

# WEB422 Assignment 3

## Submission Deadline:

Friday, February 16<sup>th</sup> @ 11:59pm

## Assessment Weight:

9% of your final course Grade

## Objective:

To continue to work with our "Listings" 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* listings. **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:

<https://wpas-a3-winter-2024.vercel.app>

### Step 1: Creating a Next App & Adding 3<sup>rd</sup> 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@latest" 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.

## 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:

- **Listing (pages/listing/[id].js)** – outputs `<p>Listing by Id</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>Listings</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';`

- **ListingDetails (components/ListingDeails.js)** – outputs `<p>ListingDetails</p>`
- **PageHeader (components/PageHeader.js)** – outputs `<p>PageHeader</p>`

## Step 3: App Component & NavBar

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

- To begin, add the following import statements
  - `import Layout from '@components/Layout';`
  - `import { SWRConfig } from 'swr';`
- Next, add the following "fetcher" function **above** the App function:

```
const fetcher = async (...args) => {
  const response = await fetch(...args);

  if (!response.ok) {
    throw new Error(`Request failed with status: ${response.status}`);
  }

  return response.json();
};
```

- Once this is complete, place the `<Component {...pageProps} />` inside our `<Layout></Layout>` component
- Finally, place the `<Layout></Layout>` component inside the following `SWRConfig` component to make our fetcher available, globally: `<SWRConfig value={{ fetcher }}></SWRConfig>`
- When complete, your `App` component should be returning:

```
<SWRConfig value={{ fetcher }}>
  <Layout>
    <Component {...pageProps} />
  </Layout>
</SWRConfig>
```

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:

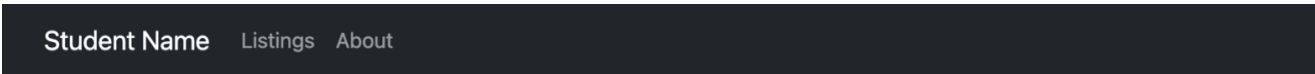
- **Container, Nav, Navbar** from "react-bootstrap"
- **Link** from "next/link"

Next, use the documentation from the "Navbars" section of the React-Bootstrap docs (<https://react-bootstrap.github.io/docs/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 "**Listings**" and "**About**", linking to "/" (for **Listings**) 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" & "legacyBehavior" properties. For example, the "Listings" `Nav.Link` element should be:

```
<Link href="/" passHref legacyBehavior><Nav.Link>Listings</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 "Listings" item should link to "/", while the "About" item should link to ***/about***:



Student Name   Listings   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/docs/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 and the card width was not set)
- 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 listing available from our "Listings API" from Assignment 1.

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

- **Link** from "next/link";
- **Card** from "react-bootstrap";
- **ListingDetails** from "@/components/ListingDetails";
- **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 listing from the API.

**HINT:** This will involve figuring out what the "\_id" is for a listing that you would like to show. To explore the dataset for a listing that you would like to use, you can try searching the API in a web browser using the path:

**(Your Cyclic App)/api/listing?page=1&perPage=10&name=Some Name**

You can then record the \_id value of the listing that you would like to pre-render for your "About" component

- Once you know the \_id value of the listing you would like, make a "fetch" request within your promise-based "getStaticProps" function to fetch the listing from your API using the path: **(Your Cyclic App)/api/listings/\_id**.
- Once the operation is successful, resolve the promise with an object containing the "props" keyword, with the data for your listing (returned from the "fetch" request), ie: **{props: {listing: 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 <ListingDetails /> 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 listing available from our "Listings API" using the path "/listing/**id**" where **id** is the title of the "\_id" value of the Listing you wish to show, ie: "/listing/1206363". **NOTE:** We will build the associated component (defined in "pages/listing/[id].js") further down in the assignment
- A "ListingDetails" element with a "listing" property containing the data from your "getStaticProps" function, ie:

**<ListingDetails listing={props.listing} />**

(**NOTE:** we will build this component further down in the assignment)

## Step 6: Listing Details Component (ListingDetails.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.

One of the places that I would like to visit is the: ["Eco Hale Hawaii in the Rainforest Lots Of Aloha"](#) Bed & Breakfast (Airbnb).

ListingDetails

To complete this page, we should build the reusable "ListingDetails" component next. As we have seen above, this component should accept the prop "listing", which contains a full listing 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, with two <Col lg></Col>Elements. Specifically:

- In the first "Col" (<Col lg>) element (ie: the left column), we will show an image of the listing using the same logic as in Assignment 2 (with a fallback image). However, as we have seen, events are handled differently in React and so we must render it using the following code (with optional line-breaks beneath it, for spacing), where **PICTURE URL** is the url of the photo that you would like to show (ie: images.picture\_url on your listing object):

```
<img
  onError={(event) => {
    event.target.onerror = null; // Remove the event handler to prevent infinite loop
    event.target.src =
      "https://placeholder.co/600x400?text=Photo+Not+Available";
  }}
  className="img-fluid w-100"
  src={PICTURE URL}
  alt="Listing Image"
/>
<br />
<br />
```

- Next, the following html must be rendered within a second "Col" (Col lg) element, also placed within the "Row" element (**NOTE:** This is identical to the data beneath the "photo\_url" in the modal window for Assignment 2, except that it also includes "Ratings")

For example, once again using the data from "Eco Hale Hawaii in the Rainforest Lots Of Aloha" (ie: /api/listing/1206363), the following HTML must be generated

```
The Puna district is a huge triangle area encompassing territory between
Hilo, Volcano, Pahoa and Kalapana ...etc. <br /><br />
<strong>Price:</strong> $80.00<br />
<strong>Room:</strong> Entire home/apt<br />
<strong>Bed:</strong> Real Bed (1)<br /><br />
<strong>Rating:</strong> 99/100 (291 Reviews)<br />
<br />
<br />
```

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

- **The Puna district is a huge triangle area encompassing territory between Hilo, Volcano, Pahoa and Kalapana ...etc.** – obtained from the **neighborhood\_overview** property (**NOTE:** Not all listings have a **neighborhood\_overview**)
- **80.00** – obtained from the **price** property (**NOTE:** Shown to two decimal places using [toFixed\(2\)](#)).



- **Entire home/apt** – obtained from the **room\_type** property
- **Real Bed** – obtained from the **bed\_type** property
- **1** – obtained from the **beds** property
- **99** – obtained from the **review\_scores.review\_scores\_rating** property
- **291** – obtained from the **number\_of\_reviews** property

Once you have completed this component, refresh your "About" page. It should show the listing that you obtained from your "getStaticProps" function, since it was provided to "ListingDetails", via the "listing" prop. For example, if the listing data for "Eco Hale Hawaii in the Rainforest Lots Of Aloha" was obtained in the "getStaticProps" function:

Student Name Listings 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.

One of the places that I would like to visit is the: ["Eco Hale Hawaii in the Rainforest Lots Of Aloha"](#) Bed & Breakfast (Airbnb).



The Puna district is a huge triangle area encompassing territory between Hilo, Volcano, Pahoa and Kalapana. Kilauea's latest eruption began on May 3, 2018 with lava flowing continuously until August 6, 2018. The affected Puna area, in the southeast corner of the island comprises ONE PERCENT of the island of Hawaii, which is larger than all of the other Hawaiian Islands combined. We are in UPPER PUNA and had No Lava. Pele will determine her future timeline, longevity and destination path. View new lava fields by air or by hiking to our newest black sand beach, Pohoiki. We have additional information on the Kilauea Flow in our Profile. The area also contains the largest remaining untouched Ohia forests in the islands, currently being threatened by Rapid Ohia Death as scientist scramble to discover the source and cure. This is a unique area and is relatively undeveloped maintaining it's origin. Agriculture is still a large part of the subsistence economy. We are located close e

**Price:** \$80.00

**Room:** Entire home/apt

**Bed:** Real Bed (1)

**Rating:** 99/100 (291 Reviews)

## Step 7: Home Component (index.js)

We now have all 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 "Listings" 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';
- **ListingDetails** from '@components/ListingDetails';
- **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/listings?page=page&perPage=10`);`

**NOTE:** You do not need to provide a "fetcher" here, since it was defined globally using the SWRConfig component (above)

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 (this will prevent flickering while the data loads). 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 listing information. Like assignment 2, listing 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 "Browse Listings : Sorted by Number of Ratings"



- An `<Accordion>` element containing one `<Accordion.Item>` element for each listing in the **pageData** array according to the following specification:
  - The `<Accordion.Item>` element must have an **"eventKey"** property (with the value of the current listings's `"_id"` value) and a **"key"** property (with the current listing's `"_id"` value)
  - The `<Accordion.Header>` element must show the values of:
    - The current listing's `"name"` value (in **bold**)
    - The current listing's `"address.street"` value
  - The `<Accordion.Body>` element must contain a **`<ListingDetails />`** component with a `"listing"` property containing the full current listing object (this will render all of the information for the listing 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></Pagination.Item>` which shows the current **page** value
  - `<Pagination.Next />` with a `"click"` event that executes the `"next"` function

## Step 8: Listing Component (listings/[id].js)

The final component that we must create is the dynamic "Listing" component. The purpose of this component is to show a **`<ListingDetails />`** component for a specific listing, specified by the `"id"` route parameter. If a listing cannot be found with a matching id, 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'`;
- **ListingDetails** from `'@/components/ListingDetails'`;
- **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 `"id"` parameter value, use SWR to make a request for the data, ie:

- `const { data, error, isLoading } = useSWR(`(Your Cyclic App)/api/listings/id`);`

Before we can render anything however, we must ensure that we are not currently `"loading"` (SWR) the data. If this is the case, we can just return **null**.

Otherwise, if we're not loading the data, we must make sure that we don't have an "error" (SWR) or that the "data" returned is valid (doesn't have a null value / empty array, etc). If this is the case, we should return the component: `<Error statusCode={404} />` (this will cause the default Next.js 404 error to be displayed).

However, if everything was successful (loading is complete, there were no errors / empty or null data, etc), then we should render the following:

- A `<PageHeader />` element with listing's "name" property.
- A `<ListingDetails />` element with a "listing" property containing the full listing object  
`<ListingDetails listing={listing} />`

### Assignment Submission:

- 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's
* Academic Integrity Policy:
*
* https://www.senecapolytechnic.ca/about/policies/academic-integrity-policy.html
*
* 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 Next.js App.

### Important Note:

- **NO LATE SUBMISSIONS** for assignments. Late assignment submissions will not be accepted and will receive a **grade of zero (0)**.
- 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.