CS410/510 Final Project Journal

April 20th
Today I began to translate my project proposal mockups into actual code. This is the first time I've done so outside of work, and I think it's helped quite a lot. The hardest thing to me is generally mid-to-high level CSS: how am I going to ensure the width of this element? How am I going to center this vertically? If you search "center vertically" on StackOverflow, you'll find several thousand results. I made the executive decision to use FlexBox CSS in most places, as I think it provides the simplest implementation for what I plan to get done.

April 26th
The FlexBox decision wasn't ideal for the navigation bar. I decided to go with Bootstrap here to get the weird 5/12ths - 2/12ths - 5/12ths setup I wanted. I implemented a dropdown menu, partly because I wanted the practice doing so in React. My approach in React tends to be hard coding a new div as part of a higher-level component, then moving its state, functions, and CSS to a lower-level component later. Here it works well, but I'm not sure if that's counter productive in more complicated scenarios.

May 11th
I've done dropdown menus in vanilla JS before, but never in React. This would end up being the most challenging part of the entire project, and I ended up eventually scrapping it. Getting the dropdown menu to *drop* was pretty easy, but I couldn't figure out how to get it to disappear when my mouse left the menu. There must be some strange interaction among React, the DOM, and the mouse location with an absolutely positioned element. My issue was that if I moused out of the parent/title element, then the dropdown menu would disappear, *even if* I was moving my mouse onto the dropdown menu. This was a problem, but I couldn't get it to work smoothly. My workaround made it fairly difficult to *close* the dropdown menu, which made the entire website feel too clunky. I decided the dropdown section wasn't completely necessary for the functionality of the site.

May 20th
I started to implement React-Router. In CS465/565, I thought this was the most challenging part of the final project for that course. After that term and several months working with Spring Framework APIs at work, this became much more intuitive. I implemented static URLs for each of the main four sections: Rankings, Conferences, Historical Records, and Team.

May 23rd
One thing I'd been missing which was quite important was a home page component. I'm not sure what to put here so I just have a search bar, for which I haven't implemented any functionality yet. I'm hoping to have a modern "Google"-style search bar, where I can display predictions (probably just using regex), and link to specific team pages.

May 27th
Historic records involve biting off way too much. This is an example of me deciding to implement something in my proposal that was not realistic. The API I'm using is very solid, but attempting to produce historic records might be 1) uninteresting, and 2) require several API calls. I replaced this whole idea with schedules.

May 29th
Up until this point, I had essentially only done framework work and CSS. I had a solid skeleton of a site but no API calls. Today, I did quite a lot of work: an API call to get rankings, and a way to toggle

between the two types of ranking polls in college football. One challenge to run API calls arose because I elected to use only stateless, functional components, to be a bit more modern in my use of React. I created API calls for team data and conference data, but I didn't implement any corresponding React yet.

May 31st

Today was my most challenging day. I created a Teams page that had three relatively challenging aspects. The Teams page would consist of links to all 130 FBS (Division I) football teams. I elected to render them as a series of squares with team abbreviations; this might not be the most clear to some but as a huge football geek it looks good to me.

First, I wanted to make it easy to find a given team on the page. There are 130 teams so it could easily become very challenging. I made it possible to highlight all teams belonging to a given conference, and I made it possible to sort the entire block of teams. This was challenging primarily since the JS Array.prototype.sort() function isn't very intuitive. One has to make sure that each of the elements being sorted is of the correct data type, and it sorts using a boolean expression, which is also a bit weird.

Second, I wanted to display the team colors in a cool way. I needed to pass the colors (obtainable via the API) from the main team page "engine" down to the details page, but also all the way up to the NavBar implemented at the start of the project. I needed to add a lifecycle hook to the NavBar and then pass in the setState function all the way down to the leaves of the component tree.

Third, I needed to make the strange color combinations accessible. Several schools have questionable color decisions like Colorado's tan and yellow combination (the API excluded their third usual color, dark brown, for some reason). I wrote a helper function that detected whether or not two colors in 3-dimensional RGB space were sufficiently close to one another using Euclidean distance. This is very easy in C/C++, much less so in JS.

June 2nd

Today I implemented the schedule and team details pages. I created a new SearchBar component that simply redirected to a team detail page. With the detail pages, I had two main problems.

First, I had to use dynamic routing, so I didn't have to create 130 different pages. It was a bit challenging to work this using ReactRouter, but I did have experience implementing this in Express for 465/565 and in Spring for work.

Second, several teams have team colors that don't work with the usual black text in the NavBar. I wrote another accessibility helper function that detects how dark a team's color is (just averaging the RGB values), and then subjectively changes the NavBar text from black to white if it exceeds a certain threshold.

Regrets and Thoughts:

1. I wish I had used Redux to handle state for my application. First, it was challenging to pass state from one end of my component tree to another to get the team details color system working correctly. Second, this could have been a great first exposure to an important, relatively new technology.
2. I should have made more detailed mockups in my proposal. I thought of a lot of features on the fly, which worked for a relatively small project like this, but it would have been nicer to have a bit more constraints when actually in the development stage.
3. I should have made accessibility more of a priority early. It's not too challenging to retrofit a small website like this with accessibility features, but I think it sets a bad precedent. Moving forward, I'm going to focus more on this.
4. I was surprised by how fast I was able to code this. I had back-loaded a fair bit of the project, nothing too extreme, but I was able to get things done quickly. I've definitely become more proficient at

React since fall term with 4/565, which is a really great feeling. The depth of my proposal also contributed to this.