Jack Cheng Ding Han (150159519), Chung Lam Yip, Holam Yu

COM3504 The Intelligent Web

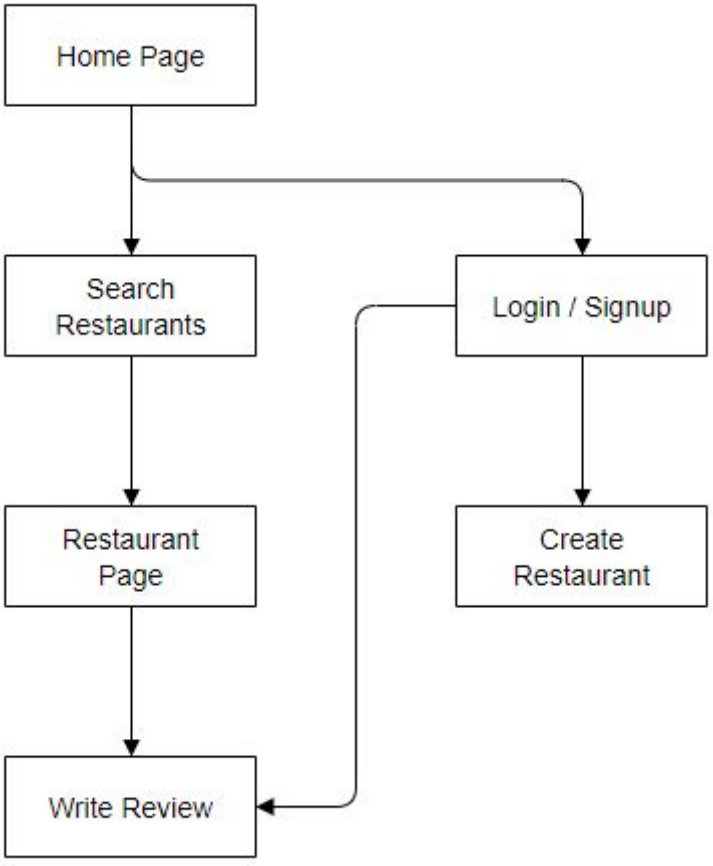
Jack Cheng Ding Han (150159519), Chung Lam Yip, Holam Yu

Introduction

More than half of web traffic now comes from mobile phones as opposed to desktop computers (kemp 2017). It is therefore important to optimise websites for the mobile browsing experience. This report details the creation of a website/progressive web app for finding restaurants and reviewing them. A user can also create new pages for restaurants which that do not yet exist in the database.

Diagram

Implementation



The diagram provided shows the instructions to the user.

When the users open the app(Locate in Home Page), the users could directly see two links which are Restaurants and Login/signup.

If the Restaurants link be clicked, the users are allowed to search the location of restaurant, see the restaurant comments. Whereas, if the users click the Login/ Signup link first, the users could activated two new function which are adding new restaurant into the app and writing review in the app.

Requirements

Searching for restaurants and displaying results (15%)

Solution:

On every page, the user is able to search using keyboards from an input located in the navigation menu, this queries restaurants by tokens in their description as well as their name. As stated in The Database section, by creating an index for certain fields such as the name and description of the restaurant, queries can be constructed and executed efficiently. Examples of the syntax used include:

- Surrounding text in quotes mean it must be present.
- Prefixing text with a minus sign means it must be absent.

On the dedicated search page, the user is not only capable of searching using keywords but also by selecting cuisine tags that each restaurant has. Unlike keywords, the results must satisfy all the tags in the input for them.

This search page also allows the user to use a map to search for all restaurants satisfying the keywords and tag cuisine queries within a certain radius (adjustable using a slider) relative to a certain location (the coordinates of an address within an input or, if the user allows it, the current location of the user).

Issues:

Searching using the map must still be able to work should the browser not allow the website to know the current location of the user.

Requirements:

The first required way to search, using keywords and tags, has been satisfied. By default, a query will use the disjunction of keywords typed in to the form but conjunction can still be done, not by the “AND” operator, but by surrounding the keywords that must be present in quotes. The query will always use the conjunction of cuisine tags (if any).

The second required way to search, using a map, has been satisfied. The browser will ask the user to allow the website to know the user’s location and the system will return the results on a map only for restaurants (that satisfy the keyword and cuisine queries if the inputs for them have been filled in) within a user-defined radius (the range of the slider is 1 km to 17 km)

Limitations:

The result displayed on the map depends on the Google Maps API. Markers on the map may overlap when zoomed out.

The restaurant page (15%)

Solution: The restaurant page display all data of the restaurant, with the geolocation being process to display on a mini-map. The mini-map can be clicked on to redirect to google map page with the restaurant location marked. User can also find restaurant by same cuisines on the page. All reviews would be shown on the page, and a average rating would be display as well.
Issues: Many data needed to be process on this page, making the loading time of the page taking considerable amount of time.
Requirements: Since there are no user uploaded photo implemented, the requirement of showing it had not been implemented. Other requirements were met.
Limitations: When linking from mini-map to google map, the process was a redirect rather than a pop up. Implementing direction function within our website could be helpful for remove too much dependency of other website.

Allowing restaurant reviews (5%)
Solution: Only user can access the write review form. Restaurant reviews were submitted with a *POST* form, field to fill out included author name, star rating, and a optional comment. Since the review data was related to new data added, a refresh would be necessary for displaying the result on page. The *POST* form was followed by a socket-io when submitted, so that other user know a new review has been added.
Issues: Designing socket-io for a user submitted form was difficult, since danger attackers could implement a cross site scripting with the data submitted by the socket.
Requirements: A requirement that had not yet been met was the user upload photos. Other requirement including allow logged in users to add ratings, reviews to the page were all met.
Limitations: A limitation of the solution was the extent of promises, since when creating a review, a lot of depency needed to be checked, e.g. user logged in, correct restaurant id, correct form data. This issues also lead to socket-io sending update message even when the form data failed to process.

Creating a new restaurant page (15%)
Solution: The user can create a new restaurant page by register and login to the system. The user can create a new restaurant page by upload the restaurant photo, snapshot the user photo, upload the restaurant name, upload the cuisine type of the restaurant, upload the location of the restaurant and upload the details of the restaurant. After the new restaurant information upload to the system, user could be able to search the new restaurant.
Issues:
Requirements: This design have fully reach the requirement because the user could create a new restaurant page and upload it into the system. The database is carefully saving the information of the new restaurant in both online and offline.
Limitations: The user could not clear see the status whether they are in log in or log out. They come only know this by seeing whether they can add new restaurant or not.

The Server Architecture (20%)
Solution: Ajax:login, register, search
Socket-io: receive new review
POST: add restaurant, write review

Issues: Decide which function to use can be difficult.
Requirements: Data communication implemented using JSON. All 3 method of processing form data had been implemented.
Limitations: mixing function

The Database (5%)
Solution: The restaurant is structure with each element that define the property of it. The essential parameters included name of restaurant, address, geographical location, and cuisine. Extra parameters unique to this website include description, rating star, and review.
Issues: Address can structure very differently in different country. Cuisine and Review needed to map to different table with primary id.
Requirements: To search for restaurant in database, keyword can be used to match the text contain within all the field, cuisine can be used as filter to remove all un-wanting result, geographical location can also be used to remove result outside search query marker with specify radius.
Limitations: In order to fully utilized the data structure, the name of different field needed to be known, extra map api support are also needed to precisely pin the geographical location.

A Progressive Web App (10%)
Solution: User should be able to cache visited pages, and browser it offline without crashing. The cache was implemented with '*service worker*', and using *Stale-while-revalidate* method, the page should load quicker by utilizing cache data quick response. The flaws of using such approach was that the page needed to be dynamically update when fetch after cache (some variable dependence data may affected, e.g. session.user_id).
Issues: Handling cache data can be difficult, especially when the form contain may *POST* data like review and create restaurant. Some function need to be disable when offline so that user could understand what they can browsing offline. Map function was tricky to handle offline.
Requirements: The browser would told user when they offline, without crashing the page. However, *POST* form data would not be send when user submitting offline. A *localStorage* had been implemented to store the form data, but currently have no usage of it.
Limitations: As mention, the *localStorage* currently have no usage. A way to utilize the data will be to put the stored value back to the form next time the user browsing, or even submit to the server immediately after user get back online.

Jack Cheng Ding Han (150159519), Chung Lam Yip, Holam Yu

Division of Work

All the members of the group contributed fairly to the assignment solution. The solution was designed jointly and then each member lead the implementation of the corresponding requirement in the assignment specification. Within this report, each subsection was authored by the respective major contributor.

Name	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	Contrib.
Jack Han	12	6	0	6	5	1	0	5	**35**
ChungLam Yip	3	7	3	7	7	3	0	5	**35**
HoLam Yu	0	2	2	2	8	1	10	5	**30**
Total Score	**15**	**15**	**5**	**15**	**20**	**5**	**10**	**15**	**100**

Extra Information

To run the code, please make sure:

1. Run '*npm install*' to install all packages in *package.json.*
2. Make sure MongoDB port is the same as specified (Default: *27017*)
3. Listen at *https://localhost:3000/*

Bibliography

S. Kemp,
Digital in 2017: Global overview
, Jan. 2017. [Online]. Available:
https://wearesocial.com/special-reports/digital-in-2017-global-overview