Debugging Log

Logged a total of **2.5 hours** of debugging.

Entry 1

1.0 hours spent debugging

Failure

When calling the /findFriendsByPosition endpoint, the frontend throws a TypeError: Cannot read properties of undefined (reading 'length') in the ShowResults() function. This happens when trying to read data.results.length.

Experiments

Experiment 1.1

Question: Why is data.results undefined when the backend route is called?

Steps Taken: Logged the raw response from the backend in the frontend using console.log(data). Compared the behavior of the /findFriendsByPosition route with the /findByName route, which works as expected. Reviewed backend code to see how the response is structured in both routes. Noticed that /findByName returns { results: [...] }, but /findFriendsByPosition returns [] directly when fewer than 3 results are found.

Result: Confirmed that data is an array (i.e., []) instead of an object when the route /findFriendsByPosition responds with fewer than 3 results. So data.results is undefined, which causes the error in ShowResults()

Lesson: The frontend always expects an object with a results key, but the backend route sometimes returns a raw array. This inconsistency leads to the undefined property error.

Defect

```
res.send(results.length === 3 ? results : []);
```

The backend sends a raw array ([]) instead of an object with a results key (like { results: [...] }). The frontend expects data.results to always exist, but when the backend sends just an array, data.results becomes undefined, causing a TypeError when trying to access data.results.length.

Types

Type checking **would** have helped. If TypeScript or another type-checking system had been used on both the frontend and backend, it could have enforced a consistent response shape (e.g., { results: T[] }) across all endpoints. The type mismatch between expected object and returned array would have caused a compile-time or lint-time error.

Entry 2

28 minutes spent debugging

Failure

Every time I upload or re-parse building data (from a file or network), the total number of buildings keeps increasing—even when I expect it to remain the same. Some buildings appear multiple times in the search results and when fetching building by short name.

Experiments

Experiment 2.1

Question: Why are there duplicate building entries in the data array?

Steps Taken: Printed buildings.length before and after each call to parseBuildings. Re-ran the function with the same lines input. Logged all building shortNames to check for duplicates.

Result: The number of entries doubled with each re-call of parseBuildings. The same building names like "CSE" or "PAB" appeared more than once. Results from nearby building queries became inconsistent.

Lesson: The original buildings array was not cleared before pushing new data, causing accumulation.

Defect

```
line around 10, the parseBuildings method
```

The function appends new building data to the existing buildings array without first clearing it. As a result, multiple calls to parseBuildings lead to duplicated building entries, breaking search and distance-based queries.

Types

Type checking **would not** have helped. TypeScript or Flow might ensure data type consistency, but this is a logical/data flow issue rather than a type mismatch

Entry 3

1.0 hours spent debugging

Failure

The program is supposed to return nearby buildings sorted by distance, but the results appear in a strange or inconsistent order. Some closer buildings are listed after farther ones, which breaks expected behavior in the UI.

Experiments

Experiment 3.1

Question: Why is the building list not sorted correctly by distance to the given point (x, y)?

Steps Taken: Printed the computed distance values for each building. Printed the typeof for a and b inside the .sort() callback. Logged the final list to check the output order.

Result: Distances were correctly calculated (non-negative numbers). However, the .sort((a, b) => a - b) gave NaN comparison results. The final building list was in incorrect order and even seemed random at times.

Lesson: The sort callback was comparing object references instead of numbers. It needs to extract the distance values before sorting. Also, using Math.sqrt is unnecessary here — sorting by squared distance is enough and more efficient.

Defect

```
.sort((a, b) => a - b)
```

a and b are objects, not numbers, so subtracting them results in NaN. You must compute the distance inside the sort function or precompute and store it.

Types

Type checking **would not** have helped. JS allows subtracting objects, which silently returns NaN — a logic error, not a type error

Edit Log