COM3504 - The Intelligent Web
Harry Howarth, Victoria Neal, Adam Orr
Fab.io

Report Documentation

The project aim was to build a progressive website that can be used to find nearby restaurants, and be able to read and write reviews about the restaurant. A user can sign up, login and logout. When a user is logged in they can create a new restaurant for the system if it does not exist, and also when logged in the user can write reviews for restaurants. There is also forgotten password functionality. When the user logs in they are redirected to the last 'important' (index page, the search and results page, and a restaurant page) page they viewed to improve usability on the site. The whole website is highly responsive for ease of mobile use. It also caters for 'on-the-go' users making use of service workers and IndexedDB to provide functionality even when offline. We decided to make the project more authentic by creating a Dine Hard Facebook page, Instagram account and Twitter account, and linking these accounts to icons on the footer of the website. We decided to use a trello board of user stories for each element of the project. This methodology allowed us to know exactly where we up to and what was left to do at any point during the project. We chose a red colour scheme for the website as this colour compliments the background image on the homepage and compliments the text colours chosen.

Diagram - The page diagram is shown in the Appendix. The homepage is accessible from every page as it is linked in the main navigation bar at the top of every page. Offline data is put into IndexedDB through a service worker and is sent to server when back online which is then stored in the MongoDB, online data is stored in the MongoDB when it is submitted.

1.1.1 Searching for restaurants and displaying results

Solution - A user is initially able to search using a postcode, their location or keywords on the homepage. This is the first step a user must take to return restaurants in the results, so it is therefore logical to have this search bar on the homepage, rather than having to navigate to a different page before entering any search data. The restaurant results page layout consists of additional search options in a bar on the left and results on the right so that the space is utilised. This design optimises page space because if the additional search options were at the top of the page it would look too cluttered. An advantage of having the extra search options on the left is that they can be fixed at the top of the screen so that it is always accessible. The search works by first checking whether the input is a postcode or a keyword, then queries the database to return appropriate restaurants. The homepage and the results page are both highly responsive for mobile use. A user can search for keywords on the results page as well as using their location or a postcode. The results page is scalable as many restaurants can be added to the database and will be shown when they meet search criteria. The second level of search is implemented in AJAX and allows for 'search as you type'.

Issues - The main issue was discerning between a restaurant keyword and a location within the search criteria for a restaurant. Another issue that we had was making the decision that a user does an initial search and then filters afterwards based on things like cuisine and price. The alternative would be a messy homepage with dropdowns and radio buttons to filter straight away etc. Other than our own opinions, this decision was based on looking at similar sites such as TripAdvisor and JustEat. [1] [2]

Requirements - All requirements have been met for this section as a user can search by a postcode, location or key words. The corresponding restaurants are pulled from the database and the correct items are displayed on the results page for the user. A user is able to search for a list of matching restaurants by using keywords and/or tags specific to the search they are running. The keywords can include the type of cuisine of the restaurant, or any keywords used when added to the restaurant on the restaurant's creation. Their search will return all restaurants that match any of the keywords specified in an order specified by the user but by default it will return them from closest to furthest away within the distance parameter specified. A map is also shown that displays the locations of all the results returned from the user's search and also supplies a brief description and link on each restaurant when the appropriate pin is clicked on the map. The map is centered on the user's location and the markers are only shown for restaurants within the users desired search radius.

Limitations - When searching for restaurants entering certain strings causes a limitation upon the results page, and will not return all the appropriate resulting restaurants.

1.1.2 The restaurant page

Solution - The restaurant page provides details for users about the restaurant including address, telephone number, website, price range and photos. There is also a map that shows the restaurant location. When a user is logged in they are able to write a review for the restaurant including a star rating out of 7, a review description and photos. We decided to have 4 elements on the restaurant page; the details, the photos, the description and map, and the reviews. This splits up the page well so that a user can follow the page down for more information. In addition, we wanted the reviews to be updated in real time so that every user could get the most up to date information, this was resolved by use of socket.io.

Issues - An issue that we had was the organisation of all the information for a restaurant and how to display it in a user friendly way.

Requirements - A requirement was that this page contains all the relevant information for a restaurant, which the page's design allows for and shows the user in a well formatted layout. A small map showing the location of the restaurant, is shown below a slideshow of both official and review related images, and can be made full screen, this was implemented using google maps. All the needed information about a restaurant is displayed nicely formatted to the user when viewing the restaurant page including the distance from the user's location to the restaurant's location. Also displayed is the restaurant's rating score which is a average rating score given from all the reviews on the restaurant. A description written by the creator of the restaurant can be viewed alongside the map and provide an overall description for the restaurant. The map presented allows the user to review the distance from their location to the restaurant by combining the map with the restaurant information. Also as each restaurant has a unique url, it is possible for the user to bookmark a restaurant page and come back to it at a later date, which increases the usability of the website as the user doesn't have to research for the results they gained previously.

Limitations - A user cannot determine which photos were uploaded with which review as all photos associated with a restaurant are uploaded into the slideshow on the restaurant page. But this does allow for an easy way to view all the photos that relate to a restaurant without having to scroll through masses of reviews.

1.1.3 Allowing restaurant reviews

Solution - When a user is logged in they can write a review for a restaurant. The solution also allows users to view their profile to see restaurant reviews that they have previously written, and they can delete reviews that they have written here. A review is created using a modal, so when the write a review button is clicked, the modal pops up with details for a review to be entered. This means that the review can be seen immediately after it has been submitted, without being directed back to a different page.

Issues - There were issues with uploading photos for reviews as the package size was too large in that the JSON being parsed was larger than what the server would allow.

Requirements - All requirements have been met as logged in users can rate and write a review for a restaurant with a photo if they desire. A review cannot be written by a user who is not logged in and a user must login to an account registered to the website in order to write an review for a restaurant. A user is able to apply a rating to the restaurant, optionally add a photograph to the review from their device and also write a review with no limit on the length. The review is uploaded to the server and can be seen instantly by other users as the restaurant page is updated using socket.io. When a user is offline at the time of posting for the review, the review will be saved and will be resubmitted when the user reconnects to the internet.

Limitations - Only one photo can be taken at a time, and all photos for reviews have to be taken by the webcam, so if a user does not have a webcam or camera on the device that they are using then they cannot upload a review with a photo. However, they can still upload a review without a photo.

1.1.4 Creating a new restaurant page

Solution - A new restaurant is added through a form when a user is logged in, as the navigation bar only has the "create new restaurant" button when a user is logged in. There are many fields in the form to ensure the user adds enough information about the restaurant to create a suitably detailed page.

Issues - We decided that the cuisines should be hardcoded in because there is no cuisine table in the database as that became redundant. Using WEBRTC was initially a challenge but after further research we decided to use [3] as a guideline for how to approach the task.

Requirements - In order to fulfil this element, it must be possible for a user to add a restaurant to the website, this has been achieved. When a user starts to type the address, a drop-down list of relevant addresses appear where a user can select the correct one and then the remaining address fields are filled but can still be edited, this was completed using google maps. Only a logged in user is able to create a new restaurant page and add it to the server so that it will appear in future searches by other users. All fields on the restaurant creation page are mandatory and must be filled in by the user for the restaurant to be added to the server. Official photos can be added to the restaurant either from saved photos on the device or by taking a new one using any cameras connected to the device; multiple photos can be uploaded for the official photos of the restaurant. As mentioned previously, when adding an address the user is assisted by a drop-down list of addresses that match the user's input and applies the correct text to the appropriate address fields on completion. Again, if the user is not connected to the internet when submitting a new restaurant form, they will be informed and the information will be resubmitted to the server when an online connection is remade - form validation takes place offline as well so any issues with the restaurant creation form will be made aware to the user before submitting the data.
Limitations - In the unlikely circumstance that a user is using a device which contains no photographs on it, and also doesn't have a webcam the user would be unable to create a new restaurant as it requires at least one photo to be added upon creation. Due to the very unlikely chance that this will occurred it was not considered a valid use of our time resources to work around this limitation.

1.1.5 The Server Architecture

Solution - This project uses the web application framework Express for NodeJS. This allows HTTP requests to be simply handled by the server logic written in NodeJS.The data is passed through the system as JSON. Currently we have implemented searching for a restaurant, creating new restaurant, and reviews with appropriate controller methods for each. We have used both GET and POST for sending requests to the server from forms on our webpages.
Issues - The main issue we came across was finding appropriate areas to use AJAX and Socket.io for, however we used Socket.io for reviews so that they are updated in real-time.
Requirements - The main requirement was to use express and NodeJS for the server architecture, we have clearly met this requirement as both express and NodeJS have been used to create the project from the start. AJAX is also used throughout the solution to transmit data for a number of the forms within the solution; the use of AJAX allows for a more sophisticated server architecture and a secure solution. Indeed this also lends itself to allow all data communication to be implemented using JSON. Socket.io has also been used for the review portion of the solution so that each restaurant always has up to date reviews and the review section will be updated as soon as a new review has been written.
Limitations - The ease of adapting the solution is high, as more elements can be integrated into the website in the same way as the current elements.

1.1.6 The Database

Solution - As required by the assignment brief MongoDB was used. As outlined in the lectures the NodeJS package Mongoose [4] was used for creating database schemas and models in MongoDB. There are three model schemas: Users, Restaurants and Reviews. Users and Restaurants are schemas that are completely reusable in any system because they don't depend on any other schema. Users can be uniquely identified by their email address or username as well as a database ID. The only way to uniquely identify Restaurants, is through its ID. This is because we decided no other piece of information in the database is enough to uniquely identify them. Reviews, naturally, require an id from a restaurant and a username to be associated with them. However, this shouldn't be considered a limitation of our design because in actuality reuse is possible as long as you have a username ID field, that we use for restaurant could be anything that you create a schema for. Our solution has advantages over other solutions such as being as readable and simple to use. However, one would think that solutions for this section of the project would not vary too much.
Issues - A small issue that we were faced with was that windows users found the MongoDB installation and configurations difficult. We overcame this problem quickly and have managed to use MongoDB will no issues since.

Requirements - The brief requirement given for this section was 'The database of restaurants must be implemented using MongoDB and contain all the information relevant to each restaurant; it must allow the searches above'. Hence, we have fully satisfied this requirement. As the database has been created and handled using MongoDB and Mongoose and contains all the data needed to allow the solution to function appropriately.

Limitations - An exceptional circumstance that we thought of when creating the Restaurant schema is whether addresses are enough to uniquely identify a restaurant or should two restaurants be allowed to share an address. Given the simplicity of creating and modifying mongoose schemas, this solution could easily be adapted for other requirements.

1.1.7 A Progressive Web App

Solution - The entire design of our site is to be responsive around mobile. We cater for users on the go by using a service worker. This handles caching pages so the user can use the site when offline. It also handles the requests so that if a request fails when submitting a new review or creating a restaurant because the user is offline, it is stored in IndexedDB and the request is sent off and hence then review/restaurant created the next time the user is online and does something on the site. We also included useful messages that are controlled by the client-side javascript, that inform the user of the behaviour of the site when they are offline. For example, when creating a review and offline the user is informed that submitting the form will still work.

Issues - One issue has been storing the data that a user has entered when they are offline. This was solved by using a google library for IndexedDB which radically simplified storing information in IndexedDB.

Requirements - Pages are cached when visited and will be shown if the user navigates to them when they are offline. The user is notified that they are offline with a banner on various pages. On the search page and login page the forms become disabled so that the user cannot attempt search or login. The app obviously does not crash. This shows that all the requirements have been met for this section. Additionally, from other sections that are PWA requirements - when data is entered into fields for creating a restaurant and writing a review, the data is stored in IndexedDB through the service worker to be submitted when the user is online.

Limitations - If a user has not yet visited the page then it won't be cached so the user can't still use that page and its functionality. If a user clears their cache and then revisits the website then this is also an issue if they are offline.

1.1.8 Quality of the Solution

Solution - Our solution has made use of all required technologies including AJAX, socket.io, WEBRTC and express. Hence, we have met the learning objectives of the course. An example of where we went 'above' the assignment description, while keeping the workload manageable, is that we decided to implement the 'view profile' element for a user as we wanted to make sure the website was functional from a user's perspective, having the view profile would allow a user to keep track of their reviews and you can view other peoples profiles. On the results page, the restaurants can be sorted by price, or distance. Also upon logging in, the user will be returned to the previous page that they visited so that they don't lose any search results or the restaurant that they were viewing before logging in. The system also includes a forgotten password system to allow users who have forgotten their password to reset it, at the moment the generated link is output into the console - but obviously in a final solution the link would be emailed to the user's email address. When accessing the link, the user is given a form to update their password to a new one, which will then allow them to login into the system using their new account details; the link expires after one hour of generation.

Issues - No style guide was given so we built up the layout using common, standards and our own knowledge. An issue was that we did not know if the website is meant to be incredibly professional and clean or if it is meant to be user oriented with extra areas for ease of use.

Requirements - Overall, **every single** requirement has been completed, many with more features than specified. This has made our quality of solution very high as we also incorporated user friendly elements. The system makes good and effective uses of AJAX, socket.io and also WebRTC, each of which is used appropriately to improve the overall quality of the solution (the use of each have been mentioned in previous sections). The solution also has an easy to use and understand stylish design to improve a user's experience when using the solution and also to provide a high number of functionalities in a compact manner. The login management is handled using a hashing process allowing for a more secure solution and keeping the user's personal information safe is of the highest priority especially in light of recent personal information scandals. This is the same hashing system that many high profile companies, such as Twitter, use. The solution as a whole is scalable to allow for thousands of users, restaurants and even more reviews; the interface and backend both scale appropriately when given larger amounts of data to handle.

Limitations - Any limitations of the site have been included and detailed in their respective section above.

Conclusions

As a team, we have produced a website that has a database of restaurants that can be searched for through keywords and a user location or postcode, where a logged in user can rate and write a review for a given restaurant. **All** the requirements have been satisfied and the website functions successfully. We have implemented extra functionality, while keeping the significant amount of work for this assignment manageable.

Division of Work

As a team we decided to meet to work on the project together regularly - most Mondays, Wednesdays and Fridays. This allowed for constant discussion about the project and kept all team members up to date. All three members of the group contributed equally to the assignment solution. The solution was designed jointly and then for each section we discussed and implemented solutions together, its associated documentation and contributed to the writing of the final report. In particular: (using the numbering scheme from the assignment sheet)

- *Harry - worked on 1.1.1 (searching), 1.1.2 (restaurant page), 1.1.3 (creating reviews), 1.1.4 (creating a new restaurant), 1.1.5 (the server architecture), 1.1.6 (the database) and 1.1.7 (a progressive app).*
- *Victoria - worked on 1.1.1 (searching), 1.1.2 (restaurant page), 1.1.3 (creating reviews), 1.1.4 (creating a new restaurant), 1.1.5 (the server architecture), 1.1.6 (the database) and 1.1.7 (a progressive app).*
- *Adam -worked on 1.1.1 (searching), 1.1.2 (restaurant page), 1.1.3 (creating reviews), 1.1.4 (creating a new restaurant), 1.1.5 (the server architecture), 1.1.6 (the database) and 1.1.7 (a progressive app).*

Section 1.1 - 1.8 were completed by all three members of the group, equally. The final document was jointly written and finally edited by the whole group.

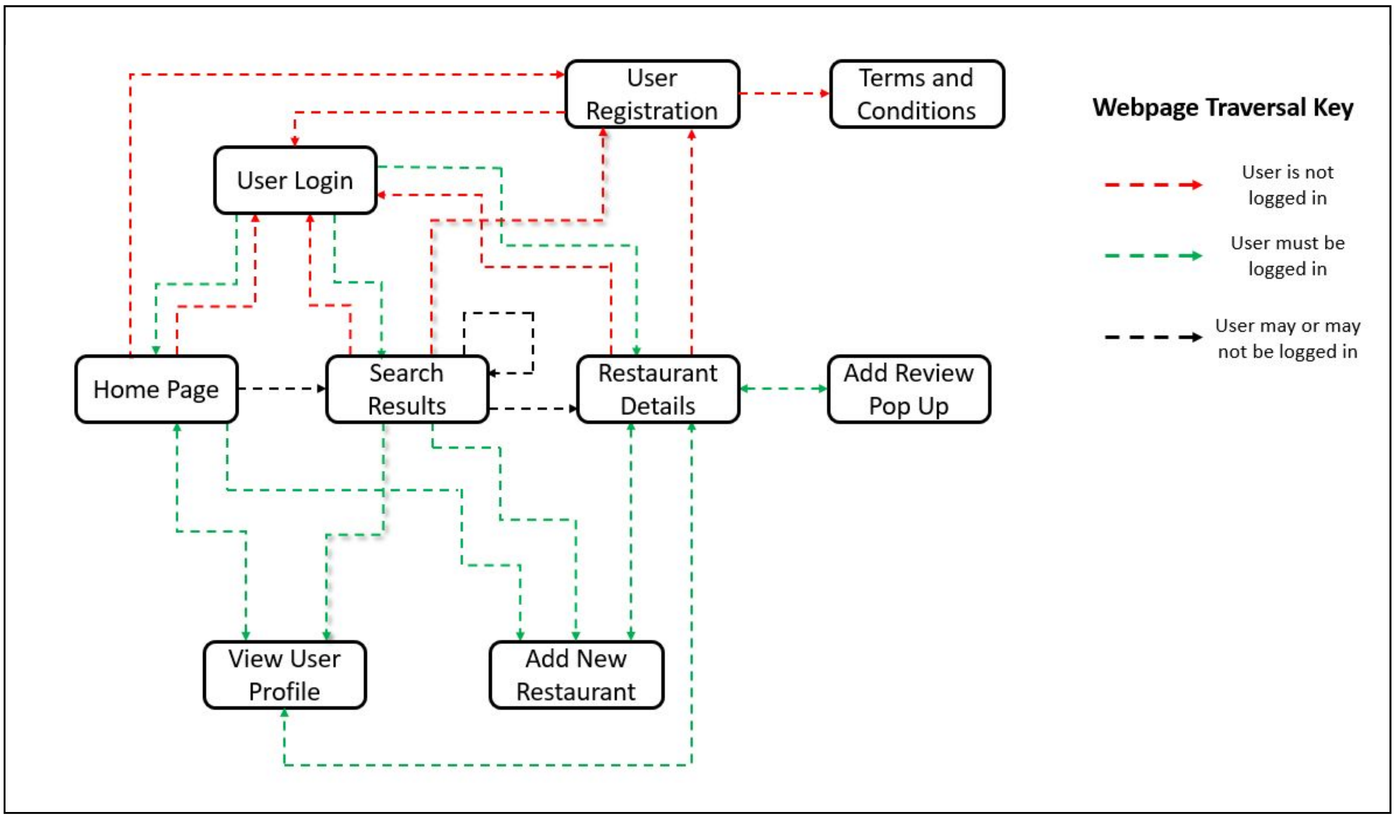
Extra Information

Before the program is run, the connection to mongoDB must be made, use mydb as the database [5]. The database must then be seeded using 3 commands “node models/seeding/users_seed.js”, “node models/seeding/restaurant_seed.js” and “node models/seeding/reviews_seed.js”. Then the node modules need to be installed with the command “npm install” then run node on app.js. Then the program can be run and you can navigate to localhost:3000 on a web browser.

Bibliography

[1] Trip Advisor, https://www.tripadvisor.co.uk/
[2] Just Eat, https://www.just-eat.co.uk/
[3] WEBRTC, https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API/Taking_still_photos
[4] Mongoose, http://mongoosejs.com/
[5] MongoDB, https://docs.mongodb.com/v3.0/tutorial/install-mongodb-on-windows/

Appendix


User Registration
Terms and Conditions
User Login
Home Page
Search Results
Restaurant Details
Add Review Pop Up
View User Profile
Add New Restaurant
Webpage Traversal Key
User is not logged in
User must be logged in
User may or may not be logged in