EzParking Client

Code Quality Report

SSE Capstone project - University of Regina

Team Apollo

Wintern 2023

Zhuo Chen

200352092

Table Of Contents

Summary

The EzParking Client application serves as the user interface for our parking service project, providing a simple and efficient way for users to request information and complete their parking needs. Using the application involves just three steps: first, users select their destination; second, they choose a suitable parking lot from the options provided; and finally, they can start the navigation to guide them to their chosen parking spot. With this streamlined process, our clients can easily and quickly find parking solutions that fit their needs.

The focus of this report is to evaluate the quality of the code implemented in the EzParking client application. The evaluation is conducted from five perspectives, including code formatting, code architecture, performance, security, and overall code quality. By thoroughly examining these aspects of the code, we can gain a comprehensive understanding of the application's code quality and identify areas that may require improvement.

The client application is built on the React.js framework and implemented using the programming language JavaScript. With React.js, the user interface is optimized for high performance, ensuring a smooth and seamless experience for users. JavaScript, on the other hand, provides the necessary flexibility and functionality to create a dynamic and interactive application. Together, these technologies form a powerful and effective foundation for the EzParking client application.

Code formatting

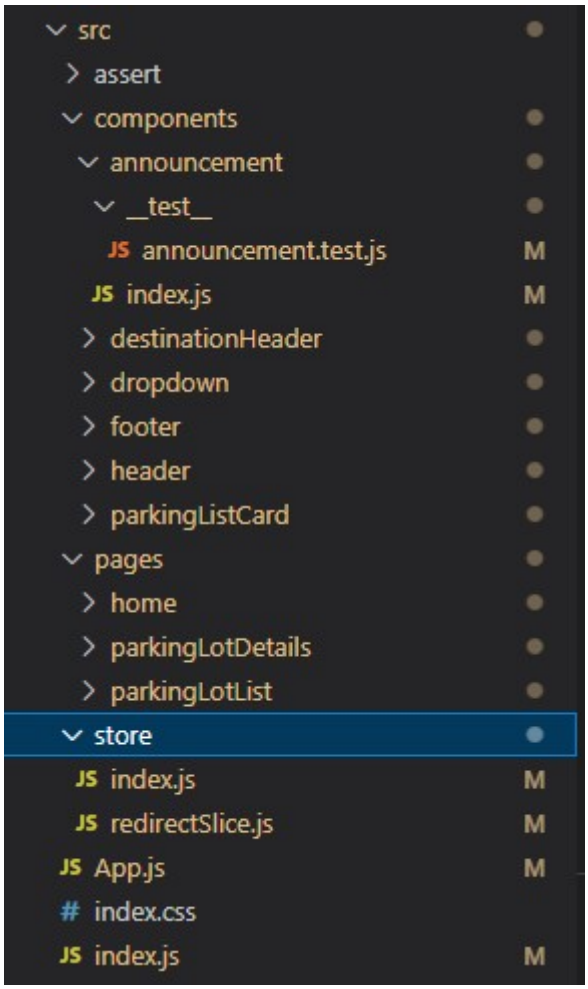
Code formatting is the primary element to improve the readability of the source code. It could prevent developers from disturbing their attention with improper formats. To ensure the code is formatted properly, ‘Prettier’ is applied in the development. It is a VS Code extension that could enforce a consistent style of all the files by following its per-defined rules and rearranging the source code. In this way, developers could hold their attention on analyzing and developing the application.

Code architecture

Since this application is implemented based on the React framework which allows developers to divide an entire application into smaller portions and reuse them properly, the code architecture can be regarded as folder structures.

From a top-down strategy, the application folder contains three main parts in the first level directory, modules and JSON files, the public folder, and the src folder. Modules and JSON files include all the libraries, modules, and information that would be used in the application. The public folder collects items, like customized icon images and public pages. While the src is the core of the application.

Figure 1 Folder Structure



Src consists of six parts, assert, components, pages, store, App and index pages, and CSS file. As shown in the figure. The component folder contains the smallest items of the application that could be rendered repeatedly. Based on the SOLID principles, each component has a test folder to examine the particular component, it is also convenient to be extended in future development. Pages is a folder to hold the integration of all the components. In the client application, three pages need to be rendered. The store is the logic section that includes the variables and functions applied globally. The App and index files are the main place to render the pages accordingly. The last one is a CSS file that contains all the styles used in the src folder

Performance

The dry principle was applied while the client application is being developed. Every component and page is rendered without redundancy. To visualize the performance, the React Profiler is used to produce the exact time performance of each action for the application operation.

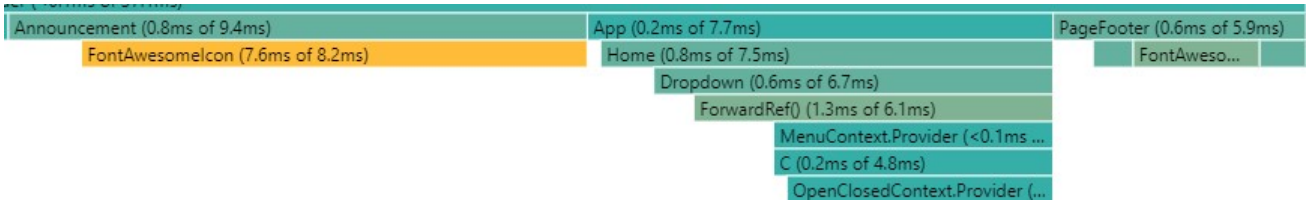


Figure 2 Rendering performance

As shown in Figure 2, the most time-consuming part is rendering the 'FontAwesomeIcon' which took 7.6ms. The time duration for rendering the entire App is 7.7ms. The performance meets our requirements and it improves the user experience.

Security

For the client application, there are two APIs embedded in the source code, one for fetching the destination information, and another used to get parking lot data. Since both the APIs are used to get information from the database and all the fetched data are displayed in the user interface, the APIs are completely unlocked to the public. However, in order to prevent our backend server from a DDoS attack, we generate a token that will be used every time the APIs are called. Once the database receives the request, and the token values are matched, the backend server will start processing the request, otherwise, the frontend will only get an error 400 result.

Overall code quality

A particular branch is created for a specific purpose, like an extension of new features. And to ensure the source code quality before it is merged into the main branch, we assign a pull request to all other team members to review and approve. By

following this workflow and reviewing the source code from the four aspects, the code quality of client application can be significantly improved.