# Group 10

## CSC 8019 Final Group Report

**Product Name**: Char Char

**Group Member:** MeiHui Li, 220155955, YiFan Qian,220084312, Yi-Ting Liang,210438622, Jiahe Li, 190685094, YiLi “Eastman” Luo, 220406293, HaoYu Zhu, 220367846, YiLong Wang, 220160797, **Zheng Liu**

**Date Of Submission:** May 12th2023

**Word Count:** 987

**Page Count:** 3 Pages Excluding Table of Content, and Individual Marking

# Contents



# Table of Figures

## Overview

CharChar is a web application designed to serve as a navigation map for users to pick, filter, and sort tourist attractions based on their preferences. It also provides features for users to browse and add comments about particular places. The application consists of two main components: the frontend, developed using Vue3 with TypeScript, and the backend, developed using Java with the Spring Boot framework. At this stage of development, the entire application is 208 KB in size and consists of approximately 4642 lines of code.

| Extension | Count | Size SUM | Size MIN | Size MAX | Size AVG | Lines | Lines MIN | Lines MAX | Lines AVG | Lines CODE |
|---|---|---|---|---|---|---|---|---|---|---|
| ds_store (DS_STORE files) | 1x | 6kB | 6kB | 6kB | 6kB | 1 | 1 | 1 | 1 | 1 |
| html (HTML files) | 1x | 1kB | 1kB | 1kB | 1kB | 44 | 44 | 44 | 44 | 41 |
| java (Java classes) | 33x | 167kB | 0kB | 40kB | 5kB | 4375 | 13 | 867 | 132 | 2236 |
| md (MD files) | 2x | 0kB | 0kB | 0kB | 0kB | 21 | 2 | 19 | 10 | 10 |
| properties (Java properties | 1x | 1kB | 1kB | 1kB | 1kB | 25 | 25 | 25 | 25 | 20 |
| sql (SQL files) | 3x | 20kB | 1kB | 16kB | 6kB | 177 | 47 | 76 | 59 | 120 |
| xml (XML configuration file) | 4x | 9kB | 0kB | 4kB | 2kB | 245 | 17 | 117 | 61 | 211 |
| Total: | 46x | 208kB | 12kB | 71kB | 24kB | 4888 | 149 | 1149 | 332 | |

*Figure 1 Backend Statistic Overview*

Languages

| language | files | code | comment | blank | total |
|---|---|---|---|---|---|
| JSON | 3 | 13,860 | 0 | 2 | 13,862 |
| Vue | 15 | 1,830 | 85 | 196 | 2,111 |
| TypeScript | 17 | 576 | 147 | 74 | 797 |
| SCSS | 2 | 31 | 0 | 1 | 32 |
| JSON with Comments | 1 | 30 | 12 | 1 | 43 |
| JavaScript | 3 | 26 | 22 | 3 | 51 |
| Markdown | 1 | 19 | 0 | 6 | 25 |
| CSS | 1 | 18 | 1 | 2 | 21 |
| HTML | 1 | 17 | 1 | 1 | 19 |

*Figure 2 Frontend Statistic Overview*

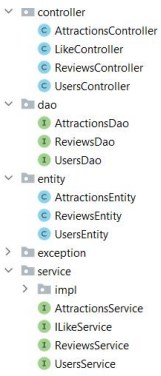


*Figure 3 Backend Spring Boot Overview*

## Backend Development Overview.

The backend handles the logistical side of the application and consists of 2236 lines of code. It takes care of creating, storing, updating, and manipulating sets of data. The backend utilizes the Spring Boot framework, which simplifies the operation of an application by dividing it into controller, service, data access object (DAO), and entity layers. The entity layer defines and manages the attributes of all types of entities, which are later used by the program for information exchange and data storage. The DAO layer handles all operations

related to the database, while the service layer uses this data to perform certain logical operations. The results of these operations are then sent to the controller layer, which communicates with the frontend via HTTP requests. The controller layer also handles input from the frontend in the same way. We also implemented Redis as a cache storage solution to temporarily reduce the response time of the system.

## Frontend Development Overview.

The frontend of the application sends information to the backend for processing and formats the appearance for different platforms using TypeScript, JavaScript, CSS, and HTML. The app can adapt to landscape or portrait formats but needs fine-tuning. It is untested on Android but should run on any browser as a web app with a server for deployment.

## Functionality Overview.

The application uses a subscription-based model where users must register for an account to access its features. This is for technology demonstration rather than a requirement. The login page is simpler than the original design, with a "Forgot Password" option under "Sign Up" and "Log In". A verification email with a registration link is sent upon signing up.

After logging in, users can change their account details by pressing the profile button. This gives access to operations such as changing the password or username and viewing comment history. Updating this information is done by passing the User ID to the backend to fetch the desired data from the database.

The login, sign-up, and personal info functions were stress-tested for multiple users. The results show that the system can handle a maximum of 862 requests per second, which is sufficient for the current stage of application usage.

| Label | # Samples | Average | Median | 90% Line | 95% Line | 99% Line | Min | Max | Error % | Throughput | Received KB/sec | Sent KB/sec |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| HTTP Request | 1000 | 2 | 3 | 3 | 4 | 4 | 1 | 5 | 0.00% | 914.1/sec | 284.76 | 261.55 |
| TOTAL | 1000 | 2 | 3 | 3 | 4 | 4 | 1 | 5 | 0.00% | 914.1/sec | 284.76 | 261.55 |

*Figure 4 Stress Testing by JMeter*


*Figure 5 Post Log-In Landing Page*

After account verification, users are directed to the landing page with a map and three buttons. The "Select on The Map" and "Search Bar" functions were not reliably implemented but are not required. To access the list of attractions, users must press "My Position" button to temporarily store their location. Pressing "Get Nearby Location" button sends the coordinates to the backend "getNearbyLocation" interface, which requests nearby attractions within 500 meters from Google. This information is rearranged into attraction entities and stored in the cache via Redis for fluent operation. The list is kept in the cache for up to a minute before being sent to a local database.

In the listing interface (Interface C), users see a list of attractions under a shrunken map. Displaying the location of each attraction with red pins was not implemented. Users can filter and sort the list by wheelchair, hearing, and pram access, and whether the place is open at the time of search. On each information card, basic information like Filtering by category was not implemented due to unreliable data from the Google Place API. Ticket price was also not provided by the API and must be stored locally. Users cannot cancel filters without returning to the previous page and using the "GetNearbyLocation" button.

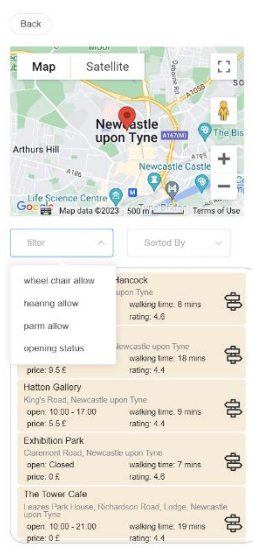


*Figure 6 Attraction Listing Page*

To the right of each display card, a button shaped lie signpost can be seen, upon pressing, it can take the user to navigation page. However, this page was never implemented and for now, it will redirect use to google map as temporary solution.

If users feel one of the attractions on the listing fits his/her preference, then clicking the card will take them to a page with detail information regarding to that information (Figure 1.7). The photo is placed in the upper half of the page with more detailed information about the attraction. These includes, the opening status, the address, etc. the application is also equipped with a comment function which allows users to leave reviews and rating to the attraction. Upon pressing the "+Add" button, the content of review, User Id, Attraction ID will be added as a Review Entity to the local data base. The user also has the option to press the like button on each individual button. This feature uses the "Like Controller" to operate and it is intended for a future function to rank the comments of an attraction based on number of likes.

## Testing Overview

Apart from the aforementioned stress testing, for all other backend methods were assessed by unit testing by a single programmer. The methods pass the test a return code of 200 will be give, otherwise a return code of 400 is provided in case the method failed. By the end of the project, all methods have passed the testing by some of them still display warning of "null pointer exception". For frontend, a checklist was drafted to keep track of various aspects of display aesthetic. However, due to time constraint, several criteria were not met.

**Checklist Body :**

| # | Item | Status | Comment |
|---|---|---|---|
| **Section One** | **Page check** | | |
| <tag>1.1 | Font Size (Match document) | ☐ Pass ☐ Fail | We've decided changing the design of font size |
| <tag>1.2 | Consistency of font size across different pages (font size/color/style) | ☐ Pass ☐ Fail | |
| <tag>1.3 | Page displays no garbled characters | ☐ Pass ☐ Fail | |
| <tag>1.4 | Page displays no overlapping | ☐ Pass ☐ Fail | |
| <tag>1.5 | Color style consistency (Match document) | ☐ Pass ☐ Fail | We've decided changing the design of color style |
| <tag>1.6 | Control positioning matches document | ☐ Pass ☐ Fail | We've decided changing the design of some control positioning matches |
| <tag>1.7 | Page has no spelling errors | ☐ Pass ☐ Fail | |
| <tag>1.8 | Appropriate addition of horizontal and vertical scrollbars | ☐ Pass ☐ Fail | |
| <tag>1.9 | Consistent wording for same functionality/buttons across different pages | ☐ Pass ☐ Fail | |
| <tag>1.10 | Consistent color style for same functionality/buttons across different pages | ☐ Pass ☐ Fail | |
| <tag>1.11 | All required links have been added and are correct | ☐ Pass ☐ Fail | |
| <tag>1.12 | Existence of word limit reminders (including ratings) | ☐ Pass ☐ Fail | |
| **Section Two** | **Functional Logic** | | |
| <tag>2.1 | Navigation is correctly positioned | ☐ Pass ☐ Fail | |
| <tag>2.2 | Search bar provides friendly suggestions for related attractions | ☐ Pass ☐ Fail | We've deleted the search bar function |
| <tag>2.3 | Sort by feature allows selection of different display sorting criteria | ☐ Pass ☐ Fail | |
| <tag>2.4 | Search radius can be adjusted in miles | ☐ Pass ☐ Fail | |
| <tag>2.5 | When stars rating is empty, page cannot be submitted and provides a reminder | ☐ Pass ☐ Fail | |
| <tag>2.6 | When review is empty, page cannot be submitted and provides a reminder | ☐ Pass ☐ Fail | |

*Figure 7 Front-End Testing Checklist*

## Individual Marking

After the end of the project, our group decided to divide the percentage based on the amount of individual effort contributed to the final product.

As such, out of 100% of total participation, MeiHui Li, YiLong Wang, YiLi Luo, JiaHe LI, YiTing Liang, YiFan Qian, HaoYu Zhu will each get 14%. For LiuZheng, he will get 2% out of the entire participation due to his inactivity despite our effort to help.