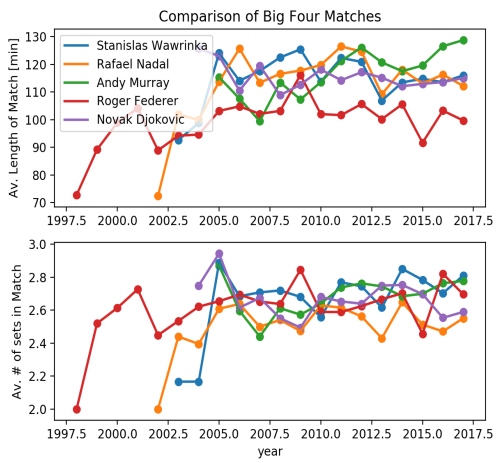
Tennis Project

Motivation:

I wanted to practice on some of my JS skills because I really enjoyed my time making my personal webpage and I wanted to do another web-based project, one that is personal to me.

I had previously done a python-based statistical analysis of Federer, Nadal, Djokovic, Murray and Wawrinka to impress some people in the /r/tennis subreddit. It was very simple, all it did was look at the average match-length and number of sets in all of their careers. I cloned a Github Repo and parsed through all the csv files with python. In the end, I ended up generating this poorly formatted plot:



Web Scraper:

I'm starting out making a simple web scraper in javascript. This webscraper scrapes the ATP World Tour Site for a pre-defined set of big names:

```
    const mapping = {
            'Roger Federer': 'https://www.atpworldtour.com/en/players/roger-federer/f324/rankings-history',
            'Nick Kyrgios': 'https://www.atpworldtour.com/es/players/nick-kyrgios/ke17/rankings-history',
            'Rafael Nadal': 'https://www.atpworldtour.com/en/players/rafael-nadal/n409/rankings-history',
            'Novak Djokovic': 'https://www.atpworldtour.com/en/players/novak-djokovic/d643/rankings-
history',
            'Andy Murray': 'https://www.atpworldtour.com/en/players/andy-murray/mc10/rankings-history'
        };
```

As you can see, I have the URLs that will be parsed. Later, I plan on adding more to the scraper that would let you choose the player. I'm thinking right now it would follow the following:

- ⟹ Use puppeteer to crawl the ATP main page, then search the player
  - ⟹ If the player is there then grab the base url and proceed to the second part
  - ⟹ I not, use error-handling to make sure that the user knows the player is not in the ATP database (or they spelled the name wrong)
- ⟹ The second part uses CheerioJS and request as well as an off-shoot (request-promise) to load html, parse the html, and return an asynchronous Promise.

As of now, the scraper returns an object that looks like this (this is for Andy Murray):

```
{
'2018/06/18': 22,
'2018/06/11': 21,
'2018/05/28': 22,
'2018/05/21': 22,
'2018/05/14': 18,
'2018/05/07': 12,
'2018/04/30': 12,
'2018/04/23': 12,
'2018/04/16': 13,
'2018/04/09': 13,
'2018/04/02': 13,
'2018/03/19': 12,
'2018/03/05': 13,
'2018/02/26': 13,
'2018/02/19': 14,
'2018/02/12': 14,
'2018/02/05': 13,
'2018/01/29': 13,
'2018/01/15': 14
}
```

One thing that I do have to consider is how configure the eventual react ⚛ to be able to use nodeJS' JS `require` module feature. I am currently reading this guide. It turns out that, although `require` is very popular and thouroughly used in nodeJS, it is not native to most browsers (yet). Therefore, I need to configure *Babel* (the compiler) and *Webpack* (the file bundler) to make use of it.

UI

As mentioned, this will be a react ⚛ application. The reason I chose this is for two reasons:

1. The last applications I built [My Personal Webpage(https://sammy-alhashemi.herokuapp.com/), which was an angular-express A app, so I wanted to do some react ⚛ again.
2. React ⚛ can produce nice modular, code that is easy to read and understand what is happening. The component lifecycle is also intuitive.

I started writing the current UI on stackblitz just for convinience sake. Also because it lets you download the code later. As of now, I have four main components:

- ⟹ The `Navbar` at the top
  - ⟹ displays the Players names I want to analyze (RF, RN, ND, AN)
  - ⟹ Working on styling to give a modern look.
- ⟹ The `Body`
  - ⟹ Currently styled with a `linear-gradient` background:

```
background: 'linear-gradient(to bottom, #1e5799 0%,#2989d8 48%,#207cca 73%,#7db9e8 100%)'
```

- ⟹ Contained in the body is a `Paragraph` component that is very simple → just contains words styled with white font
- ⟹ Also contained in the body is the `Graph` component.
  - ⟹ This uses ChartistJS to place a graph
  - ⟹ To be specific, I used a react ⚛ Component someone created called react-chartist
  - ⟹ Eventually, this graph will display the data gathered from the Webscraper.

As of now, the UI looks like this: