MEFIT: Technical Overview

Technology stack:

For my solution I decided that it was best to use ExpressJs as the framework, especially since it was my first time making a full stack website.Having a lot of resources is a must especially when troubleshooting and since Express is super popular I didn't have any trouble googling errors and finding solutions immediately.

For my data base I decided on Postgres cause it was easy to use and apparently many employers prefer it, so knowing how to use it was important.

And for the styling I kept it simple, just some good old Css and Javascript were enough to get the job done.

How the ranks were calculated:

One of the challenges was ranking the athletes for each event in a competition, that would mean going through the entire array and comparing them for each event. The main problem was the fact that we had multiple criteria of ranking. Since we have : Main-score, Tie-break,Secondary-score and Secondary-tie-break that meant that we had a 4^4 -1 possible ways to score an event.

To solve that I came up with a way to solve for all the possible ways. First I need to know the way that event is scored, for that I run an sql query to fetch the scoring method as an array. Now I need to start the sorting, luckily in javascript that built-in sort method could be altered to suit our needs. I added the logic so that depending on whether the data is a number or a time stamp it will be compared differently and it does it according to the scoring method array. At the end we get the sorted array of athletes for that particular event.

Now it’s time to give every athlete a rank. We iterate through our sorted array and verify if that athlete was already given a rank or not. We have a variable rank that keeps track of the ranking. We always verify if two athletes have the exact same score for main-score ,tie-break etc.. If they do we give them same score. We keep doing the same thing until we go through the entire array.

Now we have an array or arrays that contain the athletes and their ranks for each event, I decided that it would be easier to calculate the overall rank if all the athletes are in the same order for all events. This will save us time since we won’t have to iterate through the entire arrays looking for the athlete n times in the worst case making it m*n , m being the number of events and n being the number of athletes. Now we can just do a single loop where we add the ranks for every event and give it to the athletes.

Now we have an array of athletes sorted in alphabetical order and every athlete has their overall rank and the event rank. The only thing left is to construct the table.

How the table was constructed:

Now we have all our data set-up all that is left is to turn it from an array of json to an html table for that were gonna use vanilla javascript. The things is we have to make an html table then on each iteration we have to add the following row. Since all of the athletes are sorted alphabetically. We’re assuming that the all the athletes in a competition are participating in the all the events in that competition. That way it will be easy to pair and match the data.

We iterate through our array and we make our rows for the data from each array of athletes. At the end we end up with a simple html looking table with no sorting capabilities .

Now we add the styling, this was optional but I took the liberty of making the website look as nice as possible to improve my front-end skills. After having the styles in place, we add an event listener for each column header to be able to sort the data . We use the same tactic used before. Using the built in javascript sort method we add our own logic. We're adding to functionalities, if you click the first time it will be sorted in an ascending order, click again and its descending order. As simple as that, there's a lot of styling that I skipped since its just some plain old Css.

How the data is kept up to date:

The way I done it is that every time you choose a competition to view its leaderboard the algorithm is ran to fetch the data and construct the table. After doing some analysis I can see that the algorithms implemented are efficient enough to reconstruct the table each time. Rather that waiting for data to be inserted then compared to what we have then altering possibly all the other data, It's a recipe for disaster.

Some optional additions :

The navigation bar was one of the additions that proved to be hard to make. I added to it an animation so that it will change its styling when the user is scrolling I had a problem with the page jumping since I was changing the its position from absolute to fixed . Keeping it fixed from the beginning solved the issue. I added some small sections to give that real effect to the web application. A section that talks a bit about the project. And another for github to view the source code , email and linkedin for contacting me.

Conclusion:

In conclusion the project was a success, It served as a stepping stone for me to web development. I learned a lot of data base design concepts, backend development for making the api to let users insert data. The frontend was the fun and tedious at the same time since css has some weird default settings, so you have to make sure to change them.