CASA0017- Web Architecture
GROUP 3
2b|!2b Cafe Hunting Report

Reported by: Esther Wu
Guandi Chen
Xincen Xu
Yuhang Lei

Content

1 Introduction

The architecture of a web platform customised for coffee passionates and remote workers looking for the perfect café in Taipei[1]. The website caters to the specific needs of this target audience, going beyond the typical limitations experienced when using a laptop at a café on the weekend to provide a place for both work and leisure. The motivation behind this effort was to create a user-friendly[5] portal that would pave the way for individuals to easily find cafes that match their diverse preferences, whether for productivity or socialising.

The report will provide insight into the website development process, including the methodology employed for data collection and processing, user interfaces and experience enhancements, key phases of the web development lifecycle, and the implementation of data visualisation and interactive features. Application Programming Interface (API) integration is an important aspect that the report will explore, along with details of the database that supported the project, all of which contribute to the site's mission of facilitating a customised user experience. The website's interface is divided into two main sections: on the left, a functional column (User Input) and on the right, a map (Data Output) powered by the Google Maps API. This layout adopts a user-centered design[4] philosophy and is designed to provide an intuitive and seamless user experience.

In addition, the map will display congestion levels around key sites like metro stations, with the control options situated in the website's bottom left corner. Users aiming to find less crowded workspaces can utilise this feature to identify the locations of metro stations and select cafes outside these bustling zones. Areas within a 200m radius of an MTR station, indicated in red, are those that people seeking quieter spots should avoid, while the yellow zones represent areas extending from 200m to 500m from the station.

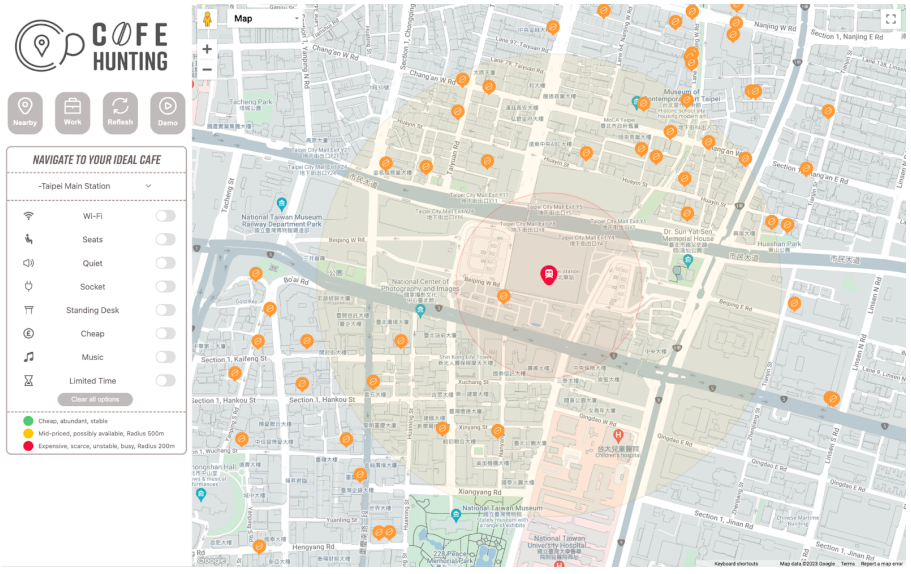

Figure 1: Busy area around the MTR

Further improvements were detailed, including the development of personalised user accounts and advanced search functionality to improve the user experience and the efficiency of the platform, driven by user feedback and data analysis. The implementation of user accounts will involve the secure handling of passwords to ensure that user data is protected through encryption during data transmission. This step is critical to maintaining the privacy of user credentials. User data analysis will play a key role in adapting the platform to meet changing user preferences. In addition, the introduction of a user review feature is expected to enhance the credibility and reliability of the platform.

2 User Interface & Experience

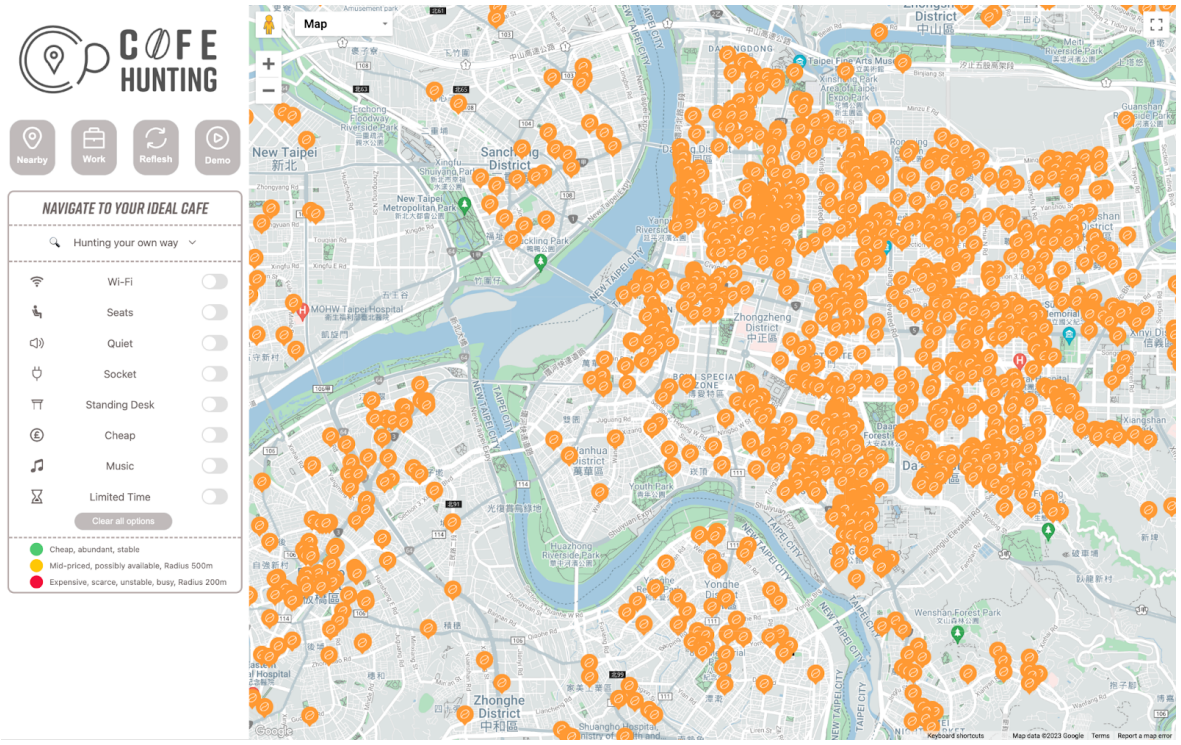

Figure 2: Website View

2.1 Website Features

Nearby: This feature leverages the Geolocation API[2], which requires user consent to obtain their current coordinates. The system then uses these coordinates to place markers on the map, displaying information about nearby coffee shops. Notably, if the user's location is outside the predefined area of Taipei City, the map automatically centres on Taipei and informs the user that they are beyond the range of the database.

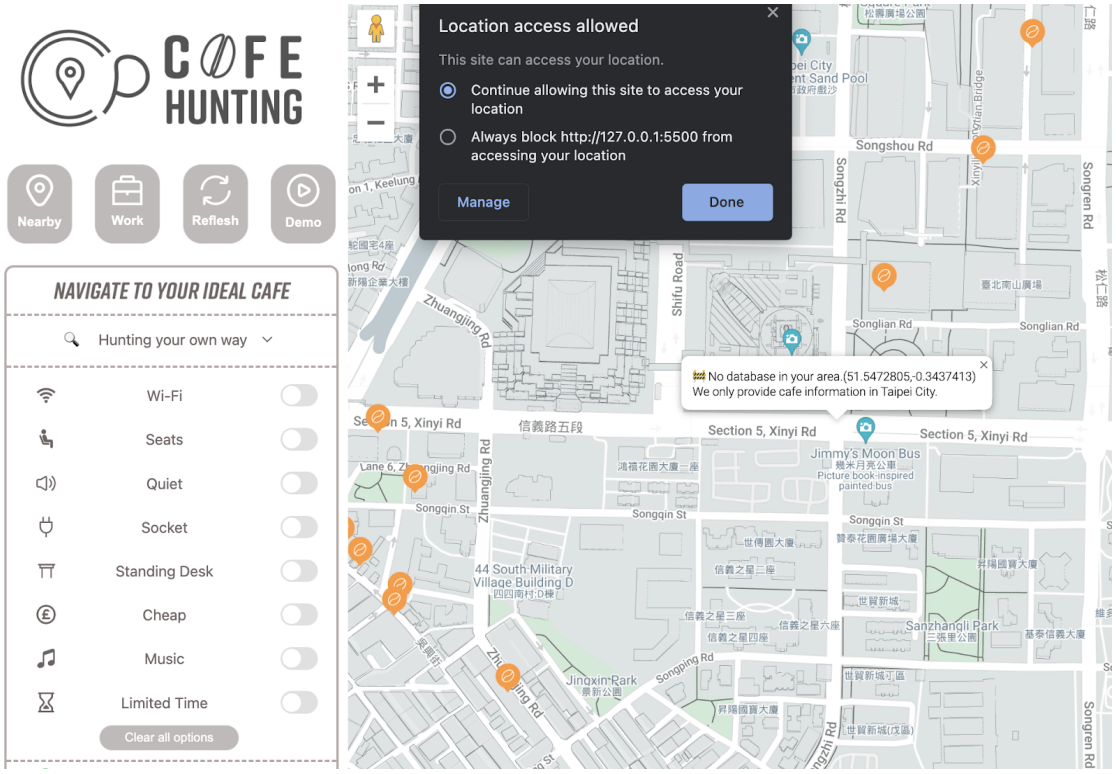

Figure 3: The website asked for the location used

Work: Targeting digital nomads and individuals who use cafes as workspaces, this feature provides quick access to coffee shops with specific amenities conducive to work. This option is integrated with additional filter options for enhanced user convenience.

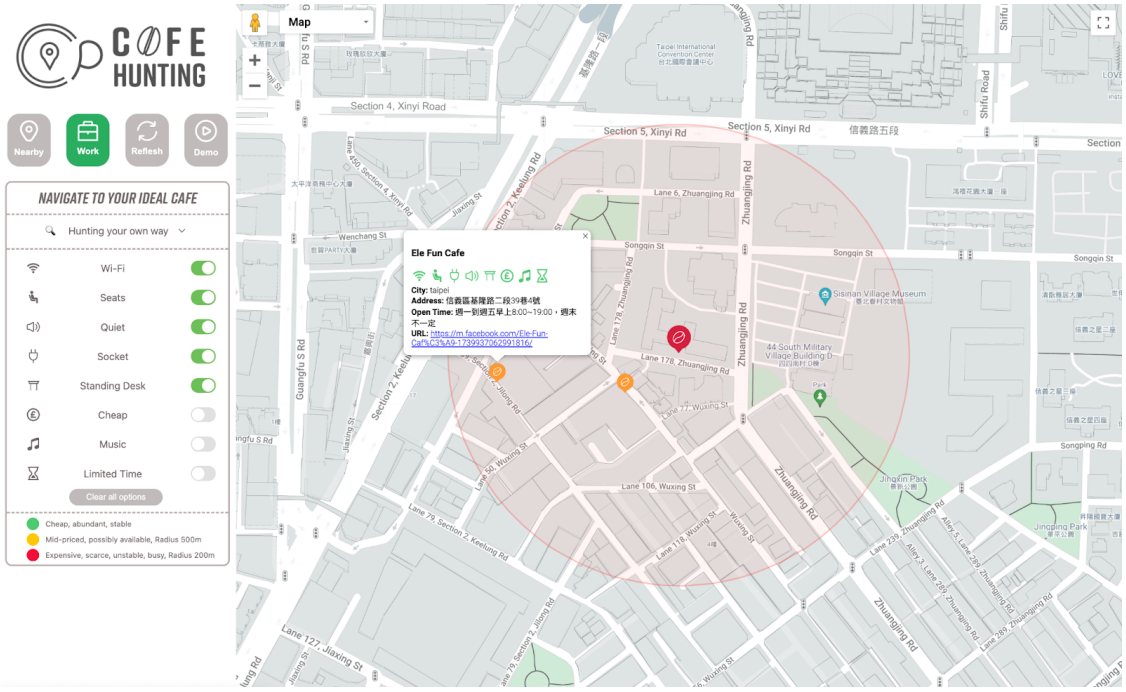

Figure 3:Work mode for quick search cafe suitable for work

Refresh: This option allows users to quickly clear any markers or selections on the map, resetting their search parameters.

Demo: Catering to users outside Taipei City, this feature provides a simulated experience of the website's functionalities, showcasing its capabilities in finding coffee shops.

Hunting Your Way: This innovative feature offers two distinct search modes: a graphical search focused around MRT stations and a data-based search using the name of the coffee shop.

Filter Option: To further refine the search, the website includes eight filtering criteria (Wi-Fi, Seats, Quiet, Socket, Standing Desk, Cheap, Music, and Limited Time). Users can apply these filters to their search, receiving immediate feedback on the map to the right, enhancing the efficiency and effectiveness of their search.

Each of these features contributes to a user-friendly and efficient platform tailored to meet the unique needs and preferences of individuals seeking the ideal coffee shop experience.

3 Exploratory, 'storytelling' nature of website

The website, designed for cafe enthusiasts and remote workers, offers an engaging, exploratory journey into the world of cafes, especially in Taipei. Its user-friendly interface immediately presents a location-based search, inviting users to discover cafes nearby. For those outside Taipei, the 'demo' mode provides a virtual tour, showcasing the vibrant cafe culture of the city.

The working mode feature is tailored for users seeking a conducive work environment, suggesting cafes based on amenities like Wi-Fi and quiet spaces. Users can search for specific cafes by name, accessing details like opening hours and addresses. The customizable filters add a personalised touch, allowing selections based on unique preferences such as music or ambiance. Direct links to cafe websites offer a deeper dive, where users can explore menus, the history of the cafe, or specific events. This storytelling aspect not only informs but also connects users with the distinct character and culture of each cafe, transforming the simple act of finding a cafe into an immersive, culturally enriching experience.

4 HTML and JS files at front-end

4.1 Index.html

The front-end of the site has been thoughtfully designed to be user-friendly and engaging, starting with an introductory section upon entering the site. This introduction clearly explains the purpose of the site and instructs the user on how to use the interactive bar on the left-hand side. It ensures that even first-time users can easily understand the functionality of the site and how to navigate it.

The layout is split into two distinct sections. On the left side, users can enter their information or preferences in an interactive area. This area includes filters that allow users to conduct customised searches for cafes based on criteria such as Wi-Fi availability, seating, noise levels and more. A 'clear all options' button makes it easy to reset the filters, while a colour guide helps to show the characteristics of the café by colour.

The element labelled "map-canvas" on the right-hand side of the interface is dedicated to displaying the map and associated data output. It displays a Google map, augmented with external JavaScript libraries and scripts (including jQuery, the Google Maps API, and custom scripts such as 'map.js', 'locateUser.js', 'mapStyle.js', and others) to handle various map interactions and user location features.

The website consists of two main pages: the home page ("index.html") and the data search page ('datasearch.html'). The home page serves as a welcome interface to the application, clearly describing its features and usage, while the datasearch page conducts detailed information retrieval based on user input.

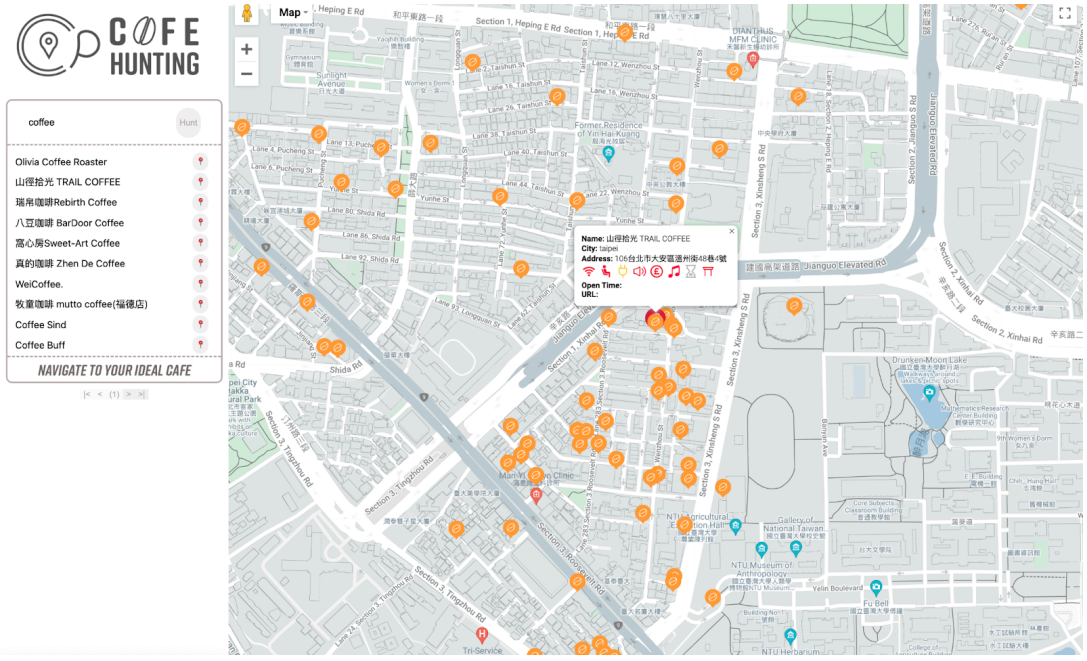

Figure 4:Data searching page ('datasearch.html')

4.2 Map.js

The map.js file is an indispensable part of a website's interactive map functionality. It initialises Google Maps, sets its initial view, and applies custom styles for an engaging user experience. The script manages a series of markers (markerArray) representing cafes, user locations and metro stations, each with a unique custom icon for easy identification. One of the key functions, "getData", interacts with the back-end API to get the relevant data based on the map centre and then populates the map with the appropriate markers. The script dynamically adjusts the data displayed on the map based on the filters selected by the user in the interactive field on the left, and even generates URLs for data retrieval. map.js also includes event listeners for user interactions (such as dragging the map), which enhances the interactivity of the map.

Two approaches were tried for connecting to the back-end API, both using the $.getJSON method to fetch data by sending requests to a specific API address. The difference lies in the API addresses used and the nuances in data processing and token creation.

The first method is to use the ‘initialize’ function to initialise the map after the page is loaded. Then, use the ‘getData’ function to get data near the centre of the map when the map is initialised or dragged. Using $.each in the callback function to iterate over the data retrieved from the back-end via $.getJSON creates a google.maps.Marker objects with a specific location, icon, and custom information for each data point. To increase user engagement, a click event listener is attached to each marker, displaying an information window with location details upon user interaction. The ‘filterMarkers’ function is triggered on page load and when the user interacts with the tickboxes, filtering the visibility of the markers based on the selected criteria. **This process could be described as the back-end sending all data to the front-end; then, the front-end would complete data filtering and display.**

The second method for connecting to the back-end API, distinct from the first, places a significant emphasis on real-time interaction and flexibility. This approach is particularly focused on how the filters change and how these changes are communicated to the back-end. In this method, the website listens for any changes in the filter options, such as Wi-Fi availability, seating, and noise levels, which are represented by checkboxes like wifiToggle, seatsToggle, and quietToggle. **In the second method, the back-end filters the data and sends limited data to the front-end. The front-end only passes back the filter value and displays the returned data without any processing of the data.**

Each filter option comes with an event listener, such as wifiToggle.addEventListener ('change', ...), that is used to detect any changes in the state of the filter. When the user interacts with these filters, the script will immediately update the appropriate variables,

such as wifi, seat, quiet, etc. These variables are then used to build parameters for API requests dynamically.

The ‘displayGeneratedUrl’ is responsible for combining the current state of the filter as a parameter into an API request URL that is refreshed as the user changes filter selections to ensure that the data retrieved from the back-end is always relevant to the user's current preferences.

Once the URL is updated, the ‘getData’ function is called. This function sends a request to the back-end application program interface for the latest data that matches the filter criteria. The map is then updated with the new markers reflecting the latest data, thus providing a dynamic and reactive user experience.

The real-time update mechanism in the second approach enhances user interactivity compared to the first approach. It allows for immediate feedback and adjustments based on user input, resulting in a more engaging and personalised map experience. In addition, this method updates data only based on user input, optimising network usage, reducing unnecessary data capture, and improving the overall performance of the website.

4.3 locateUser.js

The locateUser.js script focuses on enhancing location on the map. It sets the geographic boundaries and centre of Taipei City as the basis for locating the user. When the user clicks on the specified button, the script activates the locateUser function, which first checks if the user's device supports geolocation. If geolocation is supported, the script will retrieve and display the user's current location on the map. For users within the Taipei city limits, the map will zoom in on their location and add a marker with a circle around it to represent the area. If the user is not within these boundaries, the map will be centred on Taipei City, and an information window will appear stating that the data is only available within Taipei City. This feature ensures that users have a background-aware map experience, highlighting their surroundings or focusing on the central area of Taipei as required.

5 Design and technical reporting of API and database at the back-end

5.1 Node.js

This part contains an analysis of a ‘Node.js’ script that plays a key role in transforming JSON-formatted data into SQL insert statements. This automated tool greatly facilitates populating the MySQL database with data from various JSON files. The script first reads 'cafedata.json' and then parses the data into JavaScript objects. It then constructs SQL insert statements for each data object in the array. The final action is to write these SQL commands to a file called "Cafe.sql".

The script has included error handling at several junctures (specifically during file input/output operations and JSON analysis) to address the risk associated with errors due to formatting inconsistencies that can compromise data integrity or cause system failures. Its asynchronous design(the program can handle more than one job at once) facilitates non-blocking file operations, which improves efficiency in handling large data sets and ensures minimal impact on system performance.

It is to be noted that following secure coding practices is crucial during implementation(a group name and password are used), especially in terms of sanitising inputs to prevent SQL injection attacks. Additionally, consistency in documentation is essential, as evidenced by the fact that success messages in the console must be consistent with the actual filenames used.

In the general context of the data transformation utilities discussed, the Node.js scripts described demonstrate the effectiveness of server-side scripting in automating and streamlining database administration tasks.

5.2 back-endresearch.js

The 'back-endresearch.js' script is at the core of the back-end service implemented using 'Node.js' and the Express framework. The script is key to interfacing with the MySQL database and is set to listen on the specified port number 8816 - a parameter that was determined early in the code to ensure consistent referencing and facilitate testing.

The database connection is launched with user ‘zczqy80' predefined credentials, and the script has a powerful error handling feature that provides a clear path to log information in the case of an error connection. This proactive measure helps to diagnose and resolve connection problems in a timely manner.

After a successful database connection, 'back-endresearch.js' defines an API endpoint at "/Table/Cafe". This endpoint is configured to process an HTTP GET request that triggers an SQL query to retrieve records from the 'Cafe' table with a 'wifi' and 'quiet' rating, both above 4. The conditions of this query were purposely hard-coded as a constant variable to serve as a baseline for validating the database response during the initial testing stage.
After creating a successful database connection, 'back-endresearch.js' defines an API endpoint with fixed variables at '/Table/Cafe'.

The script has been further improved to provide more dynamic and responsive queries. The front-end now provides a series of filters such as wifi, seating, quiet, cheap, music, outlets, standing desks, and limited time, allowing the user to specify criteria for finding a cafe. These filters represent user-defined parameters that are passed to the back-end via the requested query string.

For each parameter provided, the script adds the appropriate condition to the SQL query. This approach ensures that the final query is dynamically tuned to contain only relevant conditions, thus effectively filtering the café data according to the user's preferences. For example, when the user clicks the Work-Mode button on the front, it changes the filter value for an attribute such as WiFi (e.g., switched between 0 and 4) and then sends the updated filter value to the back-end via a URL. The back-end, running in the same domain, retrieves this filter value from the URL and uses it to generate a set of results that match the filter criteria. These results are then made available to the front-end via an application program interface. The front-end then accesses this API to get the filtered data and displays it on the map.

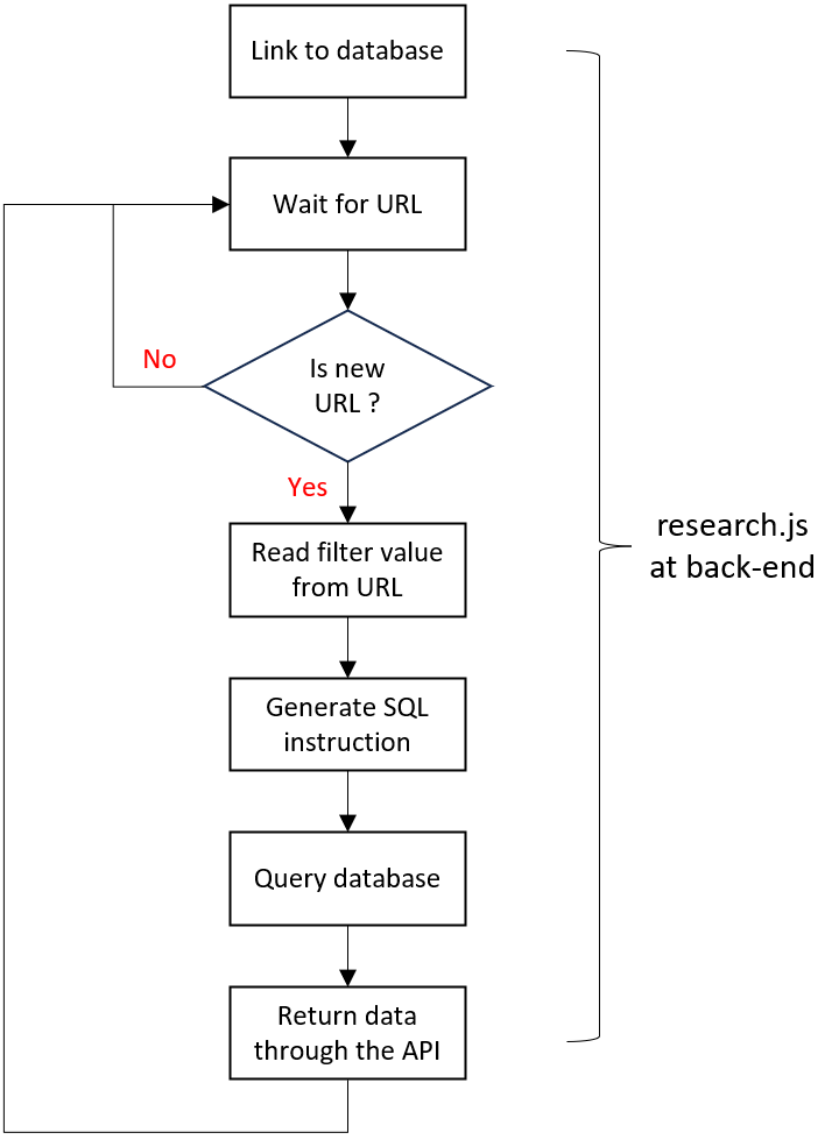

Figure 4: Workflow for back-end

As a result, this enhanced approach has replaced the previous fixed query, bringing more functionality and an interactive experience to the user.

Error handling is conducted in the script: if the SQL Query encounters an error during execution, the error details are logged, and the admin is notified of the error by the server with a 500 status code. In the case where the query is executed successfully but no matching records are returned, the server logs this lack of results and communicates it to the client with a 404 status code, indicating that there is no data that matches the specified conditions.

Instead, when the query successfully retrieves data, 'back-endresearch.js' records the results and sends the dataset to the server. Additionally, the script will provide feedback when the server launches, outputting the address and port it is listening on to the console, thus confirming that the service is ready to process the request.

It is worth noting that the actual password for the database connection in the back-endresearch.js file was intentionally omitted in this document to comply with best practices for secure credential management.

6 Future Prospective

6.1 User account

Through URL links and API to achieve the front and back-end data transmission, the web could achieve the function of users to create personal accounts based on this architecture in the future vision. Similar to the query for the cafe, the table of usernames and their passwords could be created in the database. When a user enters a password for their account at the interface, the front-end sends the account and password back to the back-end via the URL. The back-end compares the received account password with the table in the database to complete the user's login function. In addition, other storage content could be added to the table, such as user preferences and frequent visits to cafes. However, transferring account names and passwords through URLs is risky. For example, the URL connection may be intercepted, resulting in the leakage of usernames and passwords. In view of this situation, this risk could be avoided by increasing the encryption mode, which encrypts the password and username at the front-end, and then decrypts it at the back-end.

6.2 Comment and recommendations on the website

The implementation of this function is similar to creating users' accounts, where user comments are transmitted from the front-end to the back-end through a URL and stored in a database on the back-end. However, based on this architecture, there may be some difficulties in implementing user-free comments. The table in the database is difficult to query and store every comment for each coffee shop easily. Long comment content can also cause difficulties in transmitting query results to the front-end in the back-end. In addition,

filtering user comments is also one of the issues, such as illegal or unhealthy comments and malicious negative reviews. Therefore, in future versions of the website, the free balance function may be removed and changed to a scoring system. Create a separate recommendation table in the database to store the average score of each coffee shop, and based on the average score, place higher-ranked coffee shops on the user's list from high to low.

6.3 Improvements to website security

In the existing front-end, the access website of the front-end does not use any encryption protocol. In future versions, it should be ensured that the website uses HTTPS security protocol and uses SSL or TLS to encrypt the transmitted data. Especially after users can create personal accounts, this can, to some extent, prevent the leakage of their personal information. In terms of back-end databases, corresponding permissions can be set for different users, allowing only ordinary users to query data on the limited tables in the database. In addition, an administrator account should be set up for modifying and deleting database content. At the same time, the system and libraries should be updated and maintained in a timely manner on both the server and local hard drive, and data should be backed up regularly. This ensures that the data remains intact even after being attacked. It was worth pointing out that the back-end data should also be encrypted to prevent risks such as data leakage caused by the back-end. Symmetric encryption methods such as AES should be used for data transmission between the front and back-ends for data transmission speed. The key transmission should use asymmetric RAS encryption to ensure the security of the key. On the back-end, it is planned to use hash functions (SHA) to verify the integrity of data for server encryption[6].

6.4 Establishing a Database for London Cafés

Regarding the Cafe Nomad API it is developed by an open-source initiative called 'List Power,[3]' which allows the community to gather data on specific topics collectively. While building the database, it also fosters a community of shared interests. In continuation of this API's data creation model, we aim to establish a database relevant to local cafés in London. Additionally, since the data is community-generated, there are often duplications and incomplete information. When establishing the London version of the database, we also hope to incorporate this variable to enhance the quality and completeness of the information.

Conclusion

The website is structured into two distinct sections: the front-end and the back-end. The front-end is designed for user interaction and visibility, enabling users to search for their desired cafe. At the same time, the back-end is built upon the API from 'Cafe Nomad', which is used to establish a database for the website. The main objective is to integrate the front-end with the back-end. This integration is achieved through code that allows the front-end to send user-selected cafe details to the back-end. Subsequently, the back-end processes these details according to the given commands and sends the relevant data back to the front-end.

For user security, in the future, the website will manage every user name and their password to protect their security. Also, it will allow users to freely comment on the cafes, which also can be a reference for other users to choose their cafes.

Here is our website link:
http://casa0017.cetools.org/~group3/index.html
Link to github:
https://github.com/sjosk/CafeHunting

Reference List

1. cafenomad.tw. (n.d.). *Cafe Nomad - API v1.2*. [online] Available at: https://cafenomad.tw/developers/docs/v1.2 [Accessed 15 Nov. 2023].
2. MDN Web Docs. (2023). *Geolocation API*. [online] Available at: https://developer.mozilla.org/en-US/docs/Web/API/Geolocation_API.
3. Chuan Haoyou (2023). *ListPower 清單力量*. [online] GitHub. Available at: https://github.com/howtomakeaturn/ListPower [Accessed 15 Nov. 2023].
4. Wikipedia Contributors (2019). *User-centered design*. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/User-centered_design.
5. Kuang, C. and Fabricant, R. (2019). *User Friendly*. MCD.
6. 360linker (2021). *An overview of seven common encryption algorithms and their implementation*. [online] Zhihu. Available at: https://zhuanlan.zhihu.com/p/347114235 [Accessed 15 Nov. 2023].