Table Tennis Tournement App

This write-up will detail some of the design and technology choices I made for my DevelopMe technical challenge.

The final product is a desktop app which can take 4, 8 or 16 players. The app renders a responsive tournament structure with a set of randomised first matches. The user can click on players to mark them as winners, this also passes them onto the next subsequent round in the tournament structure.

Technology

I decided to answer the brief with a web-app for desktop using React as my front end framework and Redux for state management within the app.
I chose these for a multitude of reasons:

- I wanted the app to be expandable and reactive, the state management in Redux lends itself very well to this. It makes it easy to add functionality, while minimising bi-products that would otherwise occur if I were to use something like jQuery.
- Create-React-App automatically installs babel giving me access to ES6 syntax and its shorthand notation.
- Immutable.js is easily integrated and offsets the downside of having to deal with state mutations in Redux.
- I wanted to push myself in this tech challenge, in order to produce a responsive structure it made sense to use Redux. This meant that once I got the basic app working, it was a-lot easier to add extra functionality e.g. marking winners, future rounds.
- React has hot reloading, making my development workflow much faster.
- Both React and Redux come with tool plugins for Google Chrome, making them easy to develop with on a desktop browser.

I also used github for my version control, and used a project board to set myself issues and to-dos. I found it much easier to focus on individual bits of functionality once I started using the card son the project board of my repo. You can see my repo project board here -

https://github.com/Acparvis/table-tennis-tournament-app/projects/1

This also provides a history for myself where I can review and tweak my workflow in future.

Design

There are several design decisions that I made that are worth mentioning.

The random sort algorithm.

Originally, I used the Fisher-Yates shuffle to make my random player matchups for my tournament. This however, left me with a dilemma. In order to implement the shuffle, I would have to convert the players from an immutable list, to a JS list using the fromJS() function. It seemed very long winded to shuffle the list this way, as the array would have to be converted back to an immutable list afterwards before it could be passed into the state again. Instead I decided to sort the players within the immutable list using the following method-

```
player.sortBy(Math.random);
```
(player_list.jsx, line 11)

I was initially reticent to use this sort method as it is a ‘bubble sort’ and therefore not memory efficient. However, the player numbers are not that large and the app does not run slowly (I tried putting in 128 players and the app didn’t seem to have any noticeable latency). So for the sake of not having to convert to and from JS I felt it was fine to use this method.

State Structure / Player Number

My original plan was to make an app with a responsive tournament structure, that could also take any number of players. Early iterations of my app checked the number of players and .pop() removed the last one after randomising the order. This was to provide a bye-player that could be injected into the next round. In the process of making the round structure, I found it greatly complicated the round structure to have to keep checking the list length and remove/add random players. This would have to be done in real-time, when players were clicked on in the responsive tournament tree. In the end I prioritised having the responsive structure and forwent the flexibility of being able to submit any number of players. Given the time constraints I think this was the best choice, however the app already nearly works with any even number, and can actually already accommodate 32, 64 128 and 256 players. If I remove the player list number restrictions, a user could very nearly use the app for any even number. With a few more days development I think this would be implementable.

Bugs

I feel there are two known bugs worth mentioning too-

1. The app allows the user to submit a player name that contains only spaces - I intend to improve this by either improving the validation on the input, or perhaps by implementing something similar to the trim method in PHP (removes whitespace from the beginning and end of strings).

2. You can retroactively change winners two rounds back and the responsive tournament will not update. This is another bug that given more time I would have been able to fix. The downside to making a responsive structure is that you have a-lot more possibilities and factors to account within the app. I would need to run a check when a winner is named. The check would update any previously set winners in the changed matches lineage.