1. The initial web architecture

The original web architecture structures: we looked at a lot of templates and common components, like the bootstrap, OwlCarousel, superfish, jquery, and so on. After a long discussion, we decided to use bootstrap to make a unified nav bar. This is not easy, and it involves many problems about CSS. For example, admin and staff accounts have two more buttons than ordinary household accounts, which means that we must add condition judgment in JS. To adjust the correct position of buttons in the Nav bar. As for the design of the bottom bar of each web page, we did not use postition:fixed to fix it, but determine whether the bottom bar needs to be fixed by judging the scrollHeight.

1. Storage and use of user names

The idea of our website is to let users register first, and then use the registered user name and password to log in. After login, it will automatically jump to the home page. When jumping to the home page, the username of the login user should also be displayed next to the login icon.



At first, I planned to get the username from the database, but it was very troublesome, and there was no database at that time, so I discussed with my partner. Finally decided to use the window. The sessionStorage. SetItem (" userName ") way to user name in the local, at the same time, deposit and userType and userId.



After that, when we want to use username or usertype, we can use the code window.sessionStorage.getItem.

Design of the logout button and function：

At first, when the user logout button was clicked, the site only cleared the username in localStorage, which caused a typographical bug in the nav bar after the admin and staff users logged in. Log in as the household user and you will find that the nav bar still has "admin" and "report" buttons (which are unique to the admin user). The initial method is to write some criteria and refresh the page. Later after a discussion on the logout button function adds window. The sessionStorage. RemoveItem (" userType "); And the window. The sessionStorage. RemoveItem (" userID "); That's the perfect solution.

1. Css problem

We had a lot of problems with CSS at the beginning of the project. For example, at the beginning of the home Page, we couldn't arrange the images and text boxes neatly and aesthetically. We usually wrote a lot of code, such as margin-left:30px; This was a bad way to move divs around. Later we used the bootstrap method, such as the "col-MD-8" grid system. This was very helpful, especially on the country list page.

4.My Account page with feature create

This is a dynamic page. Before making it, we considered whether to write.js and.ejs separately, and finally decided to keep the same way as before. (page from js) difficult point lies in the calculation of Carbon footprint savings (kg) must call two interfaces, respectively is url: 'http://localhost:3000/countrylist' and url: 'http://localhost:3000/userPage', because want to calculate the Carbon footprint savings (kg), we must get each user transaction records in the corresponding country country\_id panel\_amount and donations, Then use country\_id to get the corresponding country's carbon\_saving\_factor in countrylist interface, Finally, use panel\_amount \* carbon\_saving\_factor to get the Carbon footprint savings(kg) for each donation record. The sum of the Carbon footprint savings(kg) from all the donations is the sum of the individual user's Carbon footprint savings(kg). Calling both interfaces at the same time is a bit of a problem because we have to run the function getSavingFootprint() to calculate Carbon footprint savings(kg) when the interface call succeeds, but we don't know which interface will succeed first, Our original solution was to add "if(result) getSavingFootprint();" to both calls. But after a group discussion, I decided that using axios was the best way to solve the problem, otherwise when you have to call 3 or more interfaces, it will not solve the problem.

There were also initial challenges in dynamically writing every user's transaction history. Due to my lack of experience, I didn't use jquery to write HTML at the beginning, which led to very redundant code, and it was very easy to make mistakes to write the table one by one. Subsequently, WE easily wrote the historical donation record information we needed by using jquery+for loop.

5.For the use of the “!important;”

When the public component conflicts with the CSS code written by myself, I often use the "!important;" declaration method to make my CSS code take effect, but then I found out that this is not correct, and it took a long time. Time to find why some text or divs on the main page are not styled correctly. So I think:

a. Be sure to use the priority of style rules to solve the problem instead of !important;

b. Only use “!important;” on specific pages that need to override site-wide or external CSS

c. Never use !important in plugins

d. Never use !important in site-wide CSS code