Aatina Punjabi – ap14974

Web technologies – prelim report

For this project, I have worked on my own. I will talk about each of the marking sections and what I have done for it, as well as the technologies used.

My website is a recipe website called "Earthly recipes". Users can upload and view recipes, add them to favourites and talk to other users through the chat room.

There is a user created called "ian" with the password "password" if you wish to log in and test out features. Otherwise, you can create a new account (with a unique username).

Technologies used

Node modules:

- Express – server side framework
  - express-fileupload – uploading images on to the server
  - express-session – creating sessions for user login etc.
  - express-ws + ws – websocket library for chat room
- sqlite3 – embedded database engine
- ejs – templating engine
- less + less-watch-compiler – dynamic stylesheet that compiles into css
- node-time-ago – used to convert time into a time-ago format
- showdown – markdown to html converter for recipe bodies
- nodemon – automatically refreshes site when changes are made to code (for development only)
- bodyparser – Parse incoming request bodies

Other:

- jquery – for general javascript
- ajax – for client side web requests
- Twitter bootstrap – responsive front end framework for layout organization and styling (mostly forms and button)

HTML

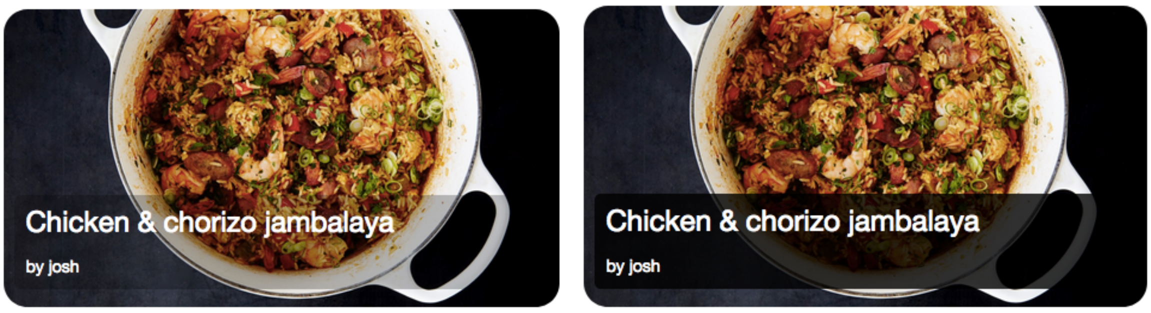
For HTML I have used a templating engine – ejs, to avoid repeated code. My code is well organized into a views folder, separated into partials (sections that are used repeated through the website) and pages (full pages of the website).
I have continuously validated my html throughout the process using an online validator [1]. I have used some HTML5 features such as nav, header and footer tags.
I do have some script tags within the html pages, but these are only small scripts specific to that page only, so there was no need to separate them into another file. My client side js files are stored in *public/js*.

Estimated rank – A

CSS

I used the LESS dynamic stylesheet language and compiled it into CSS. This allowed for nested classes and variables. My main LESS file is *styles.less*, where I have specified general styling for the website as well as variables for colours that are often used. This helped avoid mismatching of colours and easy changes in case I would like to try different colours.

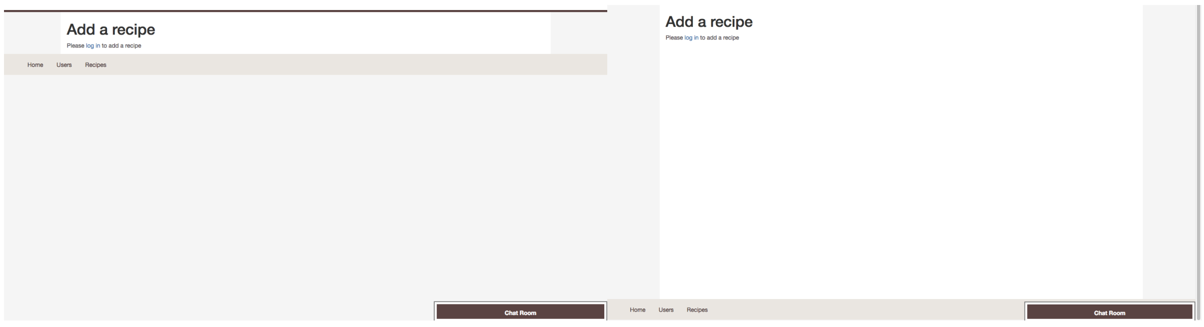
I have used some CSS animation throughout the website. There are quite a few fade ins/outs such as recipe views on the main page, hover over the navbar links and hovering over the summary for on the recipe pages. There is a spin animation for the loader used on the "View all recipes" page. A wiggle animation is used for icons on the "All users" page. The chat room found at the bottom of most pages slides up when clicked on.



Example of fade in for the black box

My LESS files are found in my main folder, and the compiled CSS files are stored in the *public/css* folder. They are spilt into files according to what section or page they are used on.

I have used css quite thoroughly and have followed many tips and tricks online. For example, extending the main container to push the footer so that it stays at the bottom of the screen, even when the content is too short to do so.



Left: footer not staying at the bottom of the page, Right: footer at correct position

Estimated rank – A

Javascript

I have used javascript and jquery significantly throughout my website. It was used quite frequently for appending or updating elements. I have also used **ajax**, to do client side database queries, without needing to refresh the page.

An example of a client side feature is **infinite scrolling**, which is used in the "view all recipes" page. The page loads 6 recipes when it is first entered and 4 more are loaded each time the user scrolls to the bottom. This is done using ajax to query the database on the client side.

The recipe view boxes are **dynamically generated**, and a query to the database is made each time, with an increasing offset variable to ensure the order of the recipes is correct. To display this correctly I have added a delay with a loader, as the new recipe views were generated too quickly, but the delay can be removed. This

Another example of client side ajax is checking if the username is taken in signup page. When a username is entered and the form is submitted, the ajax prevents the form being submitted and queries the database to check for a matching username. If it's taken then the form is not submitted.

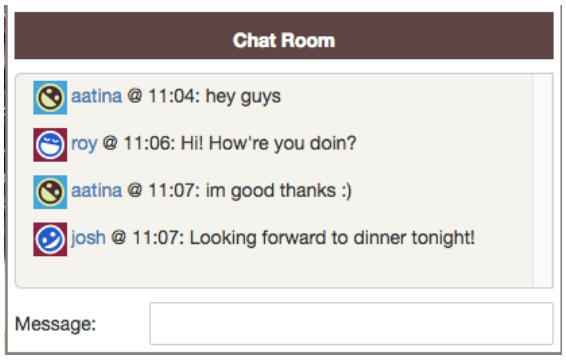
On all recipe pages, there is a star button to allow users to add a recipe to their favourites, so they can then view it on the favourites page. This is also done using an ajax query. It inserts the *user_id* and the *recipe_id* into the favourites table.

Estimated rank – A

Server

My main feature for the server side is using **web sockets** for the chat room found at the bottom of each page. The client side code for that is found in *public/js/websocket.js* and the server side code is near the top of *index.js,* where most of my server side code is found.

The chat room allows for all users to post a message to the server, and the server then broadcasts the message to all logged in users that have been connected to the web socket as soon as they log in. To test this, you can log in as two separate users on two different browser windows.



aatina @ 11:04: hey guys

roy @ 11:06: Hi! How're you doin?

aatina @ 11:07: im good thanks :)

josh @ 11:07: Looking forward to dinner tonight!

Chat room window allows for multiple users to have a conversation

For server side organisation and to avoid "callback hell", I use **Promises** to do asynchronous code. Promises wait for a reply from the database before allowing the code to go forward. I used a **wrapper class** [2] for sqlite database (found in *db.js*), to allow for easy querying throughout the code. The connection is neatly opened and closed, and queries are made through this database class.
The query function in this class takes the query string and a set of parameters for parameterised queries (see Security section).

"store.js" is where all the functions related to storing to the databases.

Estimated rank – B

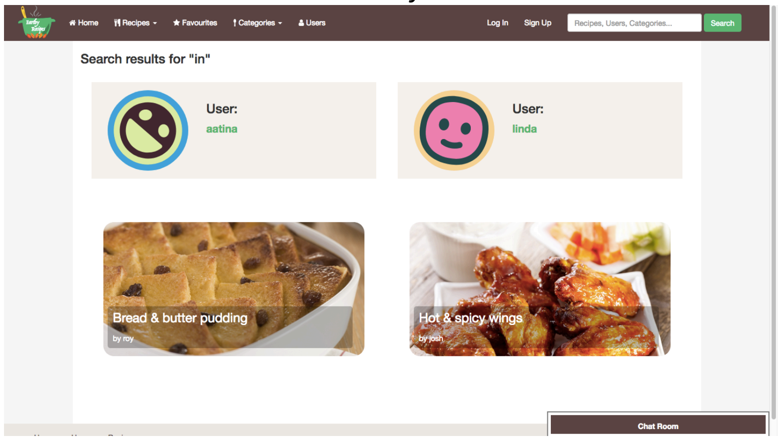
Database

The database is an embedded sqlite database – "*recipe_site.db*". It has 5 tables:

- *Users* – Main table for all the users in the website
- *Recipes* – Main table for all the recipes in the website. Includes *user_id* (id of user that uploaded this recipe) and *catergory_id.*
- *Categories* – the 6 categories that each recipe can fall into.
- *Favourites* – Links
- *Recipe-ingredients* –

My table is a **relational database**, as they are linked using key fields. For example, the favourites table is simply a link between user and recipe. The recipe-ingredients table links every ingredient to a recipe, etc.
I used **foreign keys**, e.g. *user_id* in the recipes table, so if the user is deleted from the table, all of their recipes will also be deleted. This ensures **data integrity**.

My search function is a long query that returns all users, recipes and categories that match the search query. The query does not need to match the name exactly (except for categories). It uses **unions** to get data from different tables. This query can be found in *search.js*. I have also used **inner joins** to return results from different tables by matching them using their ids. Multiple examples of this can be found in *index.js*.



Searching for "in", returns users and recipes that contain "in"

Estimated rank – A

Aatina Punjabi – ap14974

Dynamic Pages

Most of this has been explained in the previous sections. All of my pages are dynamic, and used the templating engine **ejs** to generate html. Parameters are passed into every page and are used to display the content on the page.

The best examples of dynamic pages are those that display recipe view boxes, for example the home page or favourites page. Partial html code in "*views/partials/recipe_views.ejs*" is reused on multiple pages, where the **parameters** for the recipes are sent in from index.js. This way different recipes can be sent in different orders and it will not affect the html, as long as the parameters have the same names.

The chat room is also dynamic, and generates html when a message is broadcasted by the web socket. The infinite loop requests for recipe data using ajax, generates html using j-query, and appends it to existing html classes. The search page receives one JSON output and sorts it into users, categories and recipes and generates html accordingly. Using ajax, the client requests from the database and generates html, for example in the signup page, where an error message is generated depending on the result of the ajax query.

Estimated rank – A

Security

I have taken some measures to address the security vulnerabilities. **Cross site scripting** (xxs) is prevented by sanitising all user inputs. This ensures that users cannot enter malicious scripts into inputs such as the search bar. An example of this checking for tags in the inputs and replacing them with safe characters, done in the chat room input.

SQL injections are prevented by using **parametrised queries** whenever a query is made to the database, preventing malicious queries. I have also prevented **click jacking**, by adding a meta tag in the head, so it blocks my website being used in an iframe on other websites.

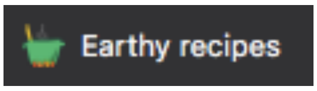
The passwords are salted and hashed before being saved into the database. This ensure that if there is data breach, then the passwords are not leaked.

PNG/SVG

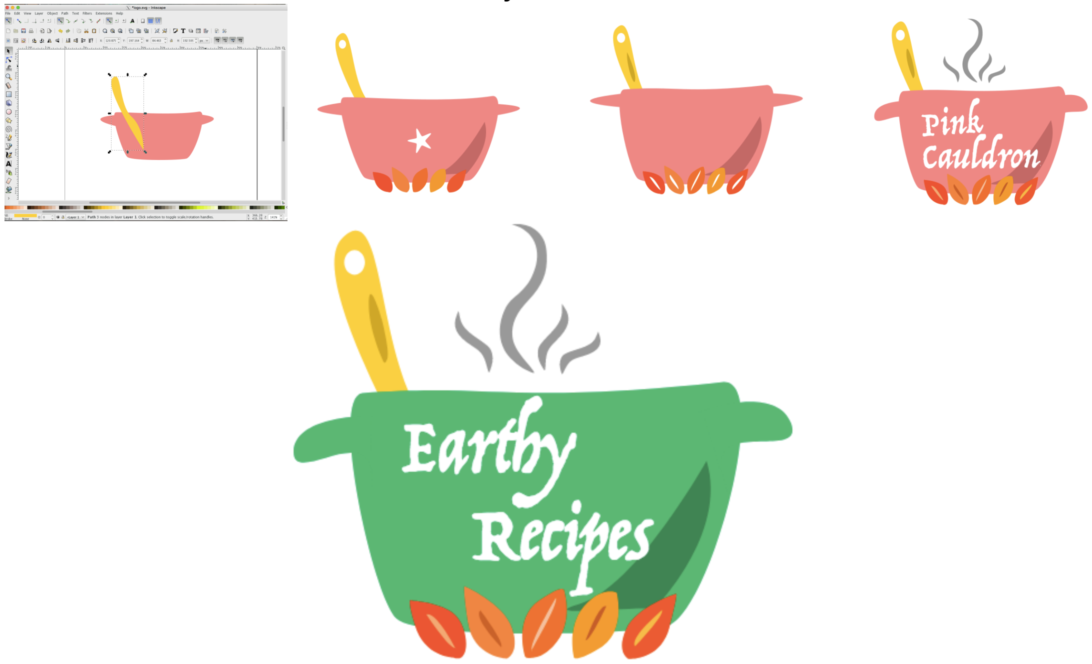
I have made quite a few of the images used throughout the website. The main SVG used was the logo, a vector drawing, which was drawn using **Inkscape**. The logo is a green cauldron, with a yellow ladle inside and orange fire (made to look like leaves) below it.

I used the Bezier tool extensively to draw out the curves. I also used paths quite often to "union" or "difference" objects together. The logo without text was also used as a favicon for the website.

Aatina Punjabi – ap14974

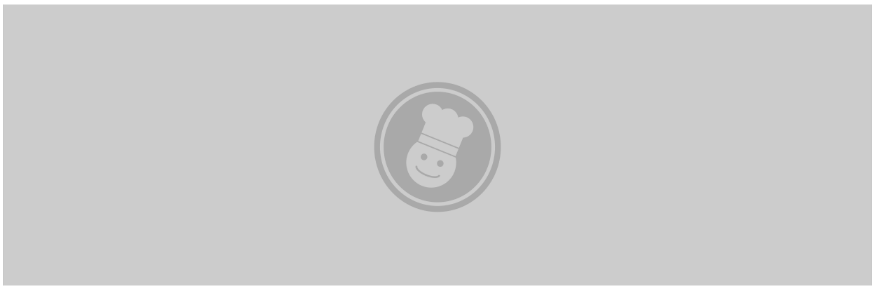


Favicon for the website



Steps of how the logo was created. It was initially pink, but changed to green to fit the website.

A PNG image used in the website is the default recipe image used when a recipe is uploaded without an image. This was created using simple shapes and paths to create the chef logo.



Default recipe image

Finally, I vectorised a jpeg image found online and edited it slightly to create a banner for the front page that suited my website. The SVG of the image can be found in the *public/images* folder, however I decided to export it as a PNG to be used in my website.

The banner used on the home page

Estimated rank – B

Depth

Here are some other features that have not been previously mentioned:

- User avatars are uniquely generated by their email [3].
- Users can upload their own recipe images, and these are stored in *public/images/recipes_images*, with a uniquely generated string of 11 characters based on the recipe ID.
- Session variables are used to store messages across pages when users are redirected to another page after completing an action – e.g. after uploading a recipe, users are directed to the home page and a message is generated to inform them of the success.
- Some forms use different methods of input, such as the bootstrap datetimepicker for birthdays when creating a user (max date is today), and the slider used to input cooking time for recipes.
- There is a 404 page if an incorrect route is entered in the url bar.
- There is a page to view all users and their bio. There's also a page to view all recipes of a certain category.
- Trailing spaces are removed from the username and email when creating a new user and usernames and emails are all converted to lower case. This is to ensure uniqueness.
- Passwords are checked against a confirm password field to ensure the user knows their password.

Estimated rank – ?

References

[1] Online validator: https://validator.w3.org/

[2] Node.js, MySQL and promises: https://codeburst.io/node-js-mysql-and-promises-4c3be599909b

[3] Uniquely generated user icons: http://vanillicon.com/