Challenges:

1. Ensure the status is synchronized with the UI

For non-static pages, the biggest challenge is how to dynamically obtain back-end data and user operations, and display them correctly on the page.

For getting back-end data, the front-end interacts with the back-end through the interface to obtain the data stored in the database. We use ajax to call the interface and display the data returned by the interface on the page. The first problem we encountered is that jQuery cannot directly use the for loop in HTML to show the contents that need to traverse the array. Through consulting the materials, we found that we need to use the loop in JS through ID selector to dynamically append the HTML sentences to HTML file. We believe that this inconvenience is due to the framework we have chosen.

Then for getting user action. jQuery is very limited in responding to user operations, mostly responding through onclick events. This response also requires frequent operation of DOM, which makes developers not only need to pay attention to business logic, but also need to focus a lot on operating DOM, which may cause rearrangement and redrawing, and will be accompanied by a huge cost of rendering.

Therefore, we can see that the above two inconvenient problems are caused by not choosing a better framework. We should choose a framework more suitable for front-end development. This not only reduces the workload, but also increases the fault tolerance.

2. Component-based Development

At the beginning of our development, we didn't realize the importance of component-based development. For example, most of our pages have navigation bars at the top and bottom. Instead of writing them in a public HTML file, we add the same codes to each page. This not only increases our development workload and produces many redundant codes, but also makes us need to modify these same codes in each file when making the same changes, which may lead to omissions.

Then we realized this problem in the development and wrote some common components. For example, we added the alert prompt box and toast prompt message using common components. We just need to import these public components on the page and pass in the prompt text. Additionally, we can also add some style controls for the prompt box in different scenarios, although we don’t need to implement it this time.

Similarly, some functions of calling interfaces and methods of calculation can also be written in public JavaScript files. Just import these JS files and we can call the same methods on different pages. It can reduce code redundancy and increase the maintainability of the code.

3. The storage and transfer of JSON objects

On the donation basket page, we need to record the items added or reduced by the user, so that the next time the user enters this page, he or she can view the items previously selected. At the beginning, we intended to transfer the data of basket to the back-end through the interface and store them in the database, so that users can still see the previous choices when logging on other devices. However, we realized that users may frequently add or delete items in the basket, which makes us need to call the interface for every operation of users, which has high requirements for the performance of our platform. For example, if many customers are operating the basket in the meantime, this interface will be highly concurrent called. After discussing with the back-end partners, we decided to temporarily store the user information and basket data in the front-end using window.localstorage and compare it with the user's current login account. When it is the same account, the content in the basket will be displayed, so that the user will see the previous selection when logging in the same account on the same device. We can also choose a better front-end storage method, such as indexdb.

When we transferred the data of basket selected by the user to the back-end, we also encountered another problem of passing JSON objects. The front-end takes the number of each country and solar panels selected by the user as a JSON object, and stores the multiple JSON objects in the array. When the front-end directly passes this array to the back-end, the back end cannot correctly parse the data. Through the console, we found that the data received by the backend is actually in the format of [object, object]. Then we converted the data of each basket into JSON string format at the front-end, and then passed it to the back-end. Then we had a new problem. When there is only one piece of data in the basket, although the front-end converts it into string format and puts it into the array, after passing it to the back end, the syntax parser will directly convert the array containing only one string into string instead of array, so the back end needs to deal with this special case. Finally, we converted the whole basket array into JSON string at the front-end, and then parsed it into JSON object after receiving the passed value at the back-end, the problem was solved.