COM3504 Intelligent Web - Restaurant Critique

Release 1.0

Authors: Will Garside, Rufus Cope, Greta Ramaneckaite

1. Introduction

This report discusses the structure and development of Restaurant Critique, a website and progressive app for finding and reviewing restaurants. The layout of the website, structure of the database, and the data storage information is discussed in the 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 padlock icon is used to signify where User login changes the User experience (UX): if it is locked, the User must be logged in to access the page; if it present but unlocked. Then the UX will be enhanced (e.g. more features or forms partially filled out with User details).

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 to make the diagram easier to read.

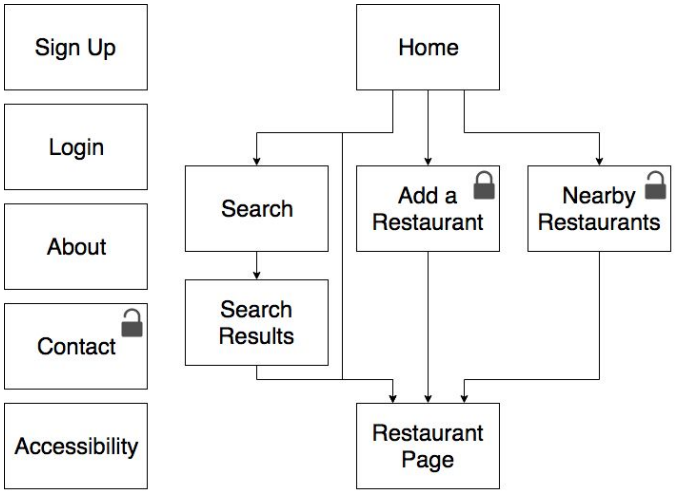

Figure 1: The layout of the website navigation

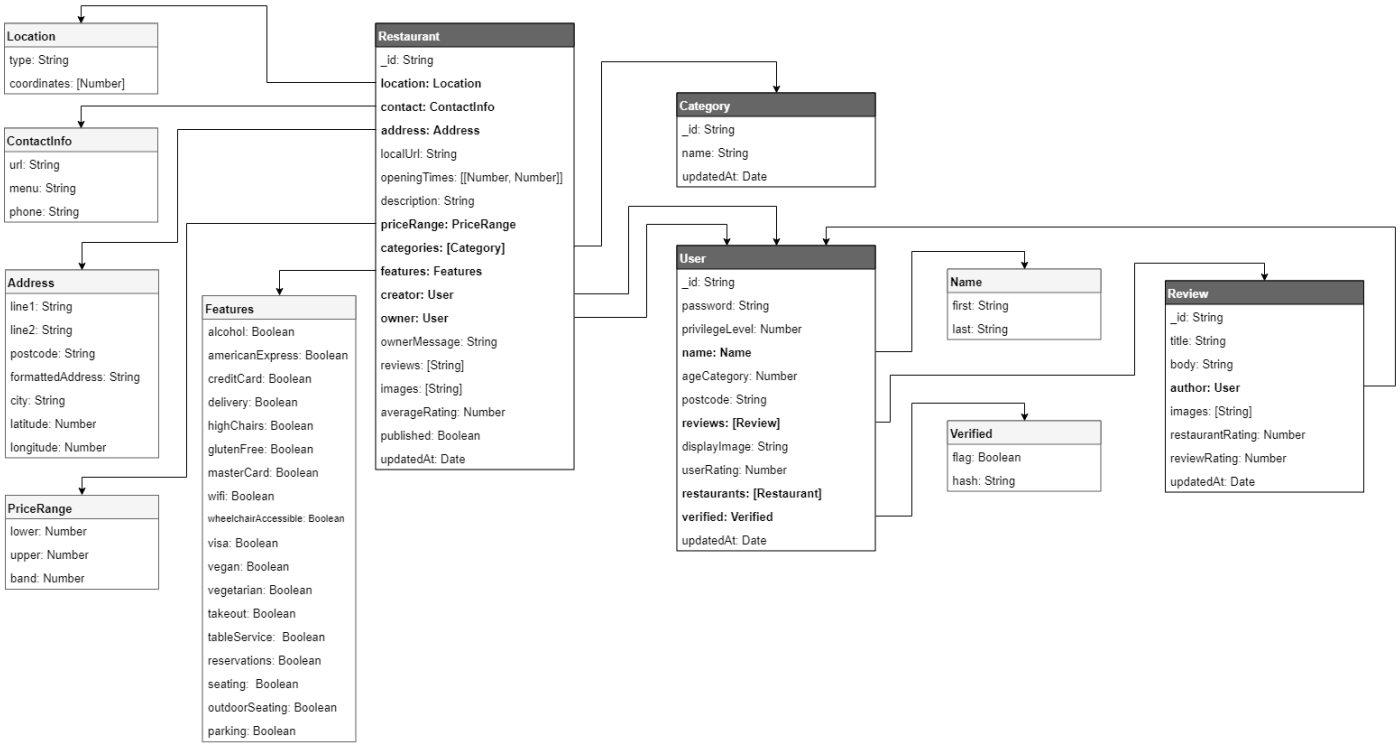

Figure 2: UML model showing the database layout

3. Individual Tasks

3.1. Searching for Restaurants and Displaying Results

Solution A User can search for a Restaurant on the home page 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 small - it's all 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 usage 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 there are plans to implement both of these functions to further optimise the search function. The schema.org tags are also yet to be implemented. The distance limit for finding Restaurants is currently set at 10 km, but will be User-definable in Release 2.0.

Requirements The requirement for returning results from free keywords has been successfully met, as well as finding Restaurants by distance. The Schema.org tags have still not been implemented, but will be added for R2.0.

Limitations The search function uses an AND operation between words 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 pages displays the Restaurant’s fields, as well as the Reviews that Users have left about it. A Review submission form is also on the page.

Issues The only issues with this page are the styling and the Reviews submission form (discussed in section 3.3).

Requirements The requirements defined for a ‘simple solution’ have all been met by the Restaurant page - other than Review submission. The requirements designated for a ‘more complex solution’ will be implemented for Release 2.0, amongst extra features.

Limitations All useful data is available to the User (apart from the average price rating), but the styling of the page is incomplete so can be difficult to extract.

3.3. Allowing Restaurant Reviews

Solution Reviews are stored as separate objects whose IDs are stored in a list in the Restaurant object. The Reviews are displayed as a section of each Restaurant’s display page. Each Reviews has its title, body, and rating displayed alongside the authors name and display image. Logged in Users will be able to access the form to submit a new Review on the Restaurant page, and the non-logged in User’s will see a message asking them to login.

Issues The images for each Review are not displayed, and are just represented by a black rectangle. This has an easy fix and requires some modification to client-side JS and will be implemented for Release 2.0. One of the larger issues on this page is the Review submission form: the form structure is on the page but no server-side functionality at the moment. The Reviews in the database are created by a database population script.

Requirements The requirement for having Review submission functionality is not currently met by Release 1.0.

Limitations The submission is not yet functional, images cannot yet be added to the Review form. No offline functionality has been added yet.

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. For Release 2.0, an attempt will be made to make the form easier to navigate and submit. There is no option to add a photo yet either, however it can be largely copied from the User signup page.

Requirements All of the fields for a Restaurant can be submitted via the form, apart from the image. Address lookup is implemented through either a postcode search or dropping a pin on a Google Map.

Limitations There is no client-side implementation at all for adding 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 in Release 1.0, to send a GET request to the server for information that is required on a page without a full refresh. ES6 has been used to take advantage of its

Issues The only issues experienced with this release's architecture are ensuring all team members correctly understand how the project is laid out and how and when to use technologies such as AJAX, POST and GET requests. socket.io is still yet to be implemented because the explicit need for it hasn't arisen yet, however it will certainly be present in Release 2.0.

Requirements AJAX has been used multiple times to great success in the solution. As mentioned above, the requirement for socket.io has not yet been met.

Limitations Currently the client-side JavaScript is also in the one-file-per-page format. Whilst this avoids complications for the development team, 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 will be discussed and evaluated for the second 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. This encourages the input of useful and valid data. The database has been deployed to one of MongoDB's free Atlas servers so that it doesn't 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 MongoDB's rapid development schedule has caused the addition and deprecation of lots of functionality in recent years, and as a result of this sources of information which are usually invaluable provide outdated methods and syntax. Unfortunately, MongoDB's own documentation is quite complex or sparse of examples in places.

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 only issues from Atlas can be if there is a conflict in the data required for testing purposes between members, however we are yet to come across this kind of issue.

Requirements: The database meets all the requirements as stated in the assignment brief.

Limitations: 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.

3.7. A Progressive Web App

Not currently implemented.

3.8. Quality of Solution/ Keeping Solution Manageable

Solution The codebase used in creating Release 1.0 has been through many iterations, with multiple core functionality re-writes. It is now largely 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 a few, if any, major issues in this release due to the thorough testing throughout the development. Omissions of requirements for the final product are not mentioned in this document, and minor issues have been discussed where relevant in this document and will be resolved by the release of 2.0.

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. Conclusions

Strong progress has been made on the solution, which is now more than halfway complete. There are few major tasks which still require completion, and site-wide styling still requires considerable work. 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.

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 nearby Restaurants page, new Restaurant submission, User signup, login and management, RegEx search functionality, database object creation, test data initialisation, and JS Documentation.
- Rufus lead the implementation and documentation of the database setup and search function initialisation.
- Greta lead the implementation and documentation of the dynamic Restaurant pages, Contact page and created the footer pages.

6. Extra Information

To run the project, simply run `npm i` in the solution directory, then run the `bin/www` file, either with Nodemon or by running it through Intellij. 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 localhost:3000. The **new Restaurant** page has not been added to the search results yet, so is accessible through the 'Add a Restaurant' link on the NavBar. The link will be removed for Release 2.0, as adding a Restaurant will not be a commonly used feature.

JSDocs have been created for this release, and are located at *IntelligentWeb\report\JSDocs\RestaurantCritique\1.0.* They can be viewed by opening **index.html**.