Table of Contents

[Homework 1 1](#_Toc30138405)

[Requirements 1](#_Toc30138406)

[Recommendation: 2](#_Toc30138407)

[Routing Review 2](#_Toc30138408)

[Routing with Parameters 4](#_Toc30138409)

## Homework 1

For this assignment you will create a client and server application that manages Todo objects. The MVC portion of the .NET application can be used to register users and to assign roles. The API portion of the .NET application provides the backend for login, logout and Todo CRUD. Authentication is managed with JSON Web Tokens. The React application presents the web interface pages for login, logout and Todo management.

**Submission Instructions**

Submit a zipped-up folder that contains exactly two folders:

* 1. The first folder contains the .NET project which uses SQLite.
  2. The second folder contains the React project without the node\_modules folder. (2 marks will be deducted if the node\_modules folder is included).

Do not use the React template that is included in .NET or marks will be deducted.

Worth 20%

**Note:**

You are encouraged to work with others and you can discuss your solutions over-the-shoulder as well but DO NOT SHARE COPIES of YOUR WORK ELECTONICALLY OR IN ANY OTHER FORM. Keep good karma with your career network and protect your reputation – do not plagiarize.

20% will be deducted each day this item is late.

## Requirements

The client and server application work together to ensure:

* Each Todo object stores ID, Description, IsComplete and UserName.
* 5/5 CRUD is properly enabled for Todos. This includes read all and read by id only.
* 8/8 Current users are only able to view and manage their own Todos from the React application.
* 4/4 Administrators and managers can view a listing of all users from the server in the React application. Users without these roles are not permitted to access the user listing at any time.
* 3/3 In the React application, the user’s name appears beside the Logout link while they are logged in.
* 3/3 The Todos menu link in the React application only appears when the user is logged in.
* 3/3 The Users menu link in the React application is only visible to administrators and managers when they are logged in. This link is not visible to other users.
* 3/3 The user never submits their name from the React application when making a request of any kind for Todo items. The server application must automatically determine who the current user is by reading the JSON web token that is generated at login.

Deductions:

* Does not use repositories for database code.
* Too much code is placed in the controller.
* Poor naming.
* Does not use SQLite.

## Recommendation:

Start with the roleDemo.zip application from day 1. Then add in the ability to manage JSON webtokens as described in the day 2 notes.

## Routing Review

In case you need it, these are my notes on routing:

Routing allows you to create content which can be accessed by different hyperlinks on a site. The router selects the component based on the chosen path. Redirect options are available for deprecated links. A wild card(\*) option is also available to handle invalid paths.

Note: The wild card(\*) is a default condition which is selected if nothing before it is selected. The wild card(\*) must be placed at the end of the list.

{/\* Our router goes here \*/}

<Switch>

<Route exact path="/" component={Home} />

<Route exact path="/new" component={New} />

{/\* Does a redirect. \*/}

<Redirect from="/old" to="/new" />

<Route path={'/about'} exact component={About} />

{/\* Shows an error page. \*/}

<Route path="\*" component={NotFound} />

</Switch>

Links can then be set with the following syntax:

<ul>

<li><Link to="/">Home</Link></li>

<li><Link to="/about">About</Link></li>

<li><Link to="/old">Old Page</Link></li>

</ul>

Example 1: Basic Routing

This example demonstrates basic routing in React. To enable routing in your application, run the following command when you are located in the same folder as the package.json file:

npm install --save react-router-dom

Then add the following components:

**Home.js**

|  |
| --- |
| import React, { Component } from 'react';  class Home extends Component {  render() {  return <h1>Home!</h1>;  }  }  export default Home; |

**About.js**

|  |
| --- |
| import React, { Component } from 'react';  class About extends Component {  render() {  return <h1>About!</h1>;  }  }  export default About; |

**NotFound.js**

|  |
| --- |
| import React from 'react';  const NotFound = () => (  <h1>This page cannot be found.</h1>  );  export default NotFound; |

**New.js**

|  |
| --- |
| import React, { Component } from 'react';  class New extends Component {  render() {  return <h1>New!</h1>;  }  }  export default New; |

Next, replace the contents of App.js with this version:

**App.js**

|  |
| --- |
| import React, { Component } from 'react';  import { BrowserRouter as Router, Route, Link, Switch }  from "react-router-dom";  import Home from "./Home";  import About from "./About";  import NotFound from './NotFound';  import New from './New';  import { Redirect } from 'react-router';  class App extends Component {  render() {  return (  <Router>  <div>  <ul>  <li><Link to="/">Home</Link></li>  <li><Link to="/about">About</Link></li>  <li><Link to="/old">Old Page</Link></li>  </ul>  <b>Header</b>  {/\* Our router goes here \*/}  <Switch>  <Route exact path="/" component={Home} />  <Route exact path="/new" component={New} />  {/\* Does a redirect. \*/}  <Redirect from="/old" to="/new" />  <Route path={'/about'} exact component={About} />  {/\* Shows an error page. \*/}  <Route path="\*" component={NotFound} />  </Switch>  <h3>Footer</h3>  </div>  </Router>);  }  }  export default App; |

When you run your project now you will be able to view different component content with corresponding hyperlinks.

Exercise 1

Add a contact page to . Then add a hyperlink to App.js so you can view it. Show you contact page component here:

|  |
| --- |
| import React, { Component } from 'react';  class Contact extends Component {  render() {  return <div>  <h1>Contact</h1>  <p>Email: DoNotReply@example.com</p>  </div>;  }  }  export default Contact; |

Show your revised App.js file here:

|  |
| --- |
| import React, { Component } from 'react';  import { BrowserRouter as Router, Route, Link, Switch }  from "react-router-dom";  import Home from "./Home";  import About from "./About";  import NotFound from './NotFound';  import New from './New';  import Contact from ‘./Contact’;  import { Redirect } from 'react-router';  class App extends Component {  render() {  return (  <Router>  <div>  <ul>  <li><Link to="/">Home</Link></li>  <li><Link to="/about">About</Link></li>  <li><Link to="/old">Old Page</Link></li>  <li><Link to="/contact">Contact</Link></li>  </ul>  <b>Header</b>  {/\* Our router goes here \*/}  <Switch>  <Route exact path="/" component={Home} />  <Route exact path="/new" component={New} />  {/\* Does a redirect. \*/}  <Redirect from="/old" to="/new" />  <Route path={'/about'} exact component={About} />  <Route path="/contact" component={Contact} />  {/\* Shows an error page. \*/}  <Route path="\*" component={NotFound} />  </Switch>  <h3>Footer</h3>  </div>  </Router>);  }  }  export default App; |

### 

### Routing with Parameters

Passing parameters through a hyperlink is easy once you have routing set up.

Example 2: Passing Parameters

To pass parameters to a component you include the parameters in the route:

<Route path={'/about/:id/:fname'} exact component={About} />

Then you can pass parameters in the link:

<Link to="/about/5/panda">About</Link>

The parameters can then be retrieved in the componentDidMount() function which is called after the component is built.

import React, { Component } from 'react';

class About extends Component {

render() {

return <h1>About!</h1>;

}

componentDidMount () {

alert("The " + this.props.match.params.fname + " is " +

this.props.match.params.id + " years old.");

}

}

export default About;

Example 3: Route Parameters

This example shows how to implement route parameters to pass data to the child. Starting with , the following route adjustment is needed in App.js

|  |
| --- |
| <Route path={'/about/:id/:fname'} exact component={About} /> |

Next the following link adjustment is needed in App.js:

|  |
| --- |
| <li><Link to="/about/5/panda">About</Link></li> |

After, replace the contents of your about.js page with the following code:

**components/about.js**

|  |
| --- |
| import React, { Component } from 'react';  class About extends Component {  render() {  return <h1>About!</h1>;  }  componentDidMount () {  alert("The " + this.props.match.params.fname + " is " +  this.props.match.params.id + " years old.");  }  }  export default About; |

Exercise 2

Modify so you can pass your first and last name to a contact page through a hyperlink parameter. Show your first name and last name on the contact page after retrieving this data from the parent. Show your contact page here:

|  |  |
| --- | --- |
| import React, { Component } from 'react';  class About extends Component {  render() {  return <h1>About! { this.props.match.params.fname } { this.props.match.params.lname}</h1>;  }  }  export default About; |  |

Show your revised App.js file here with the adjusted routing and hyperlink:

|  |  |
| --- | --- |
| import React, { Component } from 'react';  import { BrowserRouter as Router, Route, Link, Switch }  from "react-router-dom";  import Home from "./Home";  import About from "./About";  import NotFound from './NotFound';  import New from './New';  import Contact from ‘./Contact’;  import { Redirect } from 'react-router';  class App extends Component {  render() {  return (  <Router>  <div>  <ul>  <li><Link to="/">Home</Link></li>  <li><Link to="/about">About</Link></li>  <li><Link to="/old">Old Page</Link></li>  <li><Link to="/contact">Contact</Link></li>  </ul>  <b>Header</b>  {/\* Our router goes here \*/}  <Switch>  <Route exact path="/" component={Home} />  <Route exact path="/new" component={New} />  {/\* Does a redirect. \*/}  <Redirect from="/old" to="/new" />  <Route path={'/about/:fname/:lname'} exact component={About} />  <Route path="/contact" component={Contact} />  {/\* Shows an error page. \*/}  <Route path="\*" component={NotFound} />  </Switch>  <h3>Footer</h3>  </div>  </Router>);  }  }  export default App; |  |