- ⏱ 3 hours

Art Museum Project

In this project, you will create React components to build a purely frontend application that displays a list of art galleries. Using the React Router library, you will enable a user to navigate through the different art galleries to see images and descriptions of their respective art pieces. The information about the art galleries will be seeded using data extracted from the Harvard Art Museum API.

By the end of the project, you should understand and be able to use the following React Router components and hook:

- `BrowserRouter`
- `NavLink`
- `Link`
- `Route`
- `Switch`
- `useParams`

You will also learn how to specify unique keys when creating arrays of JSX elements to render.

Phase 0: Setup

Here's a breakdown of the steps you'll be taking in this phase (more detailed instruction below):

1. Create new React app!
2. cd *art-museum* and then `npm install react-router-dom` for project.
3. Start your development server.
4. Create `<Root>` component.
5. Render `<Root>` component to DOM.
6. Include `<App>` in `<Root>`.
7. Import `<BrowserRouter>` to wrap around `<App>`.
8. Create *data* folder in *src* folder.
9. Create *harvardArt.js* file in *data* folder and copy contents of seed file data.
10. Import `harvardArt` and `console.log` it.

(1) Create a new React project called `art-museum` using `create-react-app` with the template `@appacademy/react-v17` and using `npm`:

```
npx create-react-app art-museum --template @appacademy/react-v17 --use-npm
```

This command may take a while to run because it should automatically run `npm install` for you. Take the time to read ahead while this is running.

(2) Once the above command finishes, `cd art-museum` and then `npm install react-router-dom` to install the React Router library.

(3) Start your development server by running:

```
npm start
```

You should see "Hello from App" when you navigate to http://localhost:3000.

`Root` component

Now you'll format the entry file for the React application to render a `<Root>` component instead of the `<App>` component.

(4) In your entry file, *src/index.js*, create a functional component called `Root`. Return `<h1>Hello from Root</h1>` from this component.

(5) In the `ReactDOM.render` function, render the `<Root>` component instead of the `<App>` component. `ReactDOM.render` takes in two arguments: The first is a React element, and the second should be an actual HTML DOM element to nest the rendered React elements. Remember, React elements are **not real** HTML DOM elements. In the background, React takes the rendered React elements and turns them into actual HTML DOM elements.

If you refresh at http://localhost:3000, you should see the text, "Hello from Root" instead of "Hello from App". Take a look at the HTML elements tree in your browser's development tools (under the "Elements" tab). You should see an `h1` tag with the text "Hello from Root" underneath a `div` with an `id` of `"root"`. This `div` is the element selected to render the `<Root>` component when `ReactDOM.render` was invoked.

The `h1` element is the actual element created by React when rendering the `<Root>` component.

(6) Instead of rendering the `<h1>Hello from Root</h1>` inside of the `Root` component, render the imported `App` component. Now if you refresh at http://localhost:3000, you should see "Hello from App" again, **not** "Hello from Root".

The purpose for the `Root` component is to wrap the `App` component with any **Providers** that can give your React app more functionality. `BrowserRouter` from the React Router library is a provider you'll be using in all your App Academy React projects to simulate navigation in a single page app.

(7) In your *index.js* file, import `BrowserRouter` from `react-router-dom` and wrap the `App` component (found in your `<Root>` component) with the `BrowserRouter` component.

Your `Root` component should now look like this:

```
function Root() {
  return (
    <BrowserRouter>
      <App />
    </BrowserRouter>
  );
}
```

If you refresh at http://localhost:3000, you should still see the text, "Hello from App".

Art Gallery Data

(8) Make a folder in *src* called *data*.
(9) Create a file to contain the art data inside the newly created `data` folder, the file should be called ***harvardArt.js***. You can run the following command from the root of your project, in the *art-museum* directory.

```
curl https://appacademy-open-assets.s3-us-west-1.amazonaws.com/Modular-Curriculum/content/react-redux/topics/intro-to-react/projects/art-museum/harvardArt.js > src/data/harvardArt.js
```

Take a look at the imported data. Note the structure of the object and the export statement at the bottom of the file!

(10) Import the exported object from this file into *src/App.js* and name the object `harvardArt`. `console.log` the `harvardArt` object. Go to http://localhost:3000 and open the browser's Dev Tools console. There you should see the printed `harvardArt` object. The `records` key in that object is an array of information on Harvard's art galleries. The `objects` key in each art gallery is an array of information on the gallery's art pieces. You'll be using this data throughout this project, so get familiar with the structure of this data!
You can remove the `console.log`.

Phase 1: GalleryNavigation

Here's a breakdown of the steps you'll be taking in this phase (more detailed instruction below):

1. Create *src/components* folder.
2. Create *src/components/GalleryNavigation* folder and add *index.js* file to this folder.
3. Create `<GalleryNavigation>` component.
4. Export `<GalleryNavigation>` and include it inside of `<App>`.
5. Pass `harvardArt.records` to `<GalleryNavigation>` as prop `galleries`.
6. Receive `galleries` prop (using destructuring) to `<GalleryNavigation>` component.
7. Investigate (with `debugger` and/or `console.log`) the value of `galleries` prop.
8. Add `<NavLink>` to `<GalleryNavigation>` component.
9. Add `<nav>` element to `<GalleryNavigation>` to contain other elements.
10. Create component containing `<NavLink>` for each gallery object.
11. Create *src/components/GalleryNavigation/GalleryNavigation.css*.
12. Make CSS styles for `.active` elements.
13. Import *GalleryNavigation.css* in `<GalleryNavigation>` component file.

The first React component you'll be creating and rendering in the `App` component is the `GalleryNavigation` component. This component should be rendered at every route in the application. It will render links to detail pages for every art gallery.

(1) Make a *components* folder in *src*. This folder will hold all your React components besides `App` and `Root`.
(2) Make a folder called *GalleryNavigation* in the *components* folder with an *index.js* file.
(3) In this file, define a React functional component named `GalleryNavigation`. Render an `h1` element with the text "Galleries".
(4) Export the component from the file (using `export default`). Import the component into *App.js* and render it instead of `<h1>Hello from App</h1>`.
Refresh http://localhost:3000. If you see "Galleries" displayed on the page, then you successfully rendered a new component in `App`!
The `GalleryNavigation` component needs to have access to the names and ids of the galleries. The galleries data is in the *App.js* file, and the best way to pass in that data into the `GalleryNavigation` component is through its props.
(5) From *App.js*, pass in a `galleries` prop into the `GalleryNavigation` component with the value of `harvardArt.records`.
(6) Destructure `galleries` from the props of `GalleryNavigation`, i.e., the first argument of `GalleryNavigation`.
(7) Put a `debugger` or `console.log(galleries)` at the top of the `GalleryNavigation` component to test if you passed down the `galleries` prop correctly. Refresh http://localhost:3000. When you open up your browser's dev tools console, `galleries` in `GalleryNavigation` should be an array of art galleries.
(8) Add a `NavLink` to the `GalleryNavigation` component that directs users to the home page, `/` route. The `<NavLink>` can be imported from the `react-router-dom` library. Check out its documentation!
(9) Since there are now two components in your `<GalleryNavigation>` component, you should wrap the `<h1>` and `<NavLink>` components in a `<nav>` (lowercase n!).
A `<div>` would also work, but we recommend `<nav>` for semantic reasons.
(10) Next, from each element in the array of art galleries, create a `NavLink` component that will direct the users to a `/galleries/:galleryId` route where `:galleryId` is replaced with the art gallery's id. The text inside of the `NavLink` should be the gallery's name.

Tips for Step 10:

- Remember to use your instructional resources! Raise your hand on Progress Tracker!
- You want to make use of the `galleries` array you have as a prop in `<GalleryNavigation>`.
- Putting curly braces `{}` in your JSX templates lets you execute any JavaScript you want.
- The `to` property for each `<NavLink>` should be `/galleries/[insert the gallery.id here]`.
- BIG HINT: You can use `.map` off of your `galleries` array to create a React component out of each member of the `galleries` array.

This will be one of your first brushes with the React key prop. You will need to provide a key prop whenever you create components using `.map`. Usually you will provide the `.id` of your data object (or some other unique attribute).

Make sure that there is a link to the home page and links to each of the galleries in the navigation bar.

`NavLink`'s active class

`NavLink`s make it easy to show when a link's path matches the current route. You are going to take advantage of this capability to boldface the `NavLink`'s text for the current route.
(11) Create a *GalleryNavigation.css* file in the *GalleryNavigation* folder.
(12) Add styling to boldface elements with the class "active", i.e., `.active`. `.active` is the default active class for `react-router-dom`'s `<NavLink>` elements.
(13) Import the *GalleryNavigation.css* file at the top of the `GalleryNavigation` component file. (See *src/index.js* to see how to import CSS files into a `create-react-app` JavaScript file.)
In the browser, you should see the active route in boldface in the navigation bar. Make sure that the `NavLink` to the home page is bolded only at the `/` route.

Phase 2: Gallery View

Here's a breakdown of the steps you'll be taking in this phase (more detailed instruction below):

1. Create `<GalleryView>` component with placeholder content (`<h1>`).
2. Add `<GalleryView>` to `<App>`, wrapped with a `<Route>` from React Router.
3. Use `useParams` to access the gallery ID value from the URL.
4. Pass `harvardArt.records` to `<GalleryView>` as the `galleries` prop.
5. Receive the `galleries` prop in `<GalleryView>` and use the gallery ID to find correct gallery.
6. Render header with the gallery name.

It's time to make the component that renders the `/galleries/:galleryId` route and shows the details about the specific gallery with the matching `galleryId` in the URL parameter.
(1) Create a folder in *components* called *GalleryView* with an *index.js* file. Make a functional component called `GalleryView` that renders `<h1>Hello from GalleryView</h1>` and export the component as default.
(2) Import the `GalleryView` component into `App` and render at the `/galleries/:galleryId` route. You should use the `<Route>` component from `react-router-dom`. Always remember to use documentation!

Tips for Step 2:

- Remember your `<NavLink>`s in `<GalleryNavigation>`? This `<Route>` is where they will lead!
- You will need to use both the `<Route>` and `<GalleryView>` components in `App.js`.
- Now that another element is being added to `App.js`, you will need to wrap `<GalleryNavigation>` and `<Route>` in something. We use `<div>` with a `className` "page-wrapper".

- The `path` prop of the `<Route>` component works a lot like the definition of an Express route!
- Submit a Progress Tracker question if you get stuck!

Click on a link to a gallery in the navigation bar and you should see the text, "Hello from GalleryView".

(3) In the `GalleryView` component, extract the matched value for the `galleryId` URL parameter with the `useParams` hook from React Router. Print this value and make sure it looks correct in the browser's Dev Tools console. See how far you can get with just the documentation. Use Progress Tracker if you are stuck! Again: the param you are looking for is `galleryId`! (You set the parameter's name in the `Route` path that you specified in *App.js*.)
(4) To get the information on the art gallery with the specified `galleryId`, the `<GalleryView>` component needs to find the art gallery from the list of art galleries. The data for art galleries live in the `App` component. Similarly to how you passed the `galleries` information into the `GalleryNavigation` component, now pass the `galleries` information into the `GalleryView` component.
(5) In the `<GalleryView>` component, extract the gallery with the specified `galleryId` from the list of galleries in the `galleries` prop. Confirm that it is the desired gallery by using the Dev Tools (either `debugger` or using `console.log`).

Tips for Step 4 and 5:

- Refer to how your `<GalleryNavigation>` component is passed and receives its `galleries` prop. It should be pretty similar here!
- There are lots of ways to find a value in an array, but we highly suggest `.find`!
- Be careful with the data type of your ID! Make sure you are not comparing numbers to strings that merely resemble numbers!!

(6) Next, render the name of the specified gallery in `h2` tags. Make sure that the name changes whenever you navigate to a different gallery from the navigation bar.

Phase 3: `Switch`

Here's a breakdown of the steps you'll be taking in this phase (more detailed instruction below):

1. Create `<Route>` in *App.js* for the home page.
2. Create `<Route>` in *App.js* for 404 errors/unknown paths.

Make two more routes in the `App` component:
(1) One for the home page at `/` and
(2) One for "Page Not Found" 404 errors.
The home page should display a `<h2>` with the text "Harvard Art Museum" and a description (`<p>`) with the text "Look, but Don't Touch. Please select a Gallery in the navigation bar."
And one more for route for displaying an `<h2>` "Page Not Found" when none of the routes in `App` matches the current path. To make the final "Page Not Found" route, you

will need to utilize React Router's `Switch` component. Nice to have some decent documentation, isn't it?
Test this out by navigating to a route like `/not-found`. You should see "Page Not Found" and not "Harvard Art Museum". At `/galleries/:galleryId`, you should see the name of the gallery and not "Harvard Art Museum".

Phase 4: Gallery Images

Here's a breakdown of the steps you'll be taking in this phase (more detailed instruction below):

1. Create `<ArtImageTile>` component.
2. Iterate over `gallery.objects` to create `<ArtImageTile art={art}>` per art.
3. Create the contents (`<Link>`s and `<img>`s) in `<ArtImageTile>`.
4. Restrict the visibility of all arts in `<GalleryView>` using a `<Route>` with URL `/galleries/:galleryId`.

Let's populate the `GalleryView` with images of the art in the displayed gallery.
(1) Add a folder in the *components* folder called *ArtImageTile* and make a functional component that renders the first image of the art in the gallery.
Take a look at the information on the gallery object in `GalleryView`. Is there any information inside of it for artwork images? Each artwork should have an array of images.
(2) From the `GalleryView` component, map the array of artworks to an array of `ArtImageTile`'s. Pass in the artwork information into the `ArtImageTile` as the prop `art`.

Tips for Step 2:

- Spoiler-free hint: 89,111,117,32,115,104,111,117,108,100,32,108,111,111,112,32,116,104,114,111,117,103,104,32,96,103,97,108,108,101,114,121,46,111,98,106,101,99,116,115,96,46,32,58,41<- use `String.fromCharCode` (look up on MDN) to decrypt this hint.

(3) In the `ArtImageTile` component, render the first image of the artwork wrapped in a `Link` component from React Router. The `Link` should direct users to `/galleries/:galleryId/art/:artId` when clicked. This will be how we navigate to another component we build in Phase 5! `:galleryId` and `:artId` should be replaced with the id of the current art gallery and the image's art id.

Tips for Step 3:

- `<Link>` is a lot like `<NavLink>`. Use the docs!
- Put an `<img>` tag in the `<Link>` component and give it the usual HTML attributes like `src`.

- What should be used as `:artId` can be found on the object passed in as the `art` prop to the `<ArtImageTile>` component. You want to go to there!
- You can get the value to use as `:galleryId` by passing down another prop to `<ArtImageTile>`. You will also be able to use `useParams` again after you set up the correct routing in the next step.

(4) An `<ArtImageTile>` for each object in your gallery should be rendered at `/galleries/:galleryId`. To do this, create a `<Route>` component in your `<GalleryView>` component that creates an `<ArtImageTile>` component for each of the `gallery.objects`.

Tips for Step 4:

- You should already have this mapping of `gallery.objects` to `<ArtImageTile>` components from Step 2.
- Try to move that code into a route so it only shows when the URL is EXACTLY `/galleries/:galleryId`

Test out your code by navigating to an art gallery from the navigation bar. The images for each of the artwork in that art gallery should be rendered. If you click on one of the images, the path should reflect the `/galleries/:galleryId/art/:artId` pattern.

Phase 5: Gallery Art Details

Here's a breakdown of the steps you'll be taking in this phase (more detailed instruction below):

1. Create `<ArtDescription>` component.
2. Use a `<Route>` in `<GalleryView>` component to render `<ArtDescription>` at the correct route.
3. Add a `<Link>` to the art's gallery in `<ArtDescription>`.
4. Render a link to the Harvard Art Museum page included in the art object.
5. Render more information about the artwork however you choose.

(1) The last component you will create is the `<ArtDescription>` component. Set up the component file in the *components* folder just as you did for the other components you created.
(2) This component will be rendered in the `GalleryView` component at the `/galleries/:galleryId/art/:artId` route. It will provide details on the artwork at the specified `:artId` path parameter. You should also pass the `gallery` down as a prop to `<ArtDescription>`.
(3) In the `<ArtDescription>` component, render a `<Link>` back to the `/galleries/:galleryId` page. Make the text of this `<Link>` "Back to Gallery". You can also include the name of the gallery in this text.
(4) Render the title of the artwork wrapped in an external link to the actual artwork page in the Harvard Art Museum.
(5) Render all the images of the artwork, the description, credit, and technique.

Tips for Steps 1-4

- You have already done everything you need to complete these steps before!
- Reference your other files to identify the patterns you need and why.
- Ask for help on Progress Tracker if you get stuck!!
- The `<Route>` you set up that will render `<ArtDescription>` should be a sibling component to the other `<Route>` in `<GalleryView>`, i.e., the one you created in step 4 of the previous phase.
- `<ArtDescription>` will know which art to use using the `:artId` param. Get the `:artId` with `useParams` after your route is set up with the correct path. Use the ID to find the right art on `gallery.objects`.
- Look at the art object closely for all the information you could visualize. Especially that `.images` array. `.map` over that!

Test the component out by navigating to one of the art galleries from the navigation bar and clicking on one of the artworks. It should direct you to a path that matches the `/galleries/:galleryId/art/:artId` pattern.

Wrap up

Great job, you worked hard! Be proud of your progress today. Take a mental break to scan through the beautiful art. CSS is an important skill that will be used throughout all your portfolio projects. If you have time left, style your project!

Bonus: CSS

CSS is an important part of any web application project. Take some time to style your project! Add cool hover effects with tooltips or animations if you want more of a challenge!

Did you find this lesson helpful?
No
Yes
✓
Archive your file into a **.zip** folder and click **Submit Project** to upload.

Solutions become available after uploading.