COM3504 - Intelligent Web

Restaurant Critique Final Release

Authors: Will Garside, Rufus Cope, Greta Ramaneckaite

1 Introduction

This report discusses the structure and development of Restaurant Critique, a website and progressive web app for finding and reviewing restaurants. The layout of the website, structure of the database, and the data storage information is discussed in Chapter 2. Each task of the project is discussed with up-to-date progress and development issues, as well as the division of work and installation and usage instructions.

2 Diagrams

Figure 1 is a model of the website, showing how each page is accessed. The pages on the left hand side are accessible from anywhere in the site, and are linked in the header and footer.

The second image, Figure 2, is a basic UML model of the database, showing how each object is connected. The four main objects (with dedicated collections) have darker headers, and the other objects are common JSON objects, separated out to make the diagram easier to read.

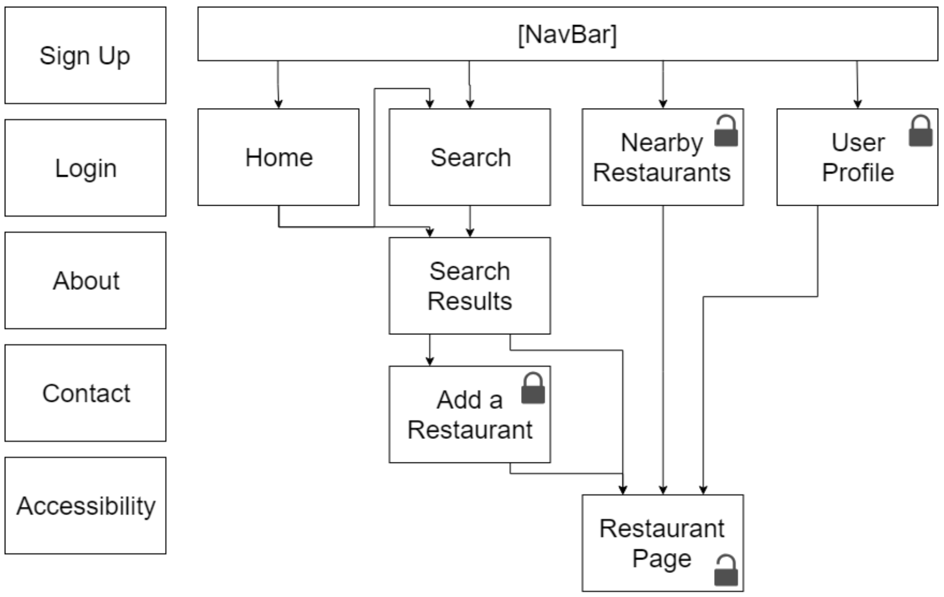

Figure 1: The layout of the website navigation

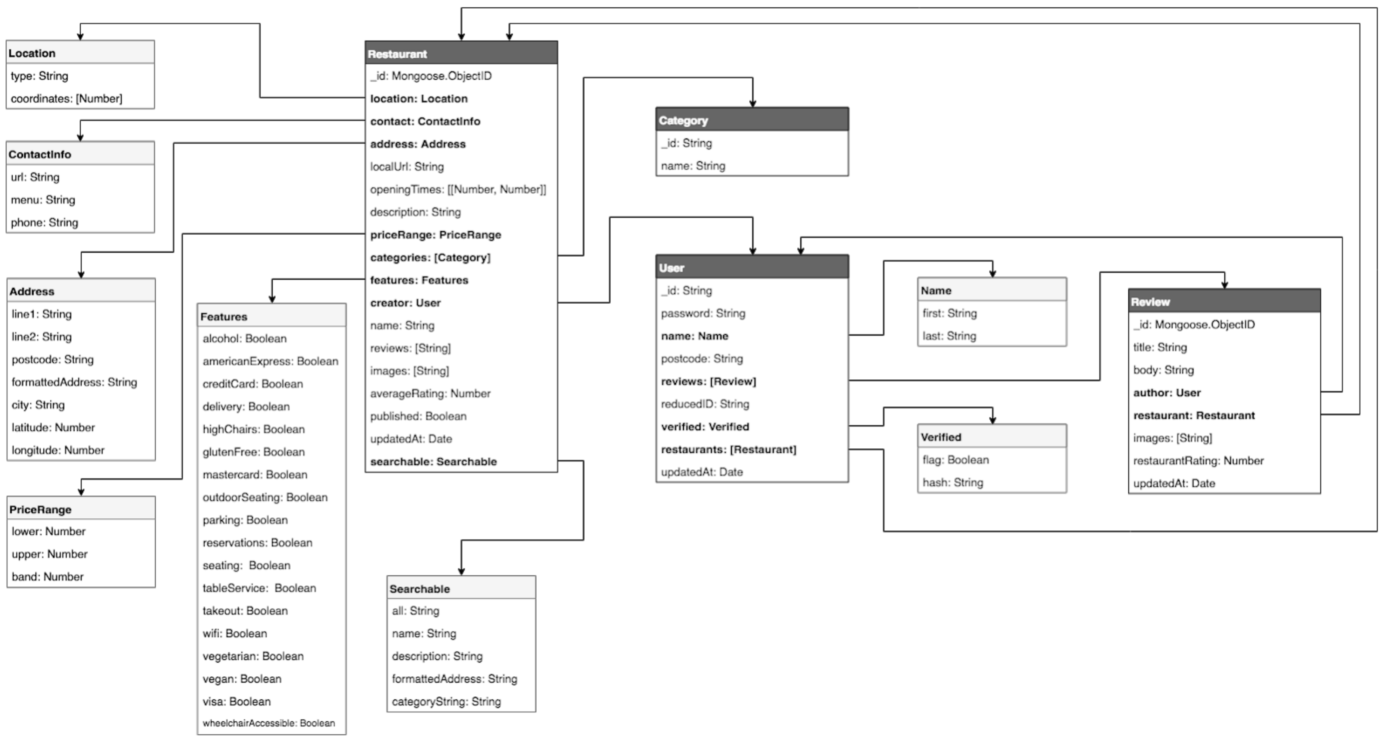

Figure 2: UML model showing the database layout

3 Individual Tasks

3.1 Searching for Restaurants and Displaying Results

Solution A User can search for a Restaurant from the navbar or on the dedicated search page. Each Restaurant object has a `searchable` field, with subfields for the name, description, address and categories of the Restaurant. While this does require more storage space and processing for each Restaurant, the overhead introduced is relatively minute - it's a simple one-time string manipulation. The `searchable` field gives the advantage of removing all punctuation and indexing the Restaurant for optimised lookups. A RegEx operation is used to match each of the words in the User's query to the `searchable` field and return the matching Restaurants. Another option for the search function was to use Mongoose's `$text` search feature. However, this doesn't support partial matching (hence the use of RegEx), and the closed nature of the built-in search feature blocks any useful customisation. A page has also been created to find Restaurants in ascending order of distance from the User. This uses Google's Javascript V3 Maps API and Geocoding service to find the User's location and then compare it to the coordinates of the Restaurants with GeoJSON mapping.

Issues There is no application of stemming or a stoplist for the `searchable` field currently, however this makes no significant difference to the results.

Requirements The requirement for returning results from free keywords has been successfully met, as well as finding Restaurants by distance. The results are also sortable and can be filtered by the user.

Limitations The search function uses an AND operation between terms in the query, and only searches set fields of the Restaurants. This means that if a User searches for a valid term combined with a non-indexed term (e.g. "Nando's 0114") then the function will return no results.

3.2 The Restaurant Page

Solution A User can view the details of a specific Restaurant by navigating to the dynamically created URL. The URL is made up of the Restaurant's name and its postcode, and triggers a GET request to the server to return the Restaurant object and its child Review objects. The page displays the Restaurant's attributes, as well as the Reviews that have been submitted for it. A Review submission form is also accessible if a User is logged in. The page has a link to Google Maps to show directions from user's current location to the restaurant. The category tags are all clickable, and display all Restaurants with that category on the search page.

Issues There are no issues with the restaurant page.

Requirements The requirements defined for a 'simple solution' have all been met by the Restaurant page - other than Review submission.

Limitations There are no limitations for the restaurant page, all requirements have been fulfilled with the following additional features successfully implemented: current day's opening times highlighted; full-screen image viewer, link to Google Maps to get directions to restaurant from current location.

3.3 Allowing Restaurant Reviews

Solution Reviews are stored as separate objects, whose IDs are stored in a list in their parent Restaurant object. The Reviews are displayed as a section of each Restaurant's page, with the title, body, and rating displayed alongside the author's name and submitted images. Logged in Users are able to access the form to submit a new Review on the Restaurant page, and a login prompt message is displayed otherwise. When offline, the failed AJAX submission is stored in an IndexedDB until the user regains connection, at which point the Service Worker Background Sync API is called to send the review to the server.

Issues There no issues with uploading and displaying restaurant reviews. Offline functionality has been fully implemented, however one error has held us back from the code being fully functional. Full code for the feature can be found in our codebase, however at this time reviews are only stored in the IndexedDB locally and are not submitted to the MongoDB database.

Requirements Reviews can be uploaded (only) when a User is logged in. The form allows a rating of the restaurant, camera captured images, and a title and main review body to be submitted. The images are uploaded to the server and displayed for all users. Offline users can submit reviews locally.

Limitations The review form uses WebRTC to capture images but lacks the functionality to upload files from the device's storage. This is due in part to issues transmitting WebRTC data alongside other files.

3.4 Creating New Restaurant

Solution When a User searches for a Restaurant, an option to create a new Restaurant will be presented at the end of the result list and also if their search returns no results. This will direct them to a form with all the necessary inputs to create a new Restaurant and submit it to the database. A User must be logged in to view the form and their account must be verified for them to submit it - if they are not verified they can save the Restaurant and return to submit it at a later point. Verified Users also have the option to save their progress. The 'Save' option adds the Restaurant to the database but sets the `published` flag to `false`.

Issues The form for submitting a new Restaurant is complex due to the amount of fields that a Restaurant object has. However, having researched similar products it is a consistent theme due to the amount of detail required.

Requirements All of the fields for a Restaurant can be submitted via the form. Address lookup is implemented through either a postcode search or by dropping a pin on a Google Map. Offline users are unable to add Restaurants. This could be added very easily later in a similar manner to offline reviews, utilising Background Sync API and a local IndexedDB.

Limitations The user can only upload images from their local storage, rather than taking live photos.

3.5 The Server Architecture

Solution The structure of the directories has followed the standard NodeJS format, with the definition of the database models in their own files, and the routes for each page being separated into relevant files also. AJAX has been implemented where necessary, to send data to the server or request data to be displayed on the page.

Issues There are no issues with the server architecture. Assignment requirements have been fulfilled.

Requirements Express middleware has been used where applicable in order to make processes more effective. Socket.io has been used for displaying new Reviews on the Restaurant page: this can be observed by opening two instances of the page and submitting a Review from one of them.

Limitations Currently the client-side JavaScript is also in the one-file-per-page format. Whilst this avoids complications and makes it easier to read/understand, it does mean that clients will have to cache a new file for every page, rather than having just one file to cache when they first visit the site. This would be analysed and addressed for a production release.

3.6 The Database

Solution The database contains collections for Restaurants, Users, Reviews and Categories, and Mongoose Schemas validate each object. Client side validation is also utilised to encourage the input of useful and valid data. The database has been deployed to one of MongoDB's free Atlas servers so that it does not have to be initialised for each development session, and the data is consistent across all instances of Restaurant Critique during the development stage. This also means that when Restaurant Critique is put into production, the Atlas server can be upgraded to handle the extra traffic without issue.

Issues Prior to setting up the Atlas server, an instance of the database was hosted locally on each group member's computer. This caused several issues with mismatching data and incompatible database upgrades, as well as requiring the foresight to initialise the database before development. Once the database was deployed on the Atlas service these issues were largely resolved, due to the removal of the need for frequent re-initialisation. The lack of partial search on documents through text indexes was a very frustrating discovery, as in SQL databases this functionality is readily available. The method used to solve this issue is discussed in Section 3.1.

Requirements The database is implemented using MongoDB with test data already generated to allow restaurant search functionality.

Limitations There are no limitations with the database setup or the test data used.

3.7 A Progressive Web App

Solution A Service Worker has been created which a number of different approaches for different requests. Pages which require it are network first, then offline page, e.g. Search or Contact page. All other GET requests are handled by a cache then network approach, whilst POST requests are network only.

Issues The Service Worker implementation was found to be full of corner cases, specifically regarding POST requests such as login and AJAX submissions. For example if you're on the home page and you login, the header will not update to show you are logged in. This is due to the page being fetched from the cache and not updated with the new state. Some of the more pressing corner cases have been addressed, however not all.

Requirements Requirements state that the user must be able to access the basic site pages plus their last viewed restaurant page whilst offline. Furthermore pages such as search which require a server connection to function must redirect to an offline notification, whilst avoiding any crashes. These requirements have all been met.

Limitations Spending the best part of 10 hours trying to get the service worker running had a big impact on time available for finding solutions to the many corner cases presented, therefore fulfilling the main requirements stated in the brief were prioritized. As a knock on effect of this some bugs may be found with some of the more advanced functionality in use with the service worker.

3.8 Quality of Solution/ Keeping Solution Manageable

Solution The codebase used in creating Restaurant Critique has been through many iterations, with multiple core functionality re-writes. It is now reusable, maintainable and well documented. This provides an overall high quality solution, with few major issues. The view engine of choice for this solution is Pug. This has led to the creation of simple and effective HTML views with integrated control statements.

Issues There are few, if any, issues with the quality of the syntax of the code: high-standard ES6 guidelines have been followed throughout the project. The manageability of the project has only caused minor errors, such as conflicts between methodologies and uniformity across different sections of the project. For each of these issues, a common choice was made to ensure project-wide uniformity.

Requirements The requirements laid out in the brief for quality and manageability of the solution have been met where possible. Each function has been implemented well before moving on to the next, rather than implementing a larger but weaker solution.

Limitations The progress of the development, while promising, has been slow in parts due to the lack of prior knowledge about NodeJS, MongoDB, and ES6. This knowledge has been gained and effectively implemented as the solution has progressed.

4 Conclusion

The solution has fulfilled the vast majority of the requirements laid out for this assignment. The final result is a fully functioning restaurant find-and-review app that follows web app development standards. The software has a consistent code structure and JavaDoc style commenting and documentation.

A strong project has been created, with reliable, reusable and modular code throughout. Service worker aside, all of the objectives defined in the brief have been met to a high standard. Several extra features and functions have also been added to display originality and ability of the developers.

During the development of this release, it has become apparent that many popular web standards and technologies have developed quickly and, in some cases unforgivingly, by adding new features and removing those no longer deemed necessary. This has led to an abundance of largely useless resources and fora regarding the web standards that have been used in this project. This has provided an appreciation of the difficulties presented by working with Web technologies, and a greater understanding of their usage.

5 Division of Work

All three members of the group have contributed to this release of Restaurant Critique. The general structure was decided upon fairly by the group.

- Will lead the implementation and documentation of the Mongoose implementation and schema declaration and usage, 'Restaurants Nearby' page, 'Add a Restaurant' page and submission, User signup, login and management, database deployment, advanced RegEx search functionality, Socket.io integration, test data initialisation, and JS Documentation.
- Rufus lead the implementation and documentation of the initial database setup, text search function initialisation, and Progressive Web App features.
- Greta lead the implementation and documentation of the dynamic Restaurant page, Contact page, footer pages, and responsive web design for the application.

6 Extra Information

To run the project, simply run `npm i` in the solution directory, then run the `bin/www` file, either with Nodemon, running it through Intellij or `npm start`. The database has been deployed so does not need to be run locally or initialised, however an internet connection is required. The homepage is located at https://localhost:3000.

JSDocs have been created for this release, and are located at *IntelligentWeb\report\JSDocs\RestaurantCritique\2.0.0* They can be viewed by opening **index.html**.

Due to errors with the service worker, it is best to run the project *without the service worker* first to see the features throughout, and then enable it to observe its functionality afterwards. This is because a portion of the project's functionality is limited by the service worker.