AppReact Router Navigation

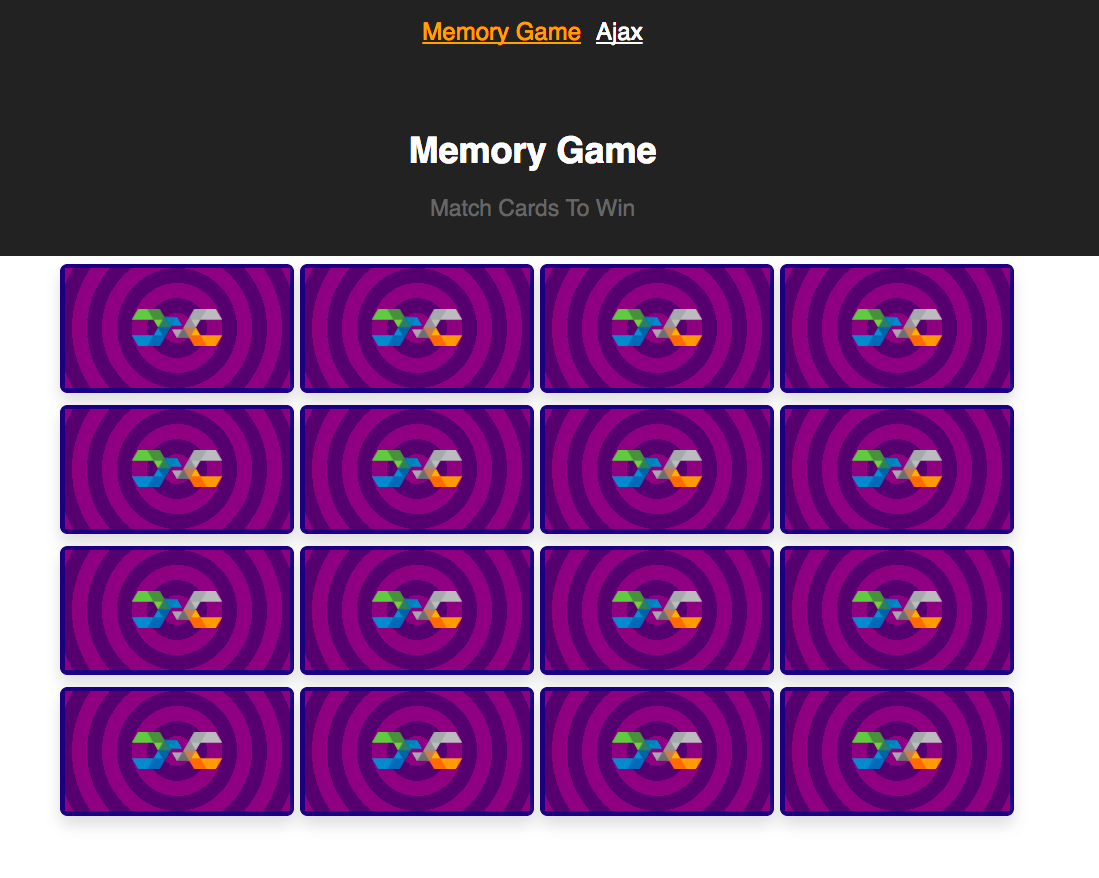
### Objective

We’re going to build upon the Memory Game application we wrote last week (Don’t worry if you haven’t finished it - As long as your memory game app renders without any errors, you’ll be able to do this exercise). We’ll be creating a navigation bar for our application, making it so that the memory game is only one of several pages you can visit.

To accomplish this, we’ll make use of a library called “react-router” - It provides a <Route/> component that you can use to conditionally render components based on the url in the address bar.

### Your task :

When you finish this exercise, your app should look something like this:



Notice that there’s a navbar at the top of the application! Clicking on “Memory Game” will take you to the memory game you built, and clicking on “Ajax” will take you to the project you’ll be working on next week.

### Let’s do it!

# Step 1 - Install react-router-dom

React-router is a separate library that extends the functionality of React, so naturally we need to use npm to install it

**npm install react-router-dom --save**

React router used to be called just “react-router” but ever since they updated to version 4, now the package is called react-router-dom. ¯\\_(ツ)\_/¯

# Step 2- Rename “App” to “MemoryGame”

First things first, we need to rename our top level component from “App” to “MemoryGame”. Our application is going to have more functionality than just the memory game now, so we’ll be creating a new App component later that makes use of our memory game component.

1. Rename the file “App.js” to “MemoryGame.js”
2. Rename the file “App.css” to “MemoryGame.css”
3. In MemoryGame.js, change “**import ‘./App.css’**” to “**import ‘./MemoryGame.css’**”
4. In MemoryGame.js, change “**class App extends …..**” to “**class MemoryGame extends ….**”
5. In MemoryGame.js change “**export default App;**” to “**export default MemoryGame;**”

You should be getting an error at this point - that’s because **index.js** is still trying to render <App />. Let’s create a new App component for our app to render.

1. Create a new “App.js”
   1. Fill it with all the basic React code! Make it export an App component that renders <h1>App</h1>
      1. Import React and { Component }
      2. Define the App class
      3. Extend Component
      4. Define the render function
      5. Make it return some JSX
      6. Export default App at the bottom
2. Create a new “App.css”
   1. Leave it empty for now
   2. Make sure App.js imports “./App.css”

At this point, your application should just be rendering the word **App.** No more memory game!

Alrighty, finally let’s create that “Ajax” component you see referenced in the screenshot above. It’s going to be a simple component that just renders text that says “Ajax”. Next class, you’ll expand on this component and turn it into a cool Ajax tool!

1. Create a new “Ajax.js”
   1. Fill it with all the basic React code! Make it export an Ajax component that renders <h1>Ajax</h1>

# Step 3- Bring react-router-dom into your project

Now we’re ready to introduce react-router!

To put it simply, react-router lets your application decide which components should display when. If, for example, your app has “home” page and a “faq” page that you’ve built using react, you would only want to show the HomeComponent when the user has navigated to “yoursite.com/home”, and you would only want to show the FAQComponent when the user navigates to “yoursite.com/faq”

React-router provides a neat way to handle this functionality, but first, it needs to wrap your entire application:

1. In **index.js**, let’s import react-router
   1. **import { BrowserRouter } from ‘react-router-dom’;**
   2. BrowserRouter is a custom react component that listens to changes in the browser’s url. It’s provided to you inside the ‘react-router-dom’ package.
2. Wrap the **<App />** tag in the render() function with **<BrowserRouter></BrowserRouter>**

That’s it! Now react-router is ready to conditionally render your components.

# 

# 

# 

# Step 4- Set up the application nav bar

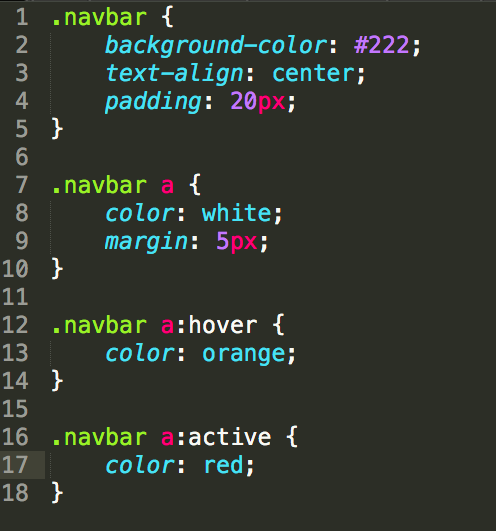
Let’s revisit **App.js**!Right now, it’s just rendering the word “App”. We’re going to make it render a react-router navbar instead.

1. Instead of rendering **<h1>App</h2>,** render a single <div></div>
   1. Give this div a className of “navbar”
2. At the top of app.js, let’s bring in a react-router Link
   1. **import { Link } from ‘react-router-dom’;**
3. Inside of the navbar div, create two Links that contain the text “Memory Game” and “Ajax” respectively
   1. <Link>Memory Game</Link>
   2. <Link>Ajax</Link>
4. Give each link an attribute called “to” and set them equal to “/memory” and “/ajax” respectively
   1. Remember, attributes go inside the opening tag! They look like <Tag attribute=”value”></Tag>

The <Link> component is actually just an <a> tag , but it’s actually hooked up to react-router so that it can trigger its conditional rendering.

At this point, you should have two links showing up on the screen. Notice that when you click on one, the url changes in your browser! Pretty neat, huh? Let’s make it a little bit prettier.

1. In App.css, add these styles:



# Step 5- Creating routes for conditional rendering

<Link> tags are what trigger new routes in your browser, but you still need to write code that chooses which components to render for which route. That’s where the <Route> tag comes in.

1. In the **import { Link } from ‘react-router-dom’;** line, let’s also pull in the Route component
   1. **Import { Link, Route } from ‘react-router-dom’;**
2. While you’re at it, import in the MemoryGame and Ajax components
   1. Import MemoryGame from ‘./MemoryGame.js’
   2. Import Ajax from ‘./Ajax.js’
3. Right after the navbar in the render() function, add two self-closing Route tags
   1. <Route />
   2. <Route />
4. At this point, react will complain that you have too many top level components. Wrap everything with a simple <div> tag to fix it!

Alright, we’re almost there.

Routes need two properties to behave how we want them to: **path** and **component**. The **path** property tells it what browser url to look for when deciding whether it should render, and **component** tells it what component it should render.

1. Add the **path** and **component** attributes to both of the Route tags
2. Set the first path equal to **“/memory”** and set the second path equal to **“/ajax”**
3. Set the first component equal to **{MemoryGame}** and set the second component equal to **{Ajax}**

That’s all it takes! Now your app should render either component based on whether you visit /ajax or /memory.

If you want your app to render the Memory Game at the homepage, just add another Route tag that has a path equal to “/” and points to the MemoryGame component. Wrap all your routes with a <Switch> tag that you import from react-router-dom