WEB422 Assignment 3

Submission Deadline:

End of Week 6

Assessment Weight:

9% of your final course Grade

Objective:

To continue to work with our "Movies" API (from Assignment 1) on the client-side to produce a rich user interface for accessing data. For this assignment, we will be leveraging our knowledge of React and Next.js to create an interface for *viewing* movies. **Please Note**: Once again, you're free to style your app however you wish. The following specification outlines how we can use the React-Bootstrap Components (Bootstrap 5). If you wish to add additional images, styles or functionality, please go ahead.

Sample Solution:

You can see a video of the solution running at the link below:

https://pat-crawford-sdds.netlify.app/shared/winter-2023/web422/A3/A3.mp4

Step 1: Creating a Next App & Adding 3rd Party Components

The first step is to create a new directory for your solution, open it in Visual Studio Code and open the integrated terminal:

- Next, proceed to create a new Next.js app by using "create-next-app" with the command "npx create-next-app" followed by the identifier for your app, ie: "my-app" and the "--use-npm" option.
- Once the tool has finished creating your Next App, be sure to "cd" into your new app directory and install the following modules using npm:
  - swr
  - bootstrap react-bootstrap
- To ensure that our newly-added Bootstrap components render properly, we must import the correct CSS file to our "pages/_app.js" file, ie:
  - import 'bootstrap/dist/css/bootstrap.min.css';
- Next, we must delete (or clear the CSS from) the "Home.module.css" file and clear the existing CSS from "globals.css", since we won't be using any of those rules within our new app. Once this is complete, place the following single line of CSS within the "globals.css" file (this will prevent stuttering when expanding elements due to the addition / removal of the scrollbar): **body{ overflow-y:scroll; }**

Step 2: Component "Skeletons"

For the next part of the assignment, we'll add our "page" / custom components (with some default text) as well as create a layout for our app.

NOTE: Components that output a simple <p>…</p> element will be overwritten further in the assignment

To proceed, add the following components:

- **Movie (pages/movies/[title].js)** – outputs <p>Movie by Title</p>
- **About (pages/about.js)** – outputs <p>About</p>
- **Home (pages/index.js)** – Remove existing JSX and all imports and modify this to output <p>Movies</p>
- **MainNav (components/MainNav.js)** – outputs <p>MainNav</p>
- **Layout (components/Layout.js)** – outputs:

  ```
  <MainNav />
  <br />
  <Container>
    {props.children}
  </Container>
  <br />
  ```

  NOTE: the "Container" is from "react-bootstrap", ie: import { Container } from 'react-bootstrap';
- **MovieDetails (components/MovieDeails.js)** – outputs <p>MovieDetails</p>
- **PageHeader (components/PageHeader.js)** – outputs <p>PageHeader</p>

Step 3: MyApp Component & NavBar

Before we start building out our components, we must first do some work with the "MyApp" component (pages/_app.js):

- To begin, add the following import statements
  - import Layout from '**@/components/Layout**';
  - import { SWRConfig } from '**swr**';
- Next, wrap the <Component {...pageProps} /> component with the following SWRConfig component to globally define the fetcher:

  ```
  <SWRConfig value={{ fetcher: (...args) => fetch(...args).then((res) => res.json()) }}></SWRConfig>
  ```
- Finally, wrap the "SWRConfig" element (above) with our <Layout></Layout> component

With this complete, we can now create our "MainNav" component, so that our navbar shows more than just <p>MainNav</p>:

To begin, we should (at a minimum) have the following components imported:

- **Container, Nav, Navbar** from "react-bootstrap"
- **Link** from "next/link"

Next, use the documentation from the "Navbars" section of the React-Bootstrap docs (https://react-bootstrap.netlify.app/components/navbar) to obtain the JSX code for a functioning Navbar according to the following specification:

- Shows ***Student Name*** in the **Navbar.Brand** element (where ***Student Name*** is your name) without an href element (ie remove href="#home" from **Navbar.Brand**).

- The **Navbar** element should have a "className" value of "fixed-top". You can also use "navbar-dark" and "bg-dark" to achieve a design that matches the example.

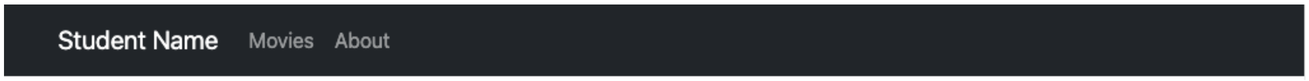
- Contains 2 <Nav.Link> elements with the text "**Movies**" and "**About**", linking to "**/**" (for **Movies**) and **"/about**" for (**About**)

    - **NOTE:** For these elements to function correctly, they must be wrapped in <Link> elements containing appropriate href properties and the "passHref" property. For example, the "Movies" Nav.Link element should be:

      <Link href="/" passHref legacyBehavior><Nav.Link>Movies</Nav.Link></Link>

- There should be **two** line breaks (<br />) after the **Navbar** element – this will ensure that the content can be seen below the *fixed* **Navbar**

- Once complete, you should have a responsive Navbar that looks similar to the following image (with your own Name in place of **Student Name**. The "Movies" item should link to "/", while the "About" item should link to "/about":



Step 4: Page Header Component (PageHeader.js)

With the navbar and other components in place we can concentrate on building a reusable "Page Header" element for our various views according to the following specification:

- The component must accept a custom property (prop), **text** (ie: <PageHeader text="some text" />

- The component must render the text prop in some kind of container. To achieve the same design as the example, a "Card" (see: https://react-bootstrap.github.io/components/cards) was used with the className "bg-light" and the text value rendered within a <Card.Body>…</Card.Body> element (**NOTE:** <Card.Title>, <Card.Text> and <Card.Image> was not used)

- There should be a line break (<br />) after the container

Step 5: About Component (about.js)

This component is primarily responsible for displaying information about the developer, including some information about a specific movie available from our "Movies API" from Assignment 1.

To begin, we should (at a minimum) have the following components imported:

- **Link** from "next/link";
- **Card** from "react-bootstrap";
- **MovieDetails** from "@/components/MovieDetails";
- **PageHeader** from "@/components/PageHeader";

Additionally, this component must make use of "getStaticProps" (See "Fetching API Data for Pre-Rendered HTML" in the notes) to fetch a specific movie from the API.

> **HINT:** This will involve figuring out what the "_id" is for a movie that you would like to show. To explore the dataset for a movie that you would like to use, you can try searching the API in a web browser using the path:
>
> **(Your Cyclic App)**/api/movies?page=1&perPage=10&title=**Some Title**
>
> You can then record the _id value of the movie that you would like to pre-render for your "About" component

- Once you know the _id value of the movie you would like, make a "fetch" request within your promise-based "getStaticProps" function to fetch the movie from your API using the path: **(Your Cyclic App)**/api/movies/**_id**.

- Once the operation is successful, resolve the promise with an object containing the "props" keyword, with the data for your movie (returned from the "fetch" request"), ie: **{props: {movie: data}}**

With our "getStaticProps" function complete, we can now ensure that our component renders the following:

> **NOTE:** To achieve the same design as the example, the text beneath the Page Header (<PageHeader />) can be placed within **<Card><Card.Body>**…**</Card.Body></Card>** elements, with the **<MovieDetails />** component placed outside of the <Card.Body>…</Card.Body>, before the closing </Card> tag.

- A <PageHeader /> element with the text "About the Developer" followed by your name.

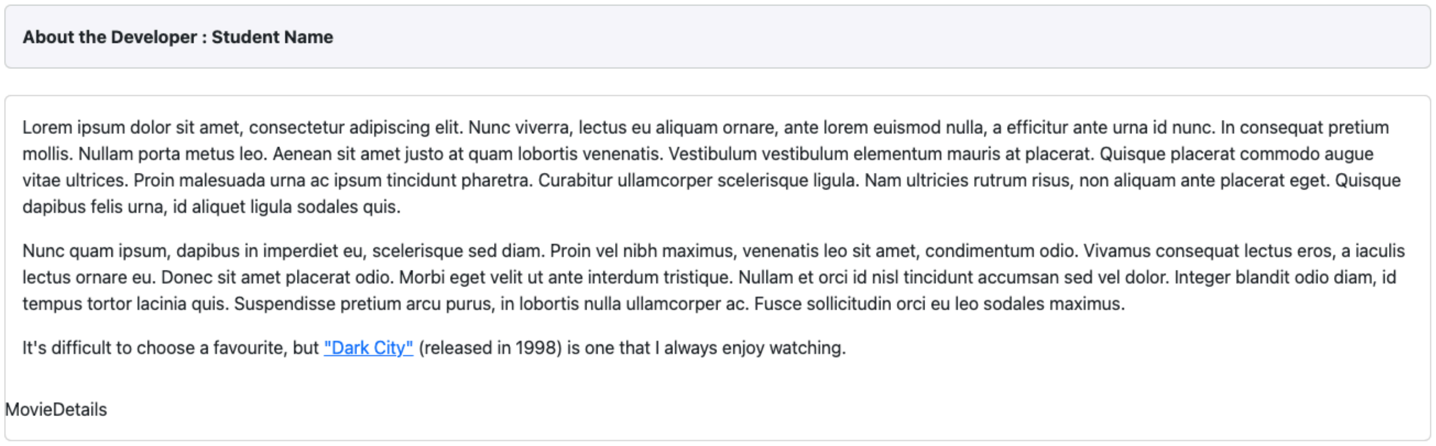
- A short description of yourself in one or more paragraphs

- A link to a specific movie available from our "Movies API" using the path "/movies/**Some Title"** where **Some Title** is the title of the movie you wish to show, ie: "/movies/Dark City". **NOTE**: We will build the associated component (defined in "pages/movies/[title].js") further down in the assignment

- A "MovieDetails" element with a "movie" property containing the data from your "getStaticProps" function, ie: **<MovieDetails movie={props.movie} />**
  (**NOTE**: we will build this component further down in the assignment)

Step 6: Movie Details Component (MovieDetails.js)

At the moment our "About" page should look something like this:



About the Developer : Student Name

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc viverra, lectus eu aliquam ornare, ante lorem euismod nulla, a efficitur ante urna id nunc. In consequat pretium mollis. Nullam porta metus leo. Aenean sit amet justo at quam lobortis venenatis. Vestibulum vestibulum elementum mauris at placerat. Quisque placerat commodo augue vitae ultrices. Proin malesuada urna ac ipsum tincidunt pharetra. Curabitur ullamcorper scelerisque ligula. Nam ultricies rutrum risus, non aliquam ante placerat eget. Quisque dapibus felis urna, id aliquet ligula sodales quis.

Nunc quam ipsum, dapibus in imperdiet eu, scelerisque sed diam. Proin vel nibh maximus, venenatis leo sit amet, condimentum odio. Vivamus consequat lectus eros, a iaculis lectus ornare eu. Donec sit amet placerat odio. Morbi eget velit ut ante interdum tristique. Nullam et orci id nisl tincidunt accumsan sed vel dolor. Integer blandit odio diam, id tempus tortor lacinia quis. Suspendisse pretium arcu purus, in lobortis nulla ullamcorper ac. Fusce sollicitudin orci eu leo sodales maximus.

It's difficult to choose a favourite, but "Dark City" (released in 1998) is one that I always enjoy watching.

MovieDetails

To complete this page, we should build the reusable "MovieDetails" component next. As we have seen above, this component should accept the prop "movie", which contains a full movie object from our API. It is the job of this component, to render the data according to the following specification:

To begin, we should (at a minimum) have the following components imported:

- **Container, Row, Col** from "react-bootstrap"

Next, we must ensure that the component renders a single React Bootstrap "Container" element, containing one "Row" element. Additionally:

- If the "poster" property exists in the "movie", it must be displayed using an <img className="w-100" /> element, placed within a <Col md>…</Col> element within the <Row>.

  For example, if the "poster" property **does exist** in the "movie", then the following column should be rendered as the first child of the "Row" element:

  <Col md><img src=**poster** alt="poster" className="w-100" /><br /><br /></Col>

  if the "poster" **does not exist**, do not render the above <Col md>…</Col> element

- Next, the following html must be rendered within a <Col md>…</Col> element, also placed within the "Row" element (**NOTE:** This is identical to the data beneath the "poster" in the modal window for Assignment 2)

  For example, once again using the data from "The Matrix"
  (ie: /api/movies/573a139bf29313caabcf3d23), the following HTML must be generated

<strong>Directed By:</strong> **Andy Wachowski, Lana Wachowski**<br><br>
<p>**Thomas A. Anderson is a man living two lives.**</p>
<strong>Cast:</strong> **Keanu Reeves, Laurence Fishburne, Carrie-Anne Moss, Hugo Weaving**<br><br>
<strong>Awards:</strong> **Won 4 Oscars. Another 33 wins &amp; 40 nominations.**<br>
<strong>IMDB Rating:</strong> **8.7** (**1080566** votes)

The HTML shown above can be obtained using the following properties from the returned JSON data:

- **Andy Wachowski, Lana Wachowski** – obtained from the **directors** property (Array displayed using the "join" method with a value of ', ')

- **Thomas A. Anderson is a man living two lives.** – (**NOTE:** this is a shortened version of the full value from the API, shown here to save space) obtained from the **fullplot** property

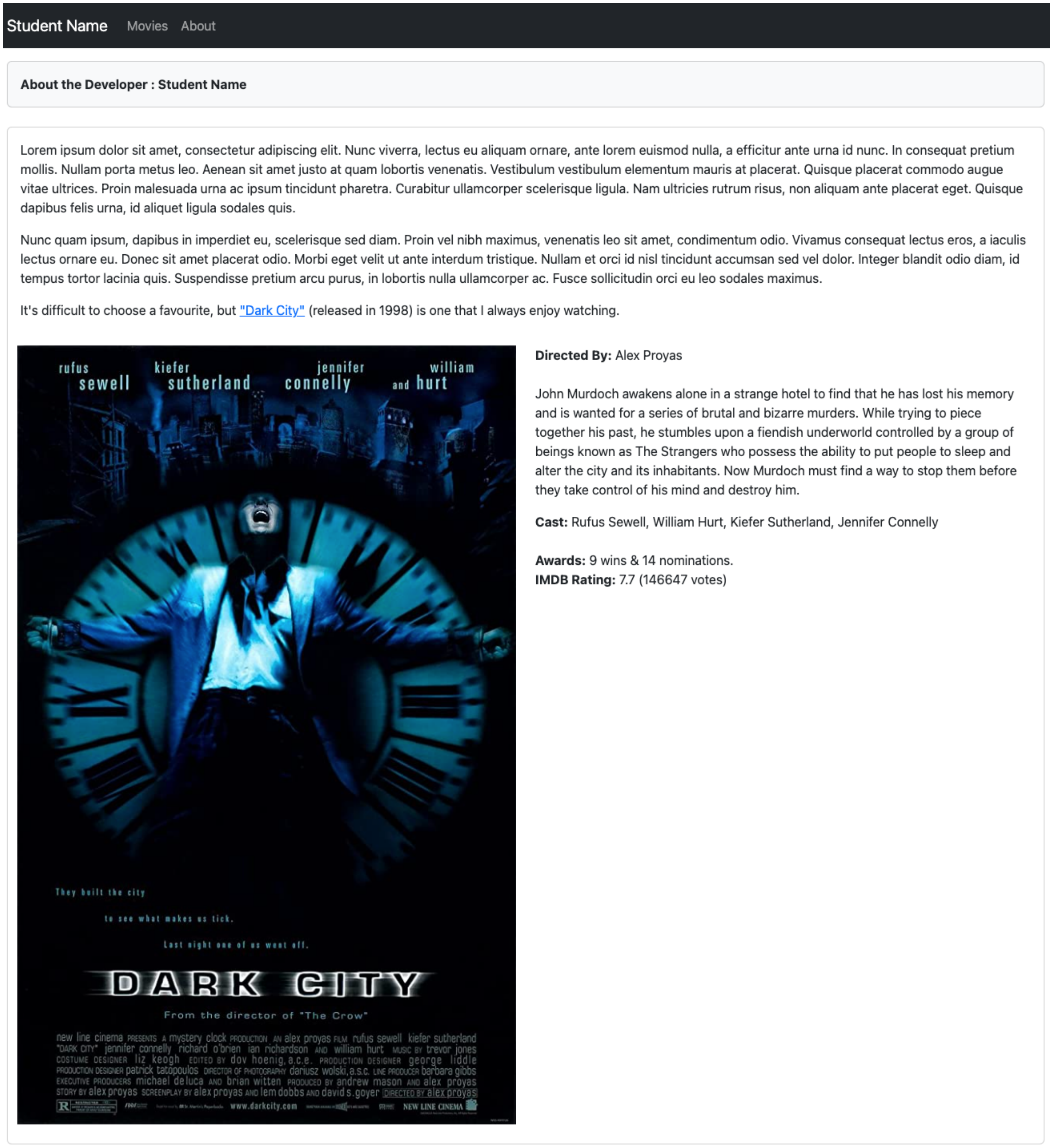
- **Keanu Reeves, Laurence Fishburne, Carrie-Anne Moss, Hugo Weaving** – obtained from the **cast** property (Array displayed using the "join" method with a value of ', '). (**NOTE:** There are some movies in the dataset without a cast. If this is the case, show **N/A** instead)

- **Won 4 Oscars. Another 33 wins &amp; 40 nominations.** – obtained from the **awards.text** property

- **8.7** – obtained from the **imdb.rating** property

- **1080566** – obtained from the **imdb.votes** property

Once you have completed this component, refresh your "About" page. It should show the movie that you obtained from your "getStaticProps" function, since it was provided to MovieDetails, via the "movie" prop. For example, if the movie data for "Dark City" was obtained in the "getStaticProps" function:



Student Name Movies About

About the Developer : Student Name

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc viverra, lectus eu aliquam ornare, ante lorem euismod nulla, a efficitur ante urna id nunc. In consequat pretium mollis. Nullam porta metus leo. Aenean sit amet justo at quam lobortis venenatis. Vestibulum vestibulum elementum mauris at placerat. Quisque placerat commodo augue vitae ultrices. Proin malesuada urna ac ipsum tincidunt pharetra. Curabitur ullamcorper scelerisque ligula. Nam ultricies rutrum risus, non aliquam ante placerat eget. Quisque dapibus felis urna, id aliquet ligula sodales quis.

Nunc quam ipsum, dapibus in imperdiet eu, scelerisque sed diam. Proin vel nibh maximus, venenatis leo sit amet, condimentum odio. Vivamus consequat lectus eros, a iaculis lectus ornare eu. Donec sit amet placerat odio. Morbi eget velit ut ante interdum tristique. Nullam et orci id nisl tincidunt accumsan sed vel dolor. Integer blandit odio diam, id tempus tortor lacinia quis. Suspendisse pretium arcu purus, in lobortis nulla ullamcorper ac. Fusce sollicitudin orci eu leo sodales maximus.

It's difficult to choose a favourite, but "Dark City" (released in 1998) is one that I always enjoy watching.

Directed By: Alex Proyas

John Murdoch awakens alone in a strange hotel to find that he has lost his memory and is wanted for a series of brutal and bizarre murders. While trying to piece together his past, he stumbles upon a fiendish underworld controlled by a group of beings known as The Strangers who possess the ability to put people to sleep and alter the city and its inhabitants. Now Murdoch must find a way to stop them before they take control of his mind and destroy him.

Cast: Rufus Sewell, William Hurt, Kiefer Sutherland, Jennifer Connelly

Awards: 9 wins & 14 nominations.
IMDB Rating: 7.7 (146647 votes)

Step 7: Home Component (index.js)

We now have all of the pieces in place to build our "Home" component. This is arguably one of the more complicated components as it must display data from our "Movies" API, one page at a time (similar to what was achieved in Assignment 2).

To begin, we should (at a minimum) have the following components imported:

- **useSWR** from 'swr';
- **useState, useEffect** from 'react';
- **Pagination, Accordion** from 'react-bootstrap';
- **MovieDetails** from '@/components/MovieDetails';
- **PageHeader** from '@/components/PageHeader';

Next, add the following **state** values to the component:

- **page** (default value: 1)
- **pageData** (default value: [])

With state values in place, we can now use SWR to make a request for the data, ie:

- const { data, error } = useSWR(`**(Your Cyclic App)**/api/movies?page=**page**&perPage=10`);

We will be using the data returned from useSWR to set our "pageData" state value, rather than using it directly within our JSX for this component. To ensure that whenever the "data" has been successfully updated (as a result of a new page being requested, for example), we can leverage the "useEffect" hook as follows (assuming "setPageData()" is your setter function for your **page** state value):

```
useEffect(() => {
  if (data) {
    setPageData(data);
  }
}, [data]);
```

Finally, before we can write the return statement (JSX) for this component, we must ensure that we have two functions declared:

- **previous** – decreases the **page** state value by **1** if it is greater than **1**
- **next** – increases the **page** state value by **1**

As you have seen from the example, the output for this component is an Accordion element, featuring detailed movie information. Like assignment 2, movie data can be obtained one page at a time, where each page shows 10 items. To create this UI, we must include:

- A <PageHeader /> element with the text "Film Collection : Sorted by Date"
- An <Accordion> element containing one <Accordion.Item> element for each movie in the **pageData** array according to the following specification:
    - The <Accordion.Item> element must have an "**eventKey**" property (with the value of the current movie's "_id" value) and a "**key**" property (with the current movie's "_id" value)

  - The <Accordion.Header> element must show the values of:
    - The current movie's "title" value (in **bold**)
    - The current movie's "year" value
    - The current movie's "directors" array
  - The <Accordion.Body> element must contain a **<MovieDetails />** component with a "movie" property containing the full current movie object (this will render all of the information for the movie within the <Accordion.Body>

- A Pagination element (<Pagination>) containing the following elements:
  - <Pagination.Prev /> with a "click" event that executes the "previous" function
  - <Pagination.Item /> which shows the current **page** value
  - <Pagination.Next /> with a "click" event that executes the "next" function

Step 8: Movie Component (movies/[title].js)

The final component that we must create is the dynamic "Movie" component. The purpose of this movie is to show a **<MovieDetails />** component for a specific movie, specified by the "title" route parameter. If a movie cannot be found with a matching title, the default Next.js error is shown.

To begin, we should (at a minimum) have the following components imported:

- **useRouter** from 'next/router';
- **useSWR** from 'swr';
- **MovieDetails** from '@/components/MovieDetails';
- **Error** from 'next/error';
- **PageHeader** from '@/components/PageHeader';

Before we can make our request (using SWR) we must obtain the "title" parameter (**HINT:** this can be done with the useRouter(); hook – see "Reading Route Parameters" from the course notes)

Next, using the "title" parameter value, use SWR to make a request for the data, ie:

- const { data, error } = useSWR(`**(Your Cyclic App)**/api/movies?page=1&perPage=10&title=**title**`);

Before we can render anything however, we must check the value of "data", ie:

- if "data" is null or undefined (ie: "falsy"), return **null** (ie: don't render anything)

- If "data" is **not** null or undefined (ie: "truthy"), we must make a subsequent check, ie:
  - If "data" contains an empty array, return the component: **<Error statusCode={404} />** (this will cause the default Next.js 404 error to be displayed)

- If "data" contains an array with at least one element, we must render a single <div> element for every "movie" object in the array with a **key** property value of the movie's "_id" value. Contained within this <div> element, the following two components must be rendered:
    - A <PageHeader /> element with movie's "title" property.
    - A <MovieDetails /> element with a "movie" property containing the full movie object **<MovieDetails movie={movie} />**

Assignment Submission:

- For this assignment, you will be required to **build** your assignment before submitting. This will involve running the command:
    - **npm run build**

  to create a production build in the ".next" folder to be included in your submission. **Please Note:** you will be required to fix any errors that are identified during the build process.

- Next, add the following declaration at the top of your index.js file

```
/*********************************************************************************
*  WEB422 – Assignment 3
*  I declare that this assignment is my own work in accordance with Seneca Academic Policy.
*  No part of this assignment has been copied manually or electronically from any other source
*  (including web sites) or distributed to other students.
*
*  Name: ______________________ Student ID: __________________ Date: ____________________
*
*
********************************************************************************/
```

- Compress (.zip) the files in your Visual Studio working directory **without node_modules** (this is the folder that you opened in Visual Studio to create your client side code).

Important Note:

- **NO LATE SUBMISSIONS** for assignments. Late assignment submissions will not be accepted and will receive a **grade of zero (0)**.
- After the end (11:59PM) of the due date, the assignment submission link on My.Seneca will no longer be available.
- Submitted assignments must run locally, ie: start up errors causing the assignment/app to fail on startup will result in a **grade of zero (0)** for the assignment.