Exercise 10 – Single Page Applications

Objectives

To be able to add routing to a monolithic application to make it a navigable single page application.

Overview

The context for this exercise is a fan-site for Star Wars. At present, the application has been created using React but the whole site is on a single web page. The owner would like you to change the application so that it is presented with the following:

- A navigation bar that is always present throughout the application
- Home page content, constituting an introduction to the website
- A page for films that has links to the original 3 films. Clicking on the link should:
  - Display the name of the film
  - Display a synopsis of the film
- A page for actors that has links to the 3 biggest stars. Clicking on the link should:
  - Display the name of the actor and a reference to the character that they played
  - A short biography

Styling and other functional JavaScript (such as Foundation and jQuery) has been provided to allow you to focus on the React part of the exercise.

Exercise Instructions

Check that the starter project runs correctly

1. In the command line, navigate into the **starter** folder for **EG10**.
2. Run the command:

```
npm install
```

3. Now run the command:

```
npm start
```

Verify that the browser outputs a single page that has borders around the three sections of the page. These are indicators as to which content needs to be on its own 'page'.

Install react-router-dom for the project

1. Open a command line and navigate to the **EG10 starter** folder.
2. Install **React Router DOM** through **npm** by using the command:

```
npm install --save react-router-dom
```

Adding a navigation bar to the Header component

Create the MainNav component

If you inspect the current **Header** component, found in the **src/main** folder, you will see that it already has a **Title** component rendered within it. The **Header** will be present on every page, so this is a good place to put the site navigation. To include the navigation, we will create a new component called **MainNav**, which will be positioned by CSS at the right-hand side of the **Header**. It will be styled to follow the style of the site, so ensure that you follow the instructions for classes and ids closely!

1. In **src/main**, create a new file called **MainNav.jsx**.
2. Add an **import** for **React**.
3. Create a functional component called **MainNav** that returns the following:
   - A wrapping **nav** element with the **className** of **top-bar-right** and an **id** of **menu**;
   - A **h6** element with the **className** of **hide** and the content of **Site navigation**;
   - A **ul** element with the **className** of **vertical medium-horizontal menu** with:
     - An **li** that contains a **Link** with the **className** of **pageLink**, a **to** property of **'/'** and the content **Home**;
     - An **li** that contains a **Link** with the **className** of **pageLink**, a **to** property of **'/films'** and the content **Films**;
     - An **li** that contains a **Link** with the **className** of **pageLink**, a **to** property of **'/actors'** and the content **Actors**.
4. **Import** the *non-default export* **Link** from **'react-router-dom'** (i.e. **{ Link }**).
5. Save the file.

Add the MainNav component to the Header

1. Open **Header** from the **src/main** folder.
2. After the **Title** component, add in the **MainNav** component
3. **Import** the **MainNav** component.
4. Save the file.

Tell React how to use the Links

To be able to render the **Link** component, it must be a child of a **Router** component. Since the whole application will need to use a **Router** component, we will add this to the **App** component, making it wrap the current **return** mark-up.

1. Open **App.jsx**.
2. Add a wrapping **Router** component around the mark up currently returned.
3. **Import** the **Router** using:

```
import { BrowserRouter as Router} from 'react-router-dom'.
```

4. Save the file and observe the output.

You should see that the navigation bar has been added to the Header. Clicking on each of the links should change the address in the address bar but not affect the rendered application.

Routing the Application from the MainNav

Now that we have the main navigation bar in the header and this activates different routes, we need to make the Application respond to these clicks and change the content in the main part of the page. This is all done in the App component by replacing the static components with Routes.

1. Open **App.jsx**.
2. Add **Route** to the **imports** from **react-router-dom**.
3. Replace the **Home** component with a **Route** component that has:
   - An **exact** property;
   - A **path** property of **"/"**;
   - A **component** property of **{ Home }**;
4. Replace the **Films** component with a **Route** component that has:
   - A **path** property of **"/films"**;
   - A **component** property of **{ Films }**;
5. Replace the **Actors** component with a **Route** component that has:
   - A **path** property of **"/actors"**;
   - A **component** property of **{ Actors }**;
6. Save the file and check that the application works with the routing applied.

Adding links to the films

The specification supplied at the start of the exercise told us we need:

- A page for films that has links to the original 3 films. Clicking on the link should:
  - Display the name of the film
  - Display a synopsis of the film

We already have the page for films, so now we need to address the links to the original films and making this change the content in this component dependent on the film that is click on.

To do this, we need to add **Link** components to some content on the page and then use a *parameterised route* to select the correct film to display.

To keep each component as concise as possible we will create a **FilmLinks** component to create the **Links** in.

Creating the FilmLinks component

1. In **src/films**, create a **FilmLinks** component.
2. Import **React** and **Link** (from **react** and **react-router-dom**).
3. Create a functional component called **FilmLinks** accepting **props** that **returns**:
   - A wrapping **nav** component;
   - A **ul** with a **className** of **vertical medium-horizontal menu items**;
   - *Three list items* inside the **ul** containing **Link** components, all with a **className** of **active** and a **to** property that starts with **props.match.url** and:

- *For Star Was IV* - ends with **/SW4** and has *content* of **Star Wars IV - A New Hope**;
- *For Star Wars V* - ends with **/SW5** and has *content* of **Star Wars V - The Empire Strikes Back**;
- *For Star Wars VI* - ends with **/SW6** and has *content* of **Star Wars VI - Return of the Jedi**;

4. Save the file.
5. In **Films**, **import** the **FilmLinks** component for use.
6. Pass **props** into the function
7. Insert a **FilmLinks** component with an attribute of **match** set to **props.match** *in between* the **h2** and the **div** that wraps **FilmDescription**.
8. Save the file and check that the URL changes in accordance with the link your clicks.

Activating the links to the films

Currently, all of the films are shown below the links we have just created. We want this area to only show the film the user has clicked on here. To do this, we need to create parameterised routes, based on the last portion of the URL. This should select the correct film from the array and render this in the **FilmDescription** component.

Provide Routing information to Films

The first step here is to provide routing information in the **Films** component.

1. Open Films.jsx
2. Replace the FilmDescription component with a Route component that has:
   - A path attribute set to `${props.match.url}/:filmId}`;
   - A component attribute set to {FilmDescription}.

So that we have something rendered in this space when we land on the Films page we need to render some HTML on that Route.

3. Add another Route component that has:
   - An exact attribute;
   - A path attribute set to `${props.match.url}`;
   - A render attribute that is an arrow function that takes no arguments and returns:
     - A h6 element with the content 'Please select a film to see its details'.
4. Save the file.

(Checking the output at this stage should show that the URL changes but the output doesn't still).

Identify the film to display

Next is to get the **FilmDescription** component to recognise which film it should be displaying. We will do this by examining the **params** supplied in the URL that has activated this **Route** and selecting the appropriate **film** from the **films** array.

1. Open **FilmDescription.jsx**.

2. Remove the **map** function from your code.
3. Replace it so that a variable **filmToDisplay** deconstructs the array returned by a **filter** of the **films** array where we look for *equality* between **film.id** and the **filmId param** that is supplied by **props.match**.
4. Replace the mark up with a wrapping **div** that has:
    - A **h3** element with a **className** of **filmTitle** and *content* of **filmToDisplay.title**;
    - A **p** element with *content* of **filmToDisplay.synopsis**.
5. Save the file.

You should see that the sub menu system is now working and that clicking on the film link shows the detail for the film.

Note: if you lose the styling for the page, go back to the Home page and refresh.

Do or do not, there is no try!

Time to fly '*Solo*'...Complete the application by making the **Actors** section work as it should. There's an even chance that you will be able to do it, but then again, "*Never tell me the odds!*"

As a reminder:

- A page for actors that has links to the 3 biggest stars. Clicking on the link should:
    - Display the name of the actor and a reference to the character that they played
    - A short biography

"*Remember the Force* [and the solution code] *will be with you, always!*"