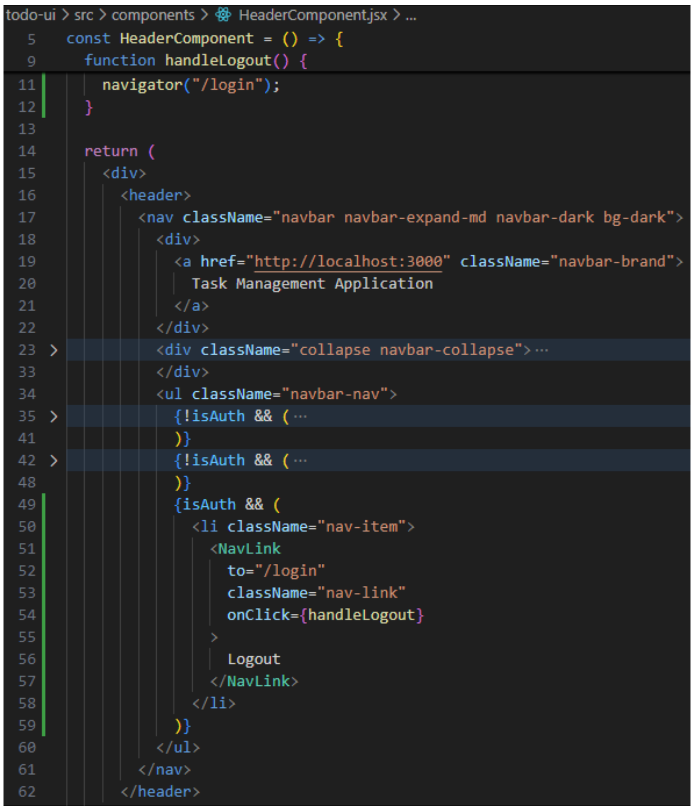
Logout Feature

Development Steps

1. Create **logout() method to clear localStorage and sessionStorage**
2. Add Logout link or button in the header


todo-ui > src > services > AuthService.js > ...

```
export const logout = () => {
  localStorage.clear();
  sessionStorage.clear();
};
```



```
todo-ui > src > components > HeaderComponent.jsx > ...
const HeaderComponent = () => {
  function handleLogout() {
    navigator("/login");
  }

  return (
    <div>
      <header>
        <nav className="navbar navbar-expand-md navbar-dark bg-dark">
          <div>
            <a href="http://localhost:3000" className="navbar-brand">
              Task Management Application
            </a>
          </div>
          <div className="collapse navbar-collapse">···
          </div>
          <ul className="navbar-nav">
            {!isAuth && (···
            )}
            {!isAuth && (···
            )}
            {isAuth && (
              <li className="nav-item">
                <NavLink
                  to="/login"
                  className="nav-link"
                  onClick={handleLogout}
                >
                  Logout
                </NavLink>
              </li>
            )}
          </ul>
        </nav>
      </header>
```

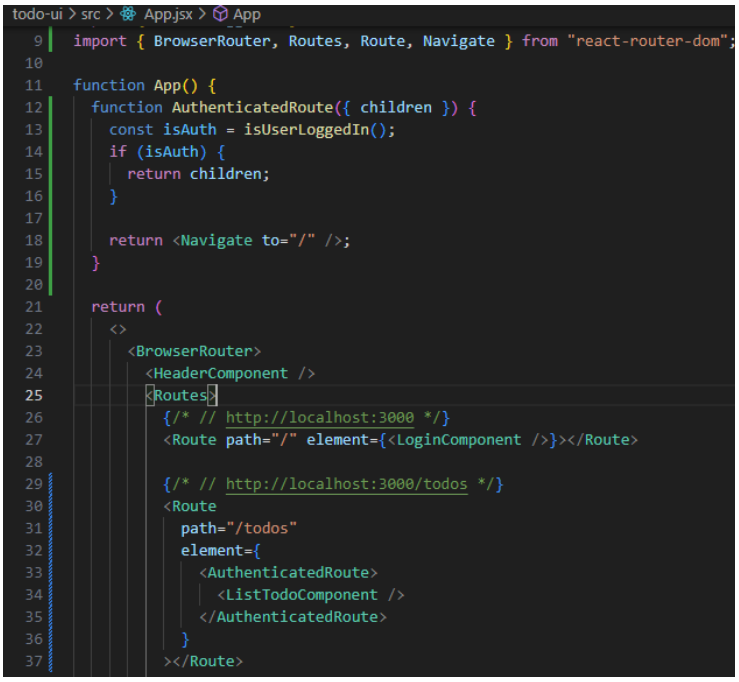
Secure the Routes

Like /todos, /add-todo, /update-todo these pages should be accessible to only loggedIn users

Development Steps

1. Create **AuthenticatedRoute** functional component
2. Use **AuthenticatedRoute** to secure the Routes (/todos, /add-todo, and /update-todo/:id)

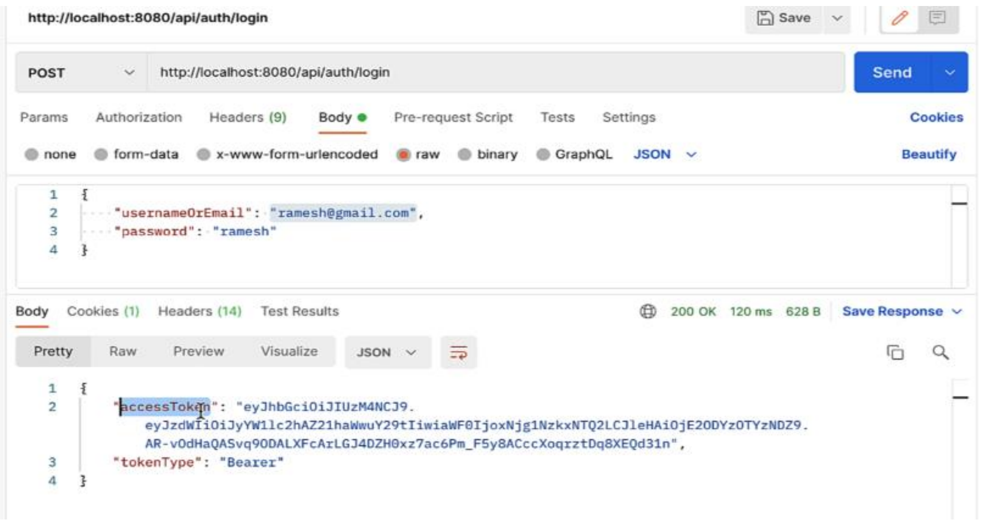
Navigate is a component that allows you to **programmatically redirect** the user to a different route.



```
todo-ui > src > App.jsx > App
import { BrowserRouter, Routes, Route, Navigate } from "react-router-dom";

function App() {
  function AuthenticatedRoute({ children }) {
    const isAuth = isUserLoggedIn();
    if (isAuth) {
      return children;
    }

    return <Navigate to="/" />;
  }

  return (
    <>
      <BrowserRouter>
        <HeaderComponent />
        <Routes>
          {/* // http://localhost:3000 */}
          <Route path="/" element={<LoginComponent />}></Route>

          {/* // http://localhost:3000/todos */}
          <Route
            path="/todos"
            element={
              <AuthenticatedRoute>
                <ListTodoComponent />
              </AuthenticatedRoute>
            }
          ></Route>
```

Using JWT Token in React App

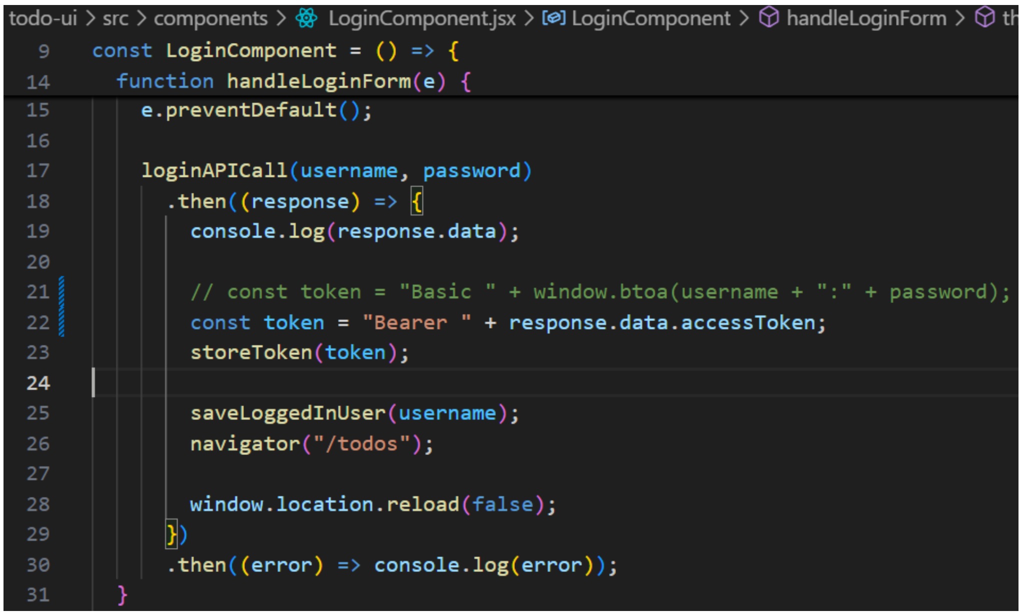
Instead of basic auth token we'll send jwt token in header with every request. We'll get JWT token in response of login api ("accessToken" attribute)



POST http://localhost:8080/api/auth/login

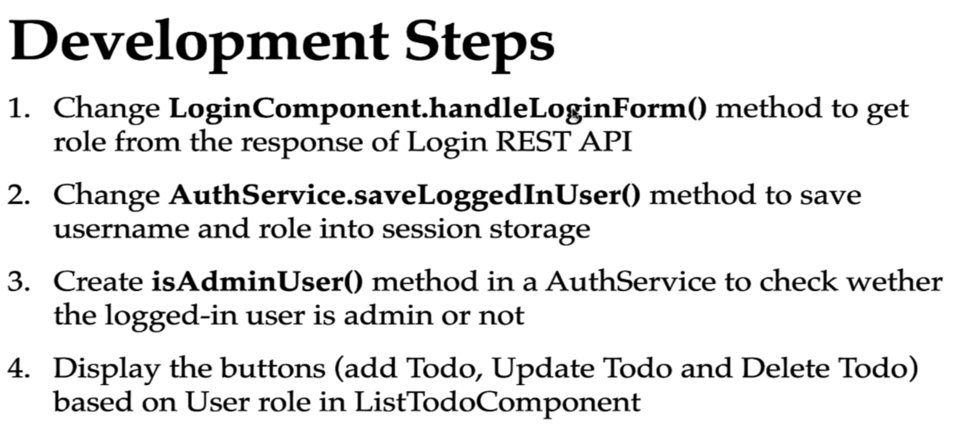
```json
{
    "usernameOrEmail": "ramesh@gmail.com",
    "password": "ramesh"
}
```

```json
{
    "accessToken": "eyJhbGciOiJIUzM4NCJ9.
      eyJzdWIiOiJyYW1lc2hAZ21haWwuY29tIiwiaWF0IjoxNjg1NzkxNTQ2LCJleHAiOjE2ODYzOTYzNDZ9.
      AR-vOdHaQASvq9ODALXFcArLGJ4DZH0xz7ac6Pm_F5y8ACccXoqrztDq8XEQd31n",
    "tokenType": "Bearer"
}
```


todo-ui > src > components > LoginComponent.jsx > LoginComponent > handleLoginForm > th

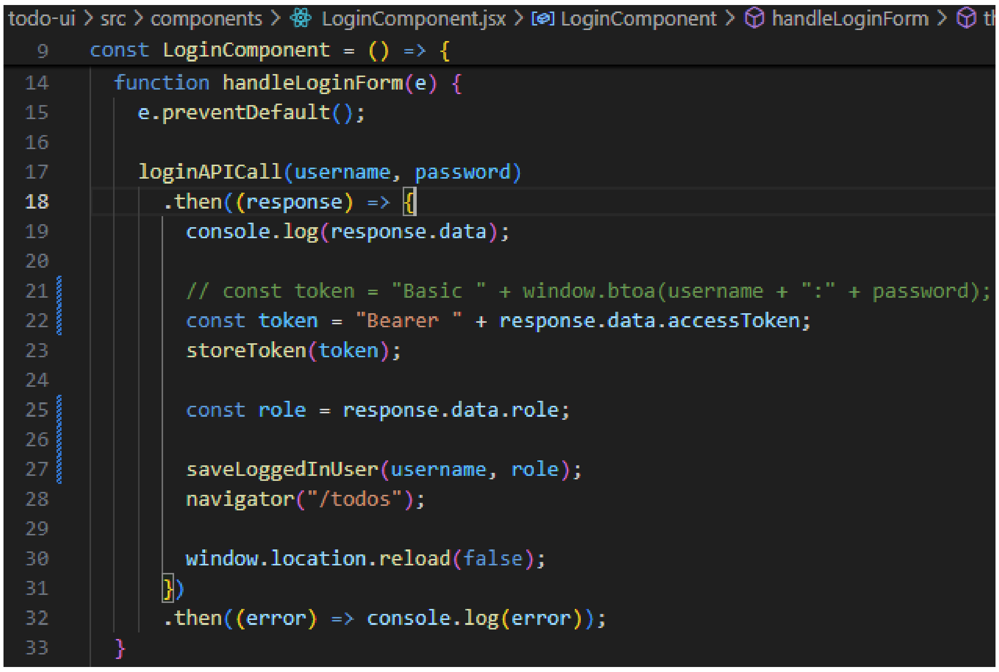
```jsx
const LoginComponent = () => {
  function handleLoginForm(e) {
    e.preventDefault();

    loginAPICall(username, password)
      .then((response) => {
        console.log(response.data);

        // const token = "Basic " + window.btoa(username + ":" + password);
        const token = "Bearer " + response.data.accessToken;
        storeToken(token);

        saveLoggedInUser(username);
        navigator("/todos");

        window.location.reload(false);
      })
      .then((error) => console.log(error));
  }
```

Role Based Access Feature in React App

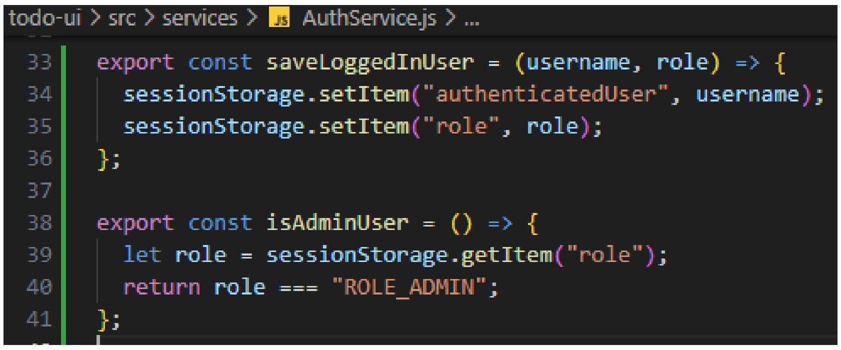


Development Steps

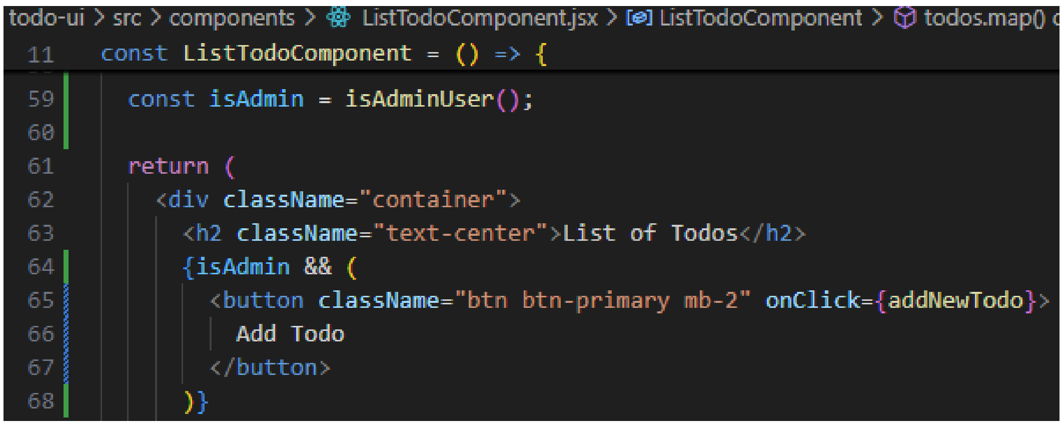
1. Change **LoginComponent.handleLoginForm()** method to get role from the response of Login REST API
2. Change **AuthService.saveLoggedInUser()** method to save username and role into session storage
3. Create **isAdminUser()** method in a AuthService to check wether the logged-in user is admin or not
4. Display the buttons (add Todo, Update Todo and Delete Todo) based on User role in ListTodoComponent


todo-ui > src > components > LoginComponent.jsx > LoginComponent > handleLoginForm > th

```
const LoginComponent = () => {
  function handleLoginForm(e) {
    e.preventDefault();

    loginAPICall(username, password)
      .then((response) => {
        console.log(response.data);

        // const token = "Basic " + window.btoa(username + ":" + password);
        const token = "Bearer " + response.data.accessToken;
        storeToken(token);

        const role = response.data.role;

        saveLoggedInUser(username, role);
        navigator("/todos");

        window.location.reload(false);
      })
      .then((error) => console.log(error));
  }
```


todo-ui > src > services > AuthService.js > ...

```
export const saveLoggedInUser = (username, role) => {
  sessionStorage.setItem("authenticatedUser", username);
  sessionStorage.setItem("role", role);
};

export const isAdminUser = () => {
  let role = sessionStorage.getItem("role");
  return role === "ROLE_ADMIN";
};
```


todo-ui > src > components > ListTodoComponent.jsx > ListTodoComponent > todos.map() c

```
const ListTodoComponent = () => {
  const isAdmin = isAdminUser();

  return (
    <div className="container">
      <h2 className="text-center">List of Todos</h2>
      {isAdmin && (
        <button className="btn btn-primary mb-2" onClick={addNewTodo}>
          Add Todo
        </button>
      )}
```