CM2020: Agile Software Projects

“Food Haven”

A web application to discover recipes

Agile Team 67

Alysha Hizam, Hsu Chee Wei, Nicholas Chow, Kumar Lekha shri, Jasmine Goh

Table of contents

1. Abstract **4**
2. Outcomes **4**
3. Introduction **4**
4. Planning and Research **5**
4.1. Resource and time allocation 5
4.1.1. Front-end 5
Front-end time and progression log 5
4.1.2. Back-end and Middleware 7
Choosing an appropriate DBMS 7
Back-end time and progression log 7
4.1.3. Breakdown of the team 8
4.1.4. Project Management Tools 8
Code Repository: Git and GitHub 8
To monitor progress: Gantt Chart (Jira) & Kanban Board (Asana) 9
Team Communication: Discord & Telegram 10
Collaborative Documentation: Google Docs 10
Feedback Collection: Google Forms 11
4.1.5. Dataset 11
4.2. User recommendation algorithm 11
1.1.1. Collaborative filtering 12
1.1.2. Content filtering 12
4.3. Collaborative filtering vs Content filtering 12
4.4. Implementation methods 12
Further improvements on the algorithm for future iterations 13
5. Design and Development **14**
5.1. User accessibility 14
5.2. User testing 14
5.3. Design - first iteration 14
5.4. Overall - second iteration 18
5.5. SUS (System Usability Scale) survey 20
Comparison of SUS Test Results: 20
Overall Evaluation: 21
5.6. Initial Design (Figma high-fidelity prototype) 21
1.1. Production prototype - First iteration 22
1.2. Secondary design 23
5.7. Presentation layer 25
5.8. Structure and Implementation 26
5.9. Search page and functionality 26
5.10. Passport.js 1

5.11. Sign up and sign in..... 1
5.12. Modules..... 1
Web application framework.....1
5.13. CSS Framework..... 1
5.14. Templating language.....1
5.15. SQLite3.....1
6. Evaluation.....1
6.1. Front-end..... 1
6.1.1. Future implementations (Front-end)..... 1
6.1.2. Technical issues (Front-end).....1
6.2. Back-end.....1
6.2.1. Conceptual issues.....1
6.2.2. Future implementations (Back-end).....1
6.2.3. Technical issues (Back-end)..... 1
7. Conclusion and summary.....1
7.1. Front-end..... 1
7.2. Back-end.....1
7.3. Overall..... 1
8. Individual Reflection.....1
9. References..... 1
10. Appendix..... 1
10.1. Work history.....1

1. Abstract

Our recipe-sharing website serves as a platform for users to share, discover, and engage with culinary creations. The website seeks to create a lively community where people can share their culinary talents, discover new dishes, and interact with other like-minded food enthusiasts. The website has features including a user-friendly interface, simple navigation, and tailored recipe suggestions based on user preferences. Users can share their recipes, like them, and comment on other users' recipes. A secure and encouraging community atmosphere is promoted by the user authentication processes as well. Overall, our recipe-sharing website strives to offer a vibrant and welcoming environment for culinary enthusiasts to explore, develop, and share their love for food.

2. Outcomes

The primary objective of our team's current project is to enable users to create and log in to their accounts on our website. After successfully logging in, users will find themselves on their personalized recommended feed, which will serve as their new homepage.

On the homepage, users will have the opportunity to explore a wide array of recipes and employ filters to refine their choices. These filters, accessible via the navigation bar, allow users to search for their desired recipe based on their categories or ranks, with rankings determined by a like-based system that we've implemented. Furthermore, a search bar is also available to aid users in finding their desired recipes with ease.

Additionally, there will be a designated page where users can craft their unique recipes. A dedicated profile page will also be available for users to conveniently access and manage the recipes they have both created and liked. Moreover, users will have the ability to access a settings page, granting them the flexibility to make adjustments to their account details as needed.

3. Introduction

The successful development of our web-based recipe-sharing application is the main goal of this project phase. We frequently went back to our initial concept paper which served as the basis for our project throughout this development process.

This report covers a wide range of studies, including user testing that uses heuristic evaluation methods. It also gives a brief review of the technologies chosen for the project, explaining the benefits and drawbacks of each. Our report concludes with a thorough analysis of the work completed, covering both the conceptual difficulties resolved throughout the project's

development and the technical obstacles encountered during the back-end and front-end development.

4. Planning and Research

4.1. Resource and time allocation

4.1.1. Front-end

Our agile software development project was centred on establishing a recipe-sharing website that relied heavily on efficient resources and time allocation to ensure a smooth project progression. We set aside the first two weeks of the project to code the front end. At this point, we started building up a GitHub repository and started working with HTML, CSS, Bootstrap, Javascript, and the EJS framework.

```
<%- include('navigation.ejs') %> <%- include('sidebar.ejs') %>
    <div class="page-content page-no-sidebar">
      <div class="container-lg pt-4">
        <% recipes.forEach(function(recipe){ %>
          <%- include('home-card.ejs', {recipe}); %>
            <% }); %>
              <div class="recipe-placeholder"></div>
```

Figure 4.1.1.1 Code snippet of front end framework

Front-end time and progression log

Task	Start Date	Duration	Complete	Incomplete
Food Haven's initial front-end design and planning	18/07/23	2.5 hours	Website layout and flow	Code not implemented
Setting up of libraries and coding environments	28/07/23 - 01/08/23	8 hours	Setting up of node.js project, connecting HTTP server to the SQLite3 database, parsing CSV data into the	Implementation of project code

			database.	
Login and register pages and user authentication	14/08/23 - 18/08/23	6 hours	Login and sign up page, form.	Connect to database and functionality
Navigation bar, sidebar and burger menu components	08/08/23 - 17/08/23	8 hours	Navigation for both the landing page and the homepage after login along with the sidebar and burger menu	Search bar functionality and linking of the pages to the navigation hrefs
Filling up of static page content such as(landing page, about page)	14/08/23 - 17/08/23	6 hours	Static information all filled up	Recipe content and other dynamic information still need back-end connection
User profile page and User Create Recipe Page	20/08/23 - 05/09/23	5 hours	Page format and recipe creation interactions	Connecting the recipe to the backend database
Recipe card components	09/08/23 - 10/08/23	6 hours	Modular card components	Dynamic recipe title and description still needs to be added
Leaderboard page and Recipe page	13/09/23 - 17/09/23	3 hours	Leaderboards & recipe page layout	Dynamic recipe information needs to be connected to the database
Button and interactive form components	20/08/23 - 25/08/23	4 hours	Like button, comments box and search bar design	Component functionality

Table 1: progression log for front-end

4.1.2. Back-end and Middleware

After translating our high-fidelity design prototypes into functional and aesthetically pleasing components, we started coding on the back-end of our project, using SQLite3.

As the design and development phase progressed, we also embarked on rigorous testing procedures to ensure the robustness and reliability of our recipe-sharing website. User testing sessions were conducted, drawing from the insights gained during the initial design phase, and feedback was meticulously incorporated to enhance the user experience further.

Choosing an appropriate DBMS

A DBMS (Database Management System) was needed to organise, store, retrieve, and manage data coming in and out of Food Haven's website. Some characteristics we were looking out for when choosing a DBMS were:

- Supports and preserves relationships between records
- Industry-standard
- Support from a large community - have an easier time looking for help
- Scalability on demand
- 24/7 server uptime

SQLite3 matched the description and was thus our pick for the database.

Back-end time and progression log

Task	Start Date	Duration	Complete	Incomplete
Searching for dataset	31/08/23	2 hours	Kaggle dataset that is scraped from Epicurious with licence included found	Cleaned categorised dataset that is up to date could not be found
Middleware logic to parse CSV dataset into the SQLite3 database and make the data usable	20/08/23 - 25/08/23	3 hours	CSV data is parsed into the database and parked under "user" Epicurious to credit the source	Further cleaning and categorising of the data
Connecting the front-end template and components to the backend and coding their logic	31/08/23 - 15/09/23	2 weeks	The website's expected functionality works as intended.	Certain complex functionalities that are too complex will be reserved for future iterations

and functionality				of the project.

Table 2: progression log of backend

4.1.3. Breakdown of the team

Our project is driven by a devoted team of five individuals, each with their own set of abilities and expertise. We used a strategic approach by breaking the project up into smaller pieces to efficiently harness our collective capabilities and maximise time management.

We held frequent meetings which improved intra-team communication and let us concentrate on the specifics of our unique areas of expertise. These internal events helped to create a space where team members could direct their energy and expertise toward the parts of the project they were most passionate about.

4.1.4. Project Management Tools

In the ever-evolving landscape of project management, choosing the right set of tools is essential to ensure seamless collaboration and effective communication among team members. Our project team carefully curated a collection of tools tailored to our specific needs. These tools empowered us to stay connected, share ideas, gather feedback, and work together efficiently, ultimately contributing to the success of our project. Below, we introduce each of these tools, highlighting their unique features and the benefits they brought to our project.

Code Repository: Git and GitHub

Our collaborative coding efforts are supported by the seamless integration of GitHub and Git, a key tool in our agile software development toolbox. Git acts as the foundation for managing our codebase since it makes it possible to separate code into separate repositories and coordinate updates from many contributors. Our team can work simultaneously on a variety of files and folders thanks to this well-designed version control system, which also protects against accidental data loss or conflicts.

In our Git-based workflow, branches are essential workspaces for team members to work on tasks or features. After completion, code reviews are conducted, involving examination, suggestions for improvements, and feedback from team members. This collaborative review process ensures that the code meets standards and consistency before merging into the master branch of the repository. To maintain an organised version control history, branches are systematically removed once they have been approved and merged.

Git commits act as a thorough record of each change made throughout the project's lifecycle, enabling us to follow changes made during pushes and merges. This fine-grained tracking not

only helps in locating and addressing problems, but also makes it easier to collaborate easily, which is crucial to our coding process.

To monitor progress: Gantt Chart (Jira) & Kanban Board (Asana)

Our recipe-sharing website's successful development depends on effective project management. We used a combination of project management systems, particularly Jira and Asana, to improve our workflow, track progress, and guarantee effective task allocation.

Gantt Chart (Jira): We used Jira's ability to build Gantt charts, which were essential for illustrating project dependencies and timetables. We were able to guarantee that project goals were fulfilled on time thanks to these charts.

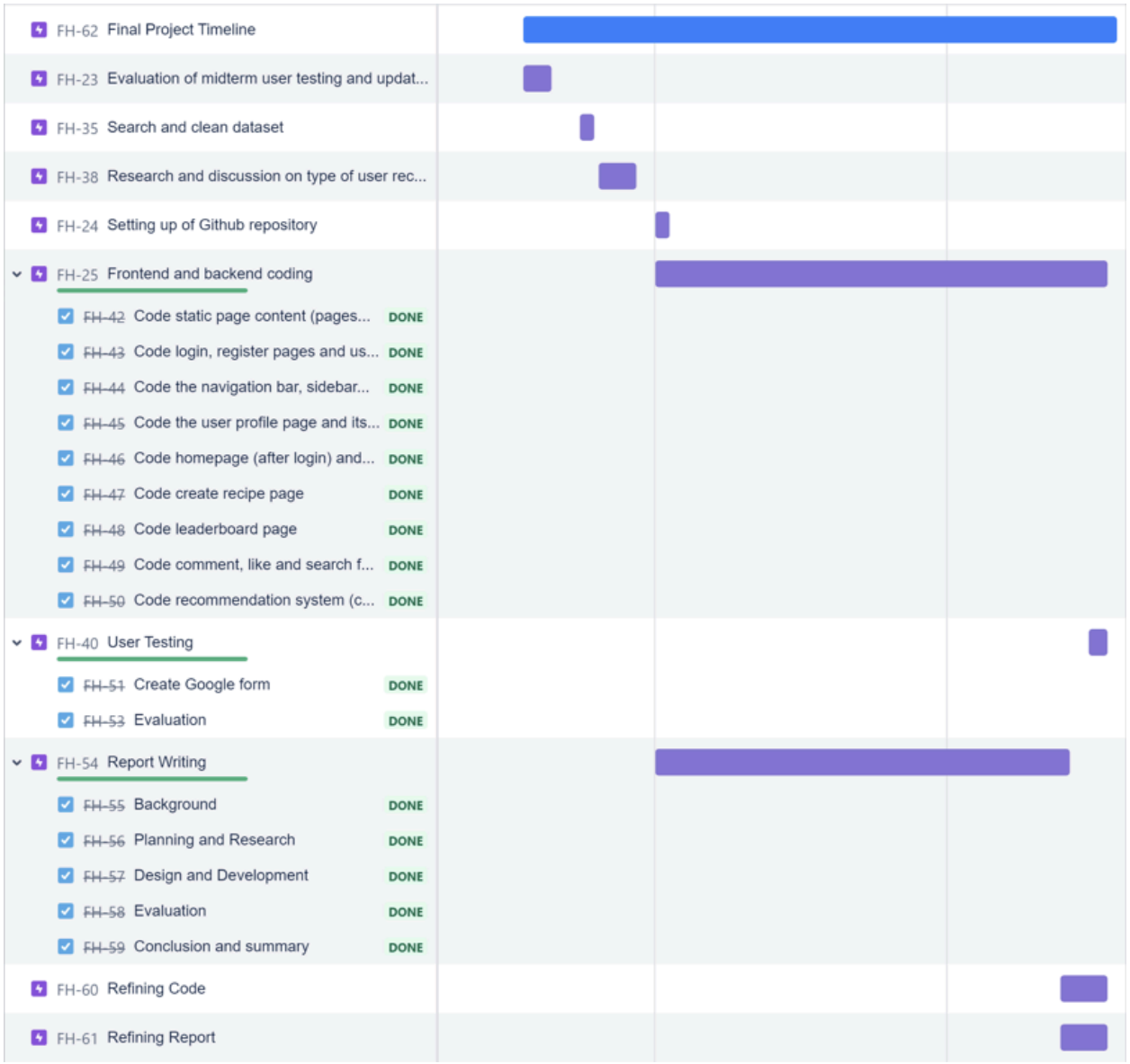

Figure 4.1.4.1 Illustration of Gantt chart (Jira)

Kanban Board (Asana): We made use of the Kanban boards in Asana as part of our agile project management strategy. These boards gave us a visual depiction of our workflow and made it easy for us to manage activities, set priorities, and monitor progress. We improved team communication, provided transparency, and sustained a constant rate of development by applying the Kanban methodology.

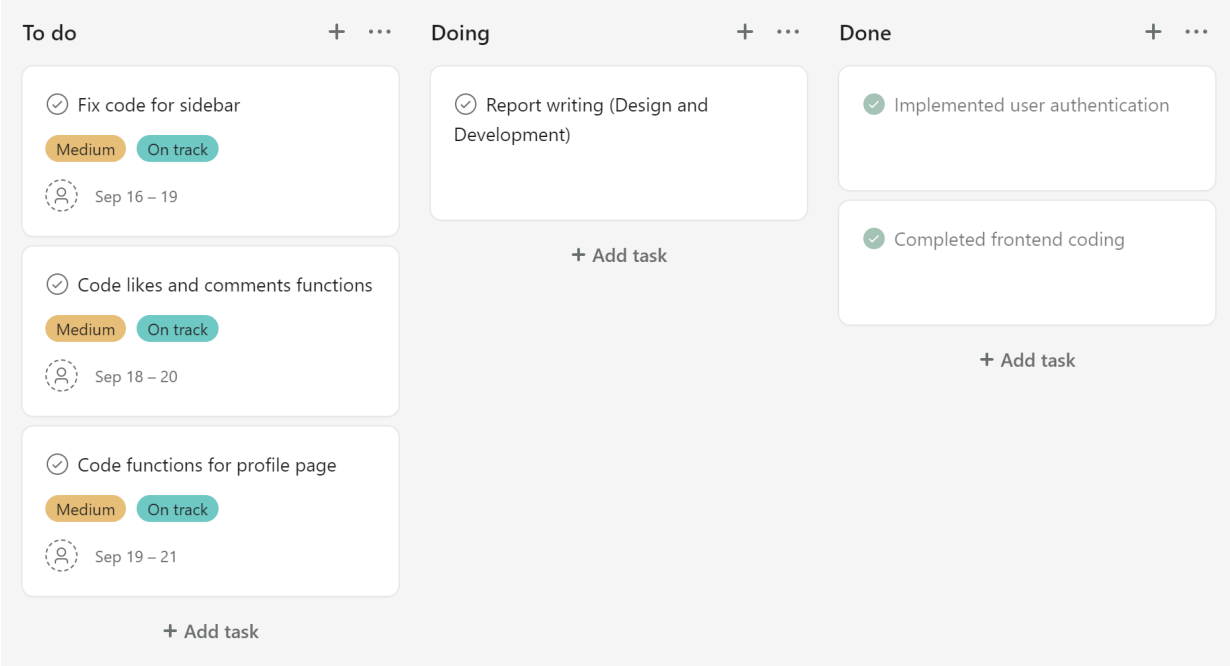

Figure 4.1.4.2 An example of Kanban Board (Asana) on August 18, 2023

These project management tools contributed considerably to the success of our project by promoting effective communication and easing task tracking.

Team Communication: Discord & Telegram

We employed Discord and Telegram as our primary communication channels. Discord provided us with versatile voice and text chat features, ideal for both casual discussions and formal meetings. Telegram, on the other hand, served as a quick messaging platform for sharing updates and important information. These choices offered flexibility, enabling team members to participate from anywhere, and ensure effective and timely communication.

Collaborative Documentation: Google Docs

Google Docs became our go-to platform for collaborative documentation. Its real-time editing capabilities allowed team members to work together on documents seamlessly. Features like

version history tracking and comment integration kept our work organised and efficient. With Google Docs, we maintained clear communication even when working asynchronously.

Feedback Collection: Google Forms

Gathering feedback from users was simplified with Google Forms. This user-friendly survey tool allowed us to design and distribute surveys and questionnaires effortlessly. Tracking responses in real time empowered us to make data-driven decisions throughout the project, enhancing our adaptability and responsiveness.

4.1.5. Dataset

Obtaining a reliable dataset of recipes was a crucial step in developing our program. However, this process presented several challenges. The major hurdle was the presentation and format of available datasets, many of which had complex structures, requiring significant time and effort to understand and integrate. This complexity posed a significant challenge due to our project's time constraints.

Ultimately, we discovered a well-maintained and pre-cleaned dataset sourced from Kaggle, containing recipes scraped from Epicurious.com and licensed under the (CC BY SA 3.0) creative commons by sharealike3.0. We were granted the freedom to use the data as long as proper attribution to the licence was provided, which we included for each recipe dataset.

```
<% if (recipe.firstName === "Epicurious") { %>
  <p>Recipe License:</p>
  <p>https://creativecommons.org/licenses/by-sa/3.0/</p>
<% } %>
```

Figure 4.1.5.1 code snippet of attributed licence

The dataset was neatly organised with columns for titles, ingredients, images, and instructions. Recipe images, stored as .png files, were accessed client-side by referencing their file names. To integrate it into our application's architecture, enabling real-time recipe recommendations, we implemented a collaborative filtering model.

4.2. User recommendation algorithm

User recommendation is a personalised recommendation system that suggests items to users based on their past interactions within a platform. The primary objective of utilising user recommendations in this context was to provide users with tailored content, without relying on simple keyword matching. Traditional keyword matching can be time-consuming and less effective in delivering personalised recommendations.

The research we conducted on user algorithms has led us to two main types of recommendation systems; collaborative filtering and content filtering, along with many different ways of implementing these systems.

1.1.1. Collaborative filtering

Collaborative filtering is a method that provides personalised recommendations by analysing the preferences of users similar to you and suggesting items based on their past preferences. It offers two main approaches: one identifies like-minded users and recommends what they have enjoyed, while the other looks at your previous preferences and suggests similar items. This approach can make accurate suggestions without needing a complete understanding of your entire profile; it leverages your past interactions and choices.

1.1.2. Content filtering

On the other hand, content filtering analyses specific attributes and characteristics of content you've interacted with in the past, recommending similar content based on your historical preferences. This filtering can take various forms, such as category-based or content-based filtering. One method identifies themes or subjects aligning with your interests, while another focuses on content features like keywords or genres. Content filtering excels at providing precise recommendations without needing an exhaustive user profile, relying on your prior interactions and choices.

4.3. Collaborative filtering vs Content filtering

The choice between collaborative and content filtering depends on our data and recommendation requirements. Collaborative filtering works well with abundant user data, making suggestions based on similar users' actions. Content-based filtering, however, is valuable when data is limited but demands accurate user historical information. Often, a combination of both approaches is used to achieve the best results.

In our project, we selected collaborative filtering as the primary recommendation method due to time constraints during user testing, making it challenging to gather extensive user history. Collaborative filtering offers the potential to enhance recommendations over time as our user base grows and provides more feedback. With increased user engagement and feedback, collaborative filtering can become even more effective in generating relevant recommendations.

4.4. Implementation methods

To understand the implementation of the user recommendation system, it's essential to grasp how our recipe feed functions. The feed operates by initially displaying five recipes when users visit their homepage. Scrolling to the bottom of the page triggers a script that makes a GET

request to fetch an additional five recipes thus an ‘endless’ feed, continually enhancing the user's experience.

Our recommendation system is based on collaborative filtering, Here's a breakdown of how it works:

```
function getRecipeRecommendations(userId, callback) {
  // Step 1: Get the recipe IDs liked by the target user
  db.all('SELECT recipe_id FROM user_likes WHERE user_id = ?', [userId], (err, likedRecipes) => {
    if (err) {
      console.error('Error fetching liked recipes:', err);
      return callback([]);
    }

    const likedRecipeIds = likedRecipes.map((row) => row.recipe_id);

    // Find recipes liked by other users who liked similar recipes
    db.all(`
      SELECT DISTINCT r.*
      FROM user_likes ul
      JOIN recipes r ON ul.recipe_id = r.id
      WHERE ul.user_id != ?
      AND ul.recipe_id NOT IN (${likedRecipeIds.join(',')})
      LIMIT 10; -- Limit the number of recommendations
    `, [userId], (err, recommendedRecipes) => {
      if (err) {
        console.error('Error fetching recommended recipes:', err);
        return callback([]);
      }

      callback(recommendedRecipes);
    });
  });
}
```

Figure 4.4.1 code snippet on collaborative filtering

We start by gathering the recipes the user likes then we identify other users who have liked similar recipes. This is achieved through an SQL query that looks for users who have liked recipes similar to the target user's likes. The recommendations obtained from this process are then integrated into the user's feed. For every 5 recipes displayed to the user, two of them are drawn from the collaborative filtering recommendation and the remaining three recipes are selected randomly.

Further improvements on the algorithm for future iterations

Our algorithm could be further improved upon further iterations as Content-based filtering, which takes into account a user's past preferences and the characteristics of recipes, is one way to improve the system. Combining collaborative filtering and content-based filtering in a 2:2:1 ratio of how the recipes are displayed to the user is one possible strategy. With two recipes based on collaborative filtering, two recipes based on content filtering, and one recipe chosen at random.

5. Design and Development

5.1. User accessibility

Alt Text for Accessibility: To ensure inclusivity, we've added proper alt text to all images. Alt text provides descriptions for images, aiding users with visual impairments while benefiting those with slow internet connections or image-disabled browsers. By including alt text, "Food Haven" becomes accessible to a diverse audience, enhancing their experience and promoting culinary exploration.

Inclusive Color Scheme: We've embraced an inclusive color scheme, using white and green for backgrounds and black and white for typography. This choice caters to users with varying visual abilities, creating an aesthetically pleasing and accessible interface. The balanced contrast and simple design foster a welcoming atmosphere for all recipe enthusiasts on our platform.

5.2. User testing

Initial user testing with our high-fidelity Figma mockup provided invaluable insights into the user experience, informing our prototype's strengths and areas for enhancement. Observing real users' interactions deepened our understanding of their expectations and pain points. This data-driven feedback informs decisions for the next prototype iteration.

Production testing occurred after implementing core website functions. Testers explored and interacted with the site, then completed a System Usability Scale (SUS) survey and Google Forms feedback. It's crucial to note that certain interactions (e.g., user following, user pages) and minor functions (e.g., password retention, user delete) were in development but included in testing for comprehensive feedback. This intentional approach helped uncover early insights and potential issues, contributing to a smoother development process and an improved final product.

5.3. Design - first iteration

In the initial SUS survey and user feedback for our first project iteration, we received an average SUS score of 82, which is notably higher than the average SUS score of 68. This feedback provided us with valuable insights into the user interface of our prototype.

Specifically, users recommended several UI enhancements, including resizing icons to optimise space utilisation, incorporating more images to visually guide users through recipe steps, and improving the contrast between button colours and text for better readability.

Moreover, users suggested functional improvements, such as clarifying the purpose and functionality of certain features like leaderboards and user interactions. Additionally, there were recommendations related to the Document Object Model (DOM), advising us to refine the placement of elements, consider a lighter colour scheme for a more visually appealing design, and enhance text layout to prevent a cluttered appearance.

Overall, this feedback has been instrumental in shaping the project's direction and ensuring a more user-friendly and visually appealing interface in the second iteration of the project.
SUS Scores and user feedback from the user testing done for the first iteration in the midterms:

SUS Scores	11. Did you encounter any difficulties or confusion while interacting with the prototype? If yes, please specify.	12. Are you satisfied with the colour scheme? If no, could you please recommend one below	13. How can we improve the website?(Features, rating system, UI design improvements, Placement of elements, etc.)
80	-	Button colours and text colour should have more contrast + sync all buttons colour to be the same except delete button	icons can be resized as it seems to be taking up quite a bit of space
72.5	nope	yes!	Can add a tutorial video too if you have!
67.5	NA	I appreciate the minimal colour scheme however there could be a few eye-catching elements or colours implemented.	The text for the recipe seems quite chunky, there could be more images to show the breakdown of each step or improve the layout of the text to avoid looking plain & chunky.
90			
62.5	Navigation between the profile and the liked videos in the nav bar was confusing	Lighter colour scheme with usage of block colours would be better	Na
77.5	No	It is a bit tacky, I think a more vibrant colour would be better.	Nil

77.5	Nope	Yes	Very good
87.5		yes looks good	ingredients option in recipe probably don't need a dropdown field since there are many other ingredients needed?
97.5	NIL, in fact the prototype was well done, interface was user-friendly as well👍🏻	yes	NIL
67.5	Some of the features were not made clickable, not sure if it was because there was not enough time to do more wireframes for the respective features. Not sure what the leaderboards does or what is the purpose of it. Like does it mean that I have the most comment and most likes within my community? Or within the country? etc. The top recipes on the leaderboards will be broadcasted to more people, so that more people will cook the dish? Maybe you could change the name instead of leaderboards, maybe it could be something like " Most Popular/Favourite recipes in your community/country" etc. So that users will better understand the purpose of it.	Colour scheme is alright, it goes well with the entire UI of the website. But perhaps you can make the footer and navigation bar the same colour instead. Whereas for the blue colour, you can consider putting it as a button feature or when you hover your mouse over something and the placeholder will turn blue etc. Blue colour is like the "highlight" so recommended to put it as minimal as possible and on something that has more purpose or is more frequently used (i.e. buttons).	As stated above :)
100	No	Yes	Allow the user to type in their ingredients as an open ended question rather than pick from a

			drop down list under upload a recipe as it does not seem feasible to list down every ingredient that could possibly be used. I also feel that putting profile settings in the user's profile page would make more sense, as it would be where the user clicks to make changes to his/her profile. Where the settings button is currently placed would be more suited for interface settings imo, to allow users to make their viewing experience of the website more pleasant (Maybe allow users to adjust font size, change colour scheme of page etc to suit their own preferences).
85	the category 'user interactions' was confusing	yes nice colours	i think all's good
82.5			
97.5	no	I like the colour scheme, it is quite aesthetic.	rating system of leadership rank based on number of likes might not be very fair. Accounts that have larger following would take up most of the top positions. How about ranking them based on scores out of 100. Users who have tried the recipe can rate out of 100 and average scores are taken. Could possibly include two search categories, one for users (they want to search up a specific

			account) and one for particular recipes of dishes. clarity on user interactions. Maybe there can be a community section where people join groups to meet others who have similar culinary interests as them. or have a comment section under the recipes.
85			The Login button seems oddly placed. Maybe you can swap positions with the Sign Up button, so the only blue button is at one side. Also, the User Interaction button brings the user to the Home page. I think if that is the main purpose, the button is not needed since tapping on the logo at the top left side serves the same purpose.
AVERAGE SUS SCORE: 82			

Table 3: sus scores for first iteration

5.4. Overall - second iteration

In the second iteration prototype's SUS survey and user feedback, we received an average SUS score of 78.12, which is notably higher than the average SUS score of 68, but lower than the first iteration's SUS score average of 82.

sus scores	11. Did you encounter any difficulties or confusion while interacting with the prototype? If yes, please specify.	12. Are you satisfied with the UI? If not, can you please specify how we can improve the visual appeal of our website?	13. How can we improve the website?(Features, rating system, UI design improvements, Placement of elements, etc.)
100	nil	yes very appealing and consistent	nil

100	nope	looks good	all's good
75	the leaderboard did not specify which recipe was the most popular so i had to assume it was in descending order, how much was one recipe more popular than the other was also not specified	recipe page could have been improved as the tabular data looks static, tabs on the profile could be larger	filters to recipes could be added to allow filtering of multiple recipes
77.5	NO	The main page is rather bland	Make the main page have more element that stands out
70	remember me button when logging in was not working	visually it is simple, clear and easy to understand. everything else works nicely	maybe include some part of the leaderboard in the home page ? like a side bar or something cos i didn't know there was a leaderboard until i clicked the sidebar after seeing it in google forms
90			
90			
95	I see the followers count on the profile, yet there is no way to follow another account?	There is a weird gap below the footer on the login, register, My Recipes page	Home page looks boring. Maybe leave the side navigation bar permanently out to make it more obvious to visitors the numerous ways they have to browse the website.
82.5	No	Yes, very easy to use.	More elements to explore. Display fit for phone.
90			
67.5	blurry pics		hq pics
72.5	internal server error when deleting account		make deleting account work
AVERAGE SUS SCORE: 78.12			

Table 4: sus scores for second iteration

The average SUS score for the second iteration of the project is calculated at 78.12, this indicates that the majority of users perceive the system as relatively usable and user-friendly. Several respondents expressed satisfaction with the user interface (UI) design, highlighting its simplicity, clarity, and ease of use.

However, it's worth noting that some users encountered specific issues, such as difficulties with the "Remember Me" button during login and blurry images. Additionally, there were suggestions for improvement, including enhancing the visual appeal of the main page as most users feel that the main page could have been populated with more features, and some even suggested including the leaderboard on the homepage. Some also requested clearer instructions for features like the leaderboard and refining the profile page's design to include features like a follower system.

Overall, the results indicate a generally positive user experience, with room for enhancement in areas identified by respondents. These insights will guide our efforts to refine and optimise our web application, ensuring a more enjoyable and user-friendly experience for all.

It's important to acknowledge that some features in your project were still in progress and not fully functional at the time of user testing. This oversight might have influenced the user feedback and scores, as users may have encountered issues or limitations associated with these unfinished features.

In future iterations of user testing, it will be important to communicate clearly which features are ready for testing and which are still in development. This transparency can help manage user expectations and provide a more accurate assessment of the features that have been fully implemented.

5.5. SUS (System Usability Scale) survey

In our project, we utilized two primary validation methods to comprehensively assess user experience: gathering user feedback and employing the System Usability Scale (SUS) in both project iterations. In the initial high-fidelity Figma prototype, we relied on SUS testing and a user feedback form to evaluate website usability. This choice was made over heuristic evaluations due to specific considerations.

Firstly, our user base was relatively small, limiting extensive prototype testing. Heuristic evaluations would have narrowed testing to usability designers, reducing available testing resources.

Secondly, we aimed for consistency, aligning with our mid-term assessment's use of the SUS survey. This approach enabled us to gauge project progress through SUS scores.

User feedback questions allowed detailed insights into project strengths and weaknesses, providing valuable qualitative data for enhancing user experience.

Comparison of SUS Test Results:

Comparing SUS test results between the two project iterations reveals noteworthy trends.

In the first round, the average SUS score was 82, indicating positive usability perceptions. Users praised the appealing color scheme and website simplicity. However, some confusion arose regarding certain features and the leaderboard's purpose. Improvement suggestions included resizing icons and optimizing recipe text layout.

In the second round, the average SUS score slightly decreased to 78.12, still reflecting positive usability feedback. Users commended the consistent and appealing UI. Ongoing concerns included leaderboard functionality clarity and missing features like account following. Users also noted issues like blurry images and internal server errors during account deletion.

Overall Evaluation:

The comparison between the two rounds of SUS tests indicates that your recipe website maintains a strong foundation of usability and visual appeal. Users consistently appreciated the colour scheme, clear layout, and easy navigation. However, there are areas for improvement, including enhancing feature clarity, addressing technical issues, and optimising the visual elements.

To further enhance the user experience, for future iterations, we will consider refining the explanation of features like the leaderboard and implementing user-requested functionalities like account following. Additionally, resolving technical glitches and ensuring high-quality images will contribute to a smoother user journey. Overall, the SUS test results serve as valuable feedback to guide your ongoing efforts to create an even more user-friendly and visually engaging recipe platform.

5.6. Initial Design (Figma high-fidelity prototype)

In our initial design, several participants recommended clarifying the purpose and functionality of certain features, such as the leaderboards and user interactions. There were also suggestions to refine the placement of elements, consider a lighter colour scheme, and improve the text layout to avoid a cluttered appearance. Thus, the team took this advice to heart and made further design changes in the second iteration based on the user feedback.

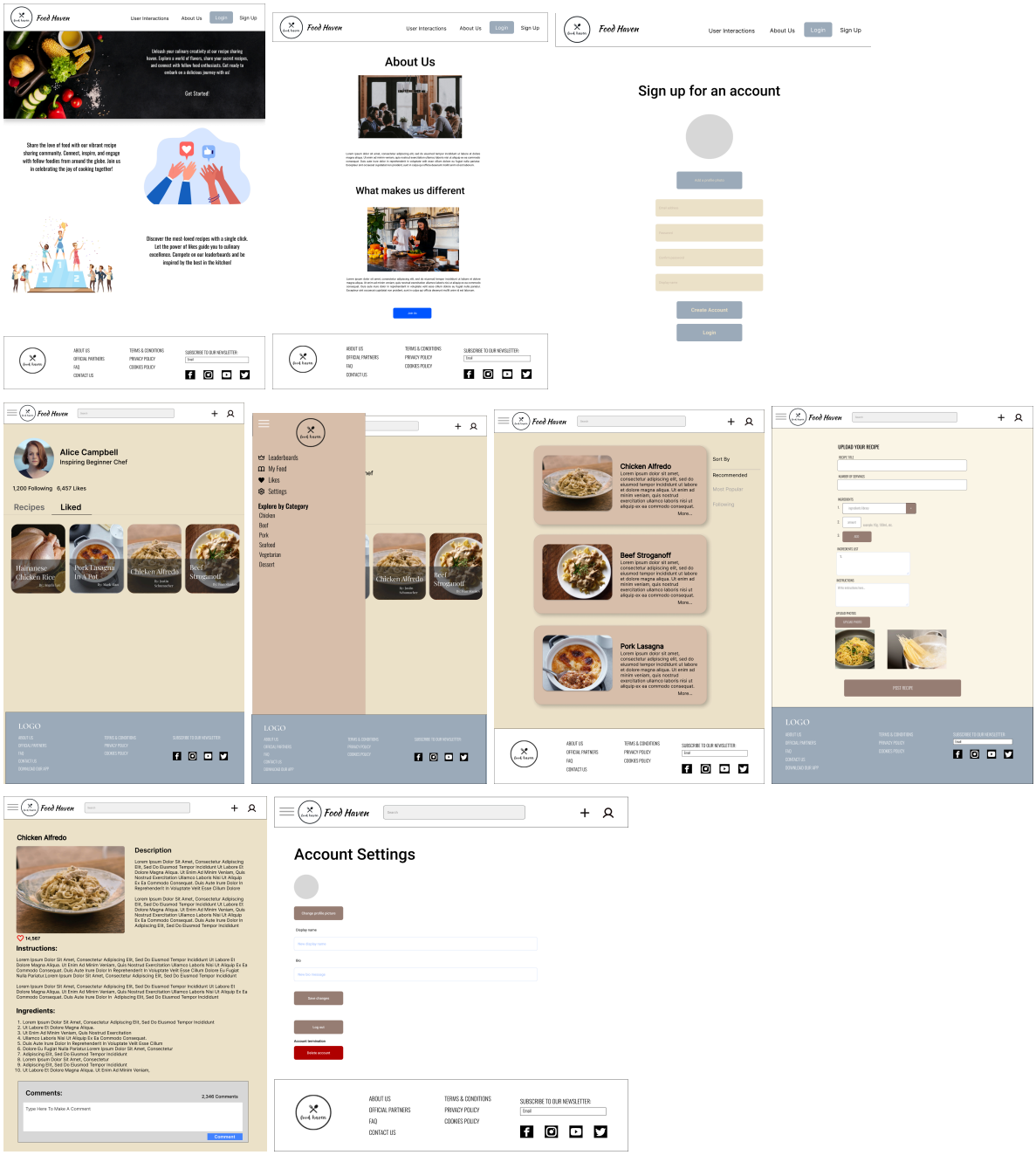

Figure 5.6.1 high-fidelity mock-ups

1.1. Production prototype - First iteration

In response to valuable user feedback, we've made significant enhancements to our project for a more intuitive and visually appealing user experience. We've resized icons, optimised space, and incorporated more visuals to guide users through recipes, enhancing usability.

We've also improved feature clarity and functionality, ensuring users understand the purpose of features like leaderboards. Technical enhancements, including DOM improvements, a lighter colour scheme, and streamlined text layout, reduce clutter and enhance the overall design.

1.2. Secondary design

Initially, we planned to send a ZIP file for users to install our project, but we prioritised user-friendliness. We hosted it on DigitalOcean, making it easily accessible via a web browser, reducing barriers to user testing and ensuring a consistent experience. This approach provides more accurate feedback and insights, with the manual installation option as a backup if needed.

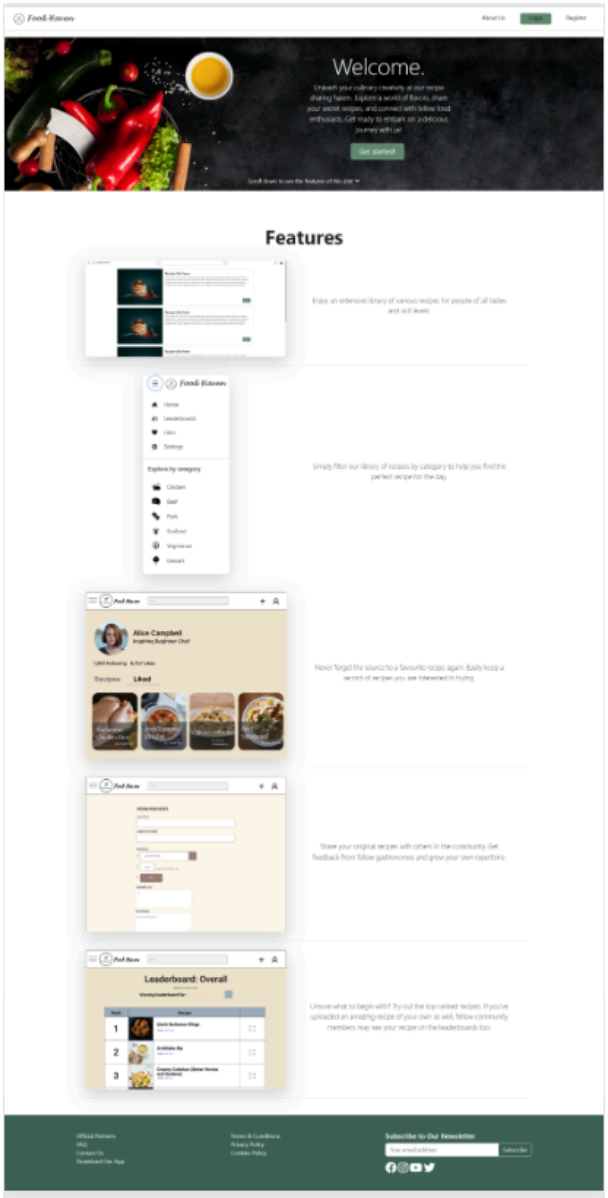

Figure 5.8.1 Landing page

In response to user feedback, we have added a guide on how the website works, along with an explanation of its features, directly to the landing page. This enhancement aims to provide users with better support and assistance in effectively using the web application.

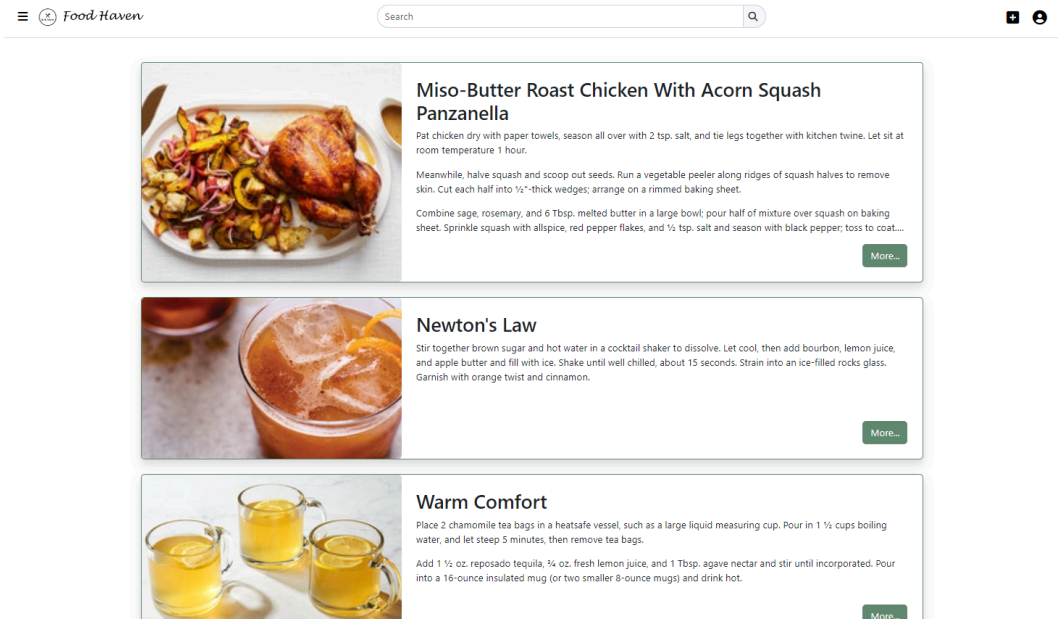

Figure 5.8.2 Food Haven Home Page

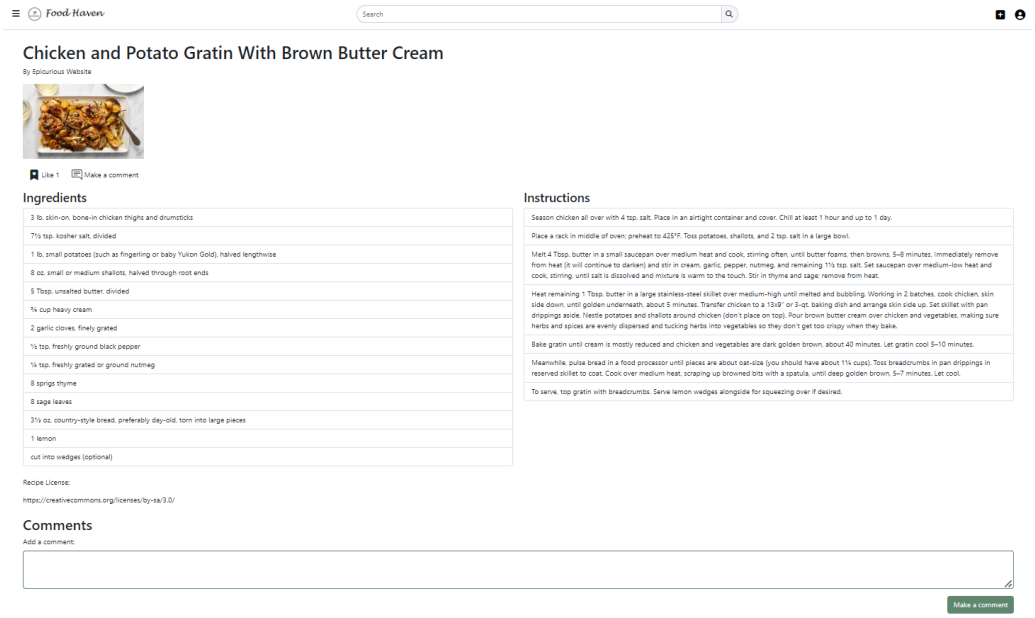

Figure 5.8.3 Food Haven Recipe Page

We have also meticulously resized icons and leveraged space more efficiently, allowing users to navigate our platform effortlessly. Instead of sticking to a static page layout with limitations on the number of displayed recipes, we've introduced a dynamic feature that will populate the home page with more recipes as users scroll to the bottom of the page, this enables users to

scroll through an infinite feed of recipes which enhances the overall user experience and ensures easy access to a wide variety of content.

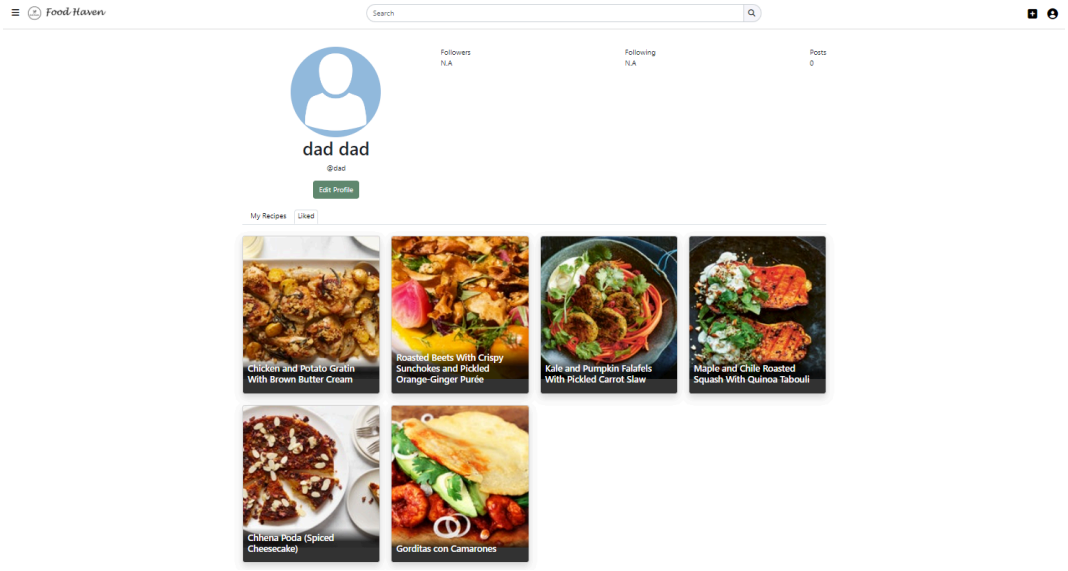

Figure 5.8.4 Food Haven Profile page

For the website's profile page, we have maintained a design similar to TikTok's profile page to create a sense of familiarity and facilitate easy navigation for users. We made slight modifications to the colour theme to meet the user feedback given. We have applied these changes consistently to all other pages. In these updates, we transitioned from the original beige colour to a cleaner and lighter white scheme, complemented by green accents and outlines.

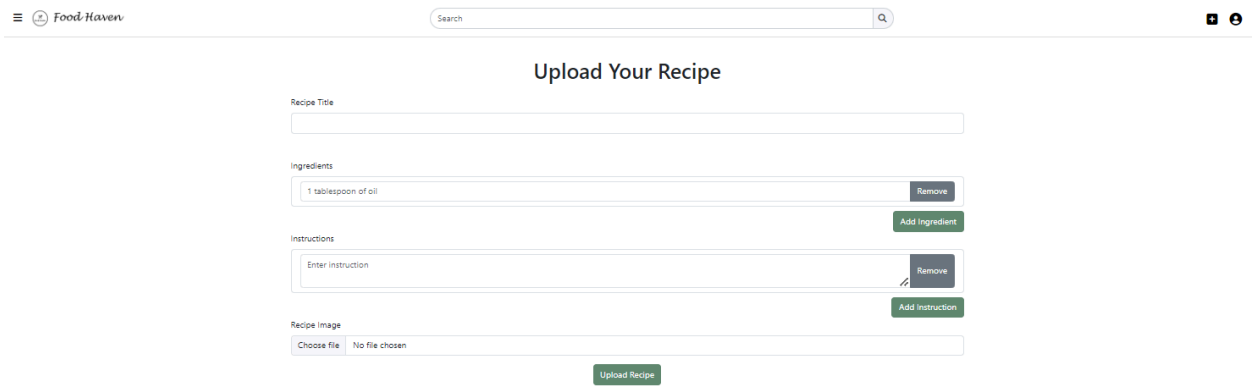

Figure 5.8.5 Food Haven Create A Recipe Page

5.7. Presentation layer

The presentation layer of our web application is designed to work seamlessly on modern browsers like Google Chrome, Microsoft Edge, and Safari, with JavaScript support enabled.

We've decided to make use of EJS as our templating language for both front-end and back-end development to improve user interactions and overall functionality.

We have several benefits over traditional HTML, CSS, and JavaScript methods thanks to our choice of EJS. These benefits include modularity, improved performance, scalability to support future growth, adaptability to changing user needs, familiarity for developers, platform compatibility, effective component reuse, dynamic content rendering, and increased expressiveness. These characteristics work together to speed up development processes, producing an application that not only reduces development time but also improves in terms of quality and usefulness.

5.8. Structure and Implementation

This website takes on a view-control approach:

- **'index.js'** is responsible for manipulating the Document Object Model (DOM) and facilitating the execution and rendering of EJS pages located in the views folder.
- '**leaderboard.ejs, leaderboard-content.ejs, recipe-page.ejs, etc.** corresponds to each page of the website to reduce overwriting each other's codes.
- **'public/scripts/...' and 'public/routes'** handles various user inputs and interactions such as navigation side panel.
- **'routes'** folder oversees page navigation, defining endpoints that respond to client requests for specific actions such as interaction with the 'like' button, which transmits the number of likes given for a particular recipe.
- Use of view control is reinforced with SPA (Single-Page-Application) ejs files that require no full page reloads with js interactions.
- SQLite3 as our database management system that allows us to import the 'epicurious' CSV and insert it as a database in '**database.db'**. The system allows users to add new data and modify existing data easily.
- **'assets'** folder holds images such as 'profileImages' that display images of each recipe.
- **'package.json'** has important information such as name, licences and installed packages

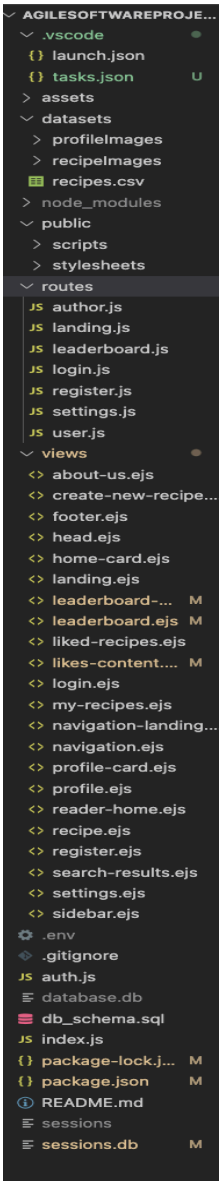

Figure 5.11.1

5.9. Search page and functionality

A user is allowed to search recipes by title using a search bar on the user interface of the website. We utilised body-parser (*Body-parser*. npm., n.d.) to parse user input in the search bar, and an SQLite query that returns the list of recipes with titles about the keywords given by the user. This is done via the code below:

```
// Function to add loaded recipe IDs to the global array for search results
function addLoadedSearchRecipeIds(recipes) {
  recipes.forEach((recipe) => {
    loadedSearchRecipeIds[recipe.id] = true;
  });
}


// Search route
app.get('/search', isAuthenticated, (req, res) => {
  const searchQuery = req.query.query;

  // Clear the array for search results when performing a new search
  loadedSearchRecipeIds = {};

  // Store the search query in a session or as a cookie
  req.session.searchQuery = searchQuery;

  // Sample SQL query (you should adapt this to your database structure):
  db.all('SELECT * FROM recipes WHERE Title LIKE ? LIMIT 5', [`%${searchQuery}%`], (err, searchResults) => {
    if (err) {
      console.error('Error searching for recipes:', err);
      res.render('error');
    } else {
      // Add the initially loaded recipe IDs for search results to the global array
      addLoadedSearchRecipeIds(searchResults);

      // Render a search results page with the matching recipes
      res.render('search-results', { searchResults });
    }
  });
});
```

Figure 5.12.1 code snippet of search query

The search bar functionality in this code enables users to search for recipes by entering a query. Upon initiating a search, any previous search results are cleared to provide fresh outcomes as an array is initiated each time the search page is called. The query is extracted from the user's input via a variable “searchQuery” which is used to store and manage the search query entered by the user. and used to filter recipes whose titles match the query. Initially, a limited set of results is fetched from the database and displayed to the user.

```
// Route to load more search results
app.get('/load-more-queries', isAuthenticated, (req, res) => {
  const page = parseInt(req.query.page) || 1;
  const pageSize = 5; // Number of queries to load per request
  const offset = (page - 1) * pageSize;

  // Retrieve the search query from the session or cookie
  const searchQuery = req.session.searchQuery || '';

  // Fetch more search results from the database
  db.all(`SELECT id, Title, Instructions, Image_Name FROM recipes WHERE Title LIKE ? LIMIT ${pageSize} OFFSET ${offset};`, [`%${searchQuery}%`], (err, recipes) => {
    if (err || !recipes) {
      console.error('Error fetching more recipes queries:', err);
      res.status(500).json({ error: 'Error fetching more recipes queries' });
    } else {
      // Filter out any duplicate recipe IDs for search results
      const uniqueSearchRecipes = recipes.filter((recipe) => !loadedSearchRecipeIds[recipe.id]);

      // Calculate how many more unique recipes are needed to reach 5
      const neededUniqueSearchRecipesCount = 5 - uniqueSearchRecipes.length;

      if (neededUniqueSearchRecipesCount > 0) {
        // Fetch additional unique recipes for search results to make up the count
        db.all(`SELECT id, Title, Instructions, Image_Name FROM recipes WHERE Title LIKE ? AND id NOT IN (${Object.keys(loadedSearchRecipeIds).join(',')}) LIMIT ${neededUniqueSearchRecipesCount};`, [`%${searchQuery}%`], (err, additionalSearchRecipes) => {
          if (err || !additionalSearchRecipes) {
            console.error('Error fetching additional recipes queries:', err);
            res.status(500).json({ error: 'Error fetching additional recipes queries' });
          } else {
            // Add the loaded unique recipe IDs for search results to the global array
            addLoadedSearchRecipeIds(additionalSearchRecipes);

            // Concatenate the additional search recipes with the unique search recipes
            const allUniqueSearchRecipes = uniqueSearchRecipes.concat(additionalSearchRecipes);

            // Send the unique search recipes as the response
            res.json({ recipes: allUniqueSearchRecipes });
          }
        });
      } else {
        // Add the loaded unique recipe IDs for search results to the global array
        addLoadedSearchRecipeIds(uniqueSearchRecipes);

        // Send the unique search recipes as the response
        res.json({ recipes: uniqueSearchRecipes });
        console.log( loadedSearchRecipeIds)
      }
    }
  });
});

//Render settings page
```

Figure 5.12.2 code snippet to load unique recipes and additional recipes

As the user continues to scroll down the page and requests additional results, the system utilises an object known as "loadedSearchRecipeIds" to keep track of the IDs of all the recipes that have already been displayed. When new recipes are to be loaded, they are checked against the IDs stored in this object. This process effectively filters out any duplicates, guaranteeing that each set of 5 recipes displayed as the user scrolls remain unique. Pagination is achieved by dynamically adjusting the SQL query's offset parameter based on the current page number. To detect when the user reaches the bottom of the page and trigger the loading of more recipes, a script is employed. This system ensures a smooth and efficient user experience, where fresh recipes seamlessly appear as the user scrolls, all while avoiding redundancy through the code's ability to handle the loading and display of distinct recipes.

5.10. Passport.js

Authentication using Passport, an authentication middleware for Node.js applications. We used Passport.js because it is flexible and modular thus it can be easily integrated into any Express-based web application such as ours.

Passport.js is integrated into the Node.js application as middleware. It provides various authentication strategies, such as local authentication (username and password), OAuth, and more. These strategies are implemented as plugins.

For our project, we stuck with local authentication as our only authentication strategy which allows our users to log in with their username and password as seen in the LocalStrategy below:

```
// Passport.js authentication strategy
passport.use(
  new LocalStrategy(function (username, password, done) {
    global.db.get('SELECT * FROM users WHERE username = ?', [username], function (err, user) {
      if (err) {
        return done(err);
      }
      if (!user) {
        return done(null, false, { message: 'Invalid username' });
      }
      bcrypt.compare(password, user.password, function (err, result) {
        if (err || !result) {
          return done(null, false, { message: 'Invalid password.' });
        }
        return done(null, user);
      });
    });
  })
);
```

Figure 5.13.1 code snippet for authentication

Our local strategy follows this process; when a user attempts to log in, the strategy would query our SQLite database to find a matching username. If the username is not found, it returns an error, signalling that the username is invalid. If the username exists, it proceeds to compare the provided password with the stored, hashed password using bcrypt. If the comparison fails or an error occurs, it returns a failure message; otherwise, it authenticates the user, allowing them access to protected resources within the application such as the routes only available to authenticated users.

Passport.js is then initialised in our application by calling it in our middleware which enables it to handle authentication requests:

```
//use functions to set up basics and sessions
app.use(passport.initialize());
app.use(passport.session());
```

Figure 5.13.2 code snippet passport.js initialisation

When a user tries to log in or access a protected area, Passport.js handles the request. For example in our project, User registration is achieved through a registration form, with the user's credentials being stored in the SQLite3 database. When a user submits their login details via a login form, Passport.js carries out the authentication process based on the chosen strategy or for our case would be the local strategy as it is the only available strategy.

After attempting authentication, Passport.js either successfully authenticates the user or returns an authentication failure.

```
// redirect based on results of authentication
router.post('/', passport.authenticate('local', {
  successRedirect: '/home', // Replace with the path to the dashboard page after successful login
  failureRedirect: '/login',
  failureFlash:true
}));
```

Figure 5.13.3 code snippet of result of authentication

For our project a successful login would then redirect the user to the homepage which contains their user feed and a failure would redirect them back to the login page with a failure message which tells them the reason for their authentication failure.

After setting up passport.js we are then provided with a variety of functions such as user serialisation which provides methods for serialising and deserialising user objects, these allow passport.js to maintain a user session and store the user's identity in a session while the deserialisation allows us to retrieve the user's information from a session when needed.

```
// Passport serialize and deserialize user functions
passport.serializeUser((user, done) => {
  done(null, user.id);
});

passport.deserializeUser((id, done) => {
    global.db.get('SELECT * FROM users WHERE id = ?', [id], function (err, user) {
    done(err, user);
  });
});
```

Figure 5.13.4 code snippet of serialisation and deserialisation

This functionality enables us to implement various user-related interactions, such as the "like" button, tracking the recipes liked by the user, or displaying recipes created by the user. Combining this feature with express-session has also allowed us to implement a functional search bar that assists users in finding their desired recipes(shown in the figure below), thereby enhancing the overall user experience.

```
// Search route
app.get('/search', isAuthenticated, (req, res) => {
  const searchQuery = req.query.query;

  // Clear the array for search results when performing a new search
  loadedSearchRecipeIds = {};

  // Store the search query in a session or as a cookie
  req.session.searchQuery = searchQuery;

  // Sample SQL query (you should adapt this to your database structure):
  db.all('SELECT * FROM recipes WHERE Title LIKE ? LIMIT 5', [`%${searchQuery}%`], (err, searchResults) => {
    if (err) {
      console.error('Error searching for recipes:', err);
      res.render('error');
    } else {
      // Add the initially loaded recipe IDs for search results to the global array
      addLoadedSearchRecipeIds(searchResults);

      // Render a search results page with the matching recipes
      res.render('search-results', { searchResults });
    }
  });
```

Figure 5.13.5 code snippet of like and search features with express-session

5.11. Sign up and sign in

We have established that login authentication plays a huge role in Food Haven's website. The ability to identify users with individual login accounts enables users to identify one another in the community when leaving comments on recipe pages and viewing the author of a recipe item. Retaining data on an individual's preferred recipes is crucial to the functionality of the recommendation system. Passport.js facilitates the verification of a client's authentication status and provides distinct web pages and restrictions for both authenticated and unauthenticated users.

```
// Render home page
app.get('/home', isAuthenticated, (req, res) => {
  const limit = 5; // Number of recipes to load initially
  db.all(`SELECT id, Title, Instructions, Image_Name FROM recipes LIMIT ?;`, [limit], (err, recipes) => {
    if (err || !recipes) {
      console.error('Error fetching more recipes:', err);
      res.status(500).json({ error: 'Error fetching more recipes' });
    } else {
      // Filter out any duplicate recipe IDs
      const uniqueRecipes = recipes.filter((recipe) => !loadedRecipeIds[recipe.id]);
```

```
//default page when opening the application
app.get('/', notAuthenticated, (req, res) => {
  res.render("landing");
});
```

Figure 5.14.1 (Top): 'isAuthenticated' to see recipes Figure 5.14.2 (Bottom): 'notAuthenticated' at start page

```
// Authentication middleware to check if the user is authenticated
function isAuthenticated(req, res, next) {
  if (req.isAuthenticated()) {
    return next();
  }
  res.redirect('login');
}

//function to prevent logged in users to go back to the login page through url
function notAuthenticated(req, res, next) {// this should go on things like the login,register and landing routes
  if (req.isAuthenticated()) {
    return res.redirect('/home');//logged in homepage redirect
  }
  next()
}
```

Figure 5.14.3 code snippet of 'isAuthenticated' and 'notAuthenticated' functions used to maintain user access

Thus users who are in a 'logged in' state would not be able to access the landing page and would be redirected back to the home page and users who have not logged in will not be able to access the home page through the URL link.

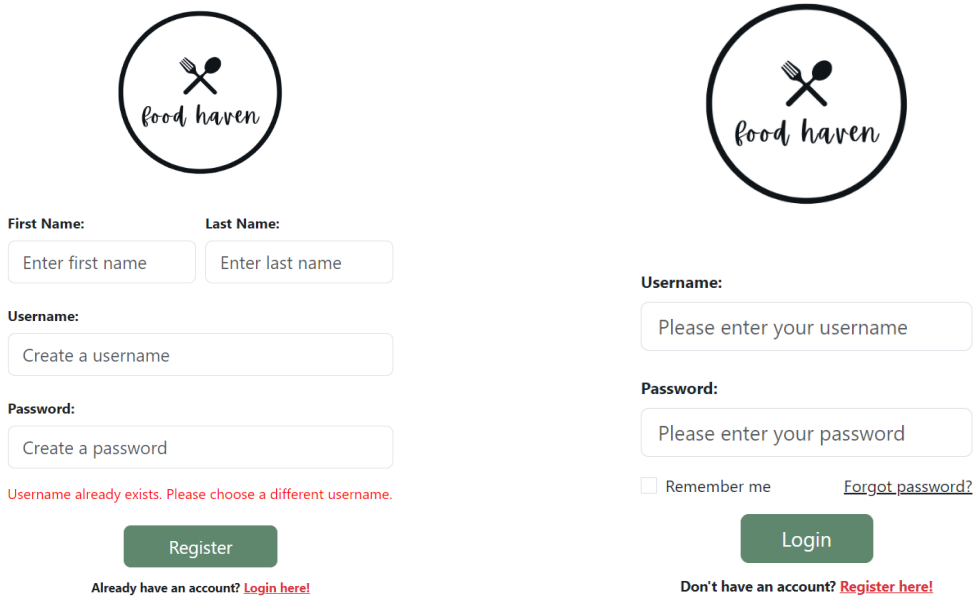

Figure 5.14.4 register page(Left), Log-in page(Right)

We have ensured that newly-created account usernames are unique in the database to avoid logins to the wrong account and information through our registration process. The registration is done by extracting data from the form upon submission from the register button. The database then checks and verifies the username availability, if the chosen username already exists. If an error occurs during this database query, it logs the error, sets an error message in the user's session, and redirects them to the registration page as shown below:

```
// Check if the username is taken
global.db.get(
  "SELECT * FROM users WHERE username = ?",
  [username],
  function (err, user) {
    if (err) {
      console.error(err);
      req.session.registrationError = "An internal error occurred. Please try again later.";
      return res.redirect("/register");
    }

    if (user) {
      req.session.registrationError = "Username already exists. Please choose a different username.";
      return res.redirect("/register");
```

Figure 5.14.5 code snippet of error messages for existing users

Then the code proceeds to hash the provided password using bcrypt with a salt factor of 10 for security.

```
// If not taken, hash the password before storing in the database
bcrypt.hash(password, 10, function (err, hashedPassword) {
  if (err) {
    console.error(err);
    req.session.registrationError = "An internal error occurred. Please try again later.";
    return res.redirect("/register");
  }
```

Figure 5.14.6 code snippet of password hashing

After successfully hashing the password, it inserts the new user's information (firstname, lastname, username, and hashedPassword) into the database, registering a user.

5.12. Modules

Web application framework

A server-side JavaScript runtime environment is needed to facilitate the handling of I/O between servers and clients. Popular options included:

- React.js
    - Fast and efficient rendering with virtual DOM
    - Easy to learn and use, with a declarative syntax
    - Component-based architecture makes code reusable and maintainable
    - Large and active community, with a wealth of resources available
    - Can be difficult to debug due to its component-based architecture
    - May require additional libraries to implement some common features
    - Not as well-suited for large and complex applications as some other frameworks

- Angular
    - Comprehensive and feature-rich framework
    - Provides a complete solution for building web applications, from data binding to routing to testing
    - Suitable for building a wide range of applications, from small websites to large enterprise systems
    - Large and active community, with a wealth of resources available
    - A steep learning curve, especially for beginners
    - Can be complex and overwhelming to use for small or simple projects
    - May not be the best choice for performance-critical applications
- Node.js
    - High performance
    - High scalability
    - Easy to learn
    - Large and active community, with a wealth of resources available
    - Cost-effective
    - Not as well-suited for CPU-intensive applications
    - Callback hell
    - Lack of maturity, API is not as stable as other platforms

Overall, Node.js's versatility led us to use it in our middleware.

5.13. CSS Framework

Bootstrap is a framework for creating responsive web pages efficiently, using HTML, CSS, and JavaScript. It played a pivotal role in our project, offering various advantages. Its user-friendly nature accommodated team members with different levels of front-end programming skills. Bootstrap's simple syntax and pre-built elements allowed for the quick creation of intricate layouts. It ensured a cohesive visual identity across the site, crucial for our collaborative work. In essence, Bootstrap saved us time while delivering a polished and professional appearance to our website.

```
<div class="page-content page-no-sidebar">
  <div class="row flex-lg justify-content-end h-75 gx-0" id="landing-hero-text">
    <div class="col-md-6 py-5">
      <div class="col-md-6 text-center">
        <h1 class="display-3 faw-bold lh-1 mb-3">Welcome!</h1>
        <p class="lead">Unleash your culinary creativity at our recipe sharing haven. Explore a
        world of flavors, share your secret recipes, and connect with fellow food enthusiasts.
        Get ready to embark on a delicious journey with us!</p>
```

Figure 5.17.1 Code snippet of Bootstrap classes used to style the elements

5.14. Templating language

Using a templating language was in line with modularisation. It allowed us to separate the work among the team members and piece it together a lot easier, such as the example shown below, where the navigation bar, the sidebar and the cards to display the recipes are spread out into separate components and pieced together in another page template.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <%- include('head.ejs') %>
  </head>

  <body>
    <%- include('navigation.ejs') %> <%- include('sidebar.ejs') %>
```

Enabling dynamic content rendering, code reuse, customisation choices, scalability, and the support of a thriving developer community, EJS has proven to be a great addition to our tech stack and improved the user experience on our recipe-sharing website. Due to its incorporation into our project, we were able to build a platform with a lot of features.
The writing for the logic of the webpages was not purely EJS and HTML, however. There was a need to use vanilla JS to add some animations and event listeners that were vital in providing a friendly user experience.

5.15. SQLite3

Using SQLite3 as the preferred database management system has its own set of benefits and limitations. It is critical to recognise SQLite3's scalability and performance constraints. SQLite3 might not be the best option for applications working with large data volumes or requiring high concurrency. Furthermore, this solution might not live up to expectations when sophisticated capabilities like replication, clustering, or high availability are required. A careful analysis of the project's unique requirements and limits is required because using SQLite3 could limit the application's ability to easily interface with other databases or services that use a different database management system.

However, SQLite3 has been known for its ease of use and maintenance as it is a lightweight, serverless, and self-contained database management system. Because of its simple setup requirements, it works especially well for small to medium-sized apps and provides developers with a hassle-free experience. Its versatility is increased by the fact that it can run on a variety of operating systems and programming languages.

6. Evaluation

6.1. Front-end

Since our website is data-driven, we used EJS, a templating language to ease the creation of a dynamic website. We used other web frameworks such as Express.js in Node.js applications that helped us render templates with data to generate HTML output.

The recipe-sharing website we created needs every client to create an account. We want to ensure that the recipe-sharing website shows the most reliable ranking and recommendation systems.

There is an endless scroll of recipes that makes searching for recipes exciting with endless options. Additionally, the search bar makes it seamless to search for recipes based on important keywords.

We aimed to create a user-friendly recipe website but encountered challenges related to Git usage. Issues like code overwrites and complex code fixes arose despite efforts to use descriptive commit messages and create branches. Our team's limited Git knowledge led to time-consuming processes to ensure correct file pushes. Over time, we learned to manage risks, stash changes, and prioritise effective verbal communication for smoother collaboration.

6.1.1. Future implementations (Front-end)

Our commitment to enhancing the recipe-sharing platform extends beyond the current iteration. We have identified several key areas for future development, each aimed at enriching the user experience and expanding the platform's functionality:

User Features Expansion: We plan to introduce new features, including user following and followers functionality. These additions will enable users to connect, fostering a sense of community and facilitating the discovery of recipes based on shared interests.

Collections Tab: To enhance organisation and personalisation, we intend to introduce a collections tab. This feature will allow users to categorise and curate the recipes they collect, creating a more tailored and user-centric experience.

Advanced User Settings: We recognise the importance of user customisation. Therefore, we aim to implement a broader range of variable settings. These settings will provide users with greater control over their experience on the platform, allowing for personalisation according to individual preferences and dietary restrictions.

Categorisation of recipes: Due to time constraints, some features were not implemented thoroughly. An important area in our project was to be able to filter recipes based on different categories easily. While we successfully created a search bar to facilitate category searches,

this takes a user more time to find a recipe. Our initial idea is to ensure that the best recipes based on a category can be looked up with a click of a button.

Nevertheless, categorising recipes based on user likes proved to be challenging towards the end of the project. Given we already had the framework for leaderboards and the supplementing pages, which necessitated significant changes to ensure the leaderboard functioned correctly.

As seen in the snippets below, these are the codes that are involved in the categorisation aspect. These categories are stored in the database as flags where 'Is_Chicken' is returned when the category 'Chicken' is selected.

```
function getCategoryColumnName(categoryName) {
  switch (categoryName) {
    case 'Chicken':
      return 'Is_Chicken';
    case 'Beef':
      return 'Is_Beef';
    case 'Pork':
      return 'Is_Pork';
    case 'Seafood':
      return 'Is_Seafood';
    case 'Vegetarian':
      return 'Is_Vegetarian';
    default:
      return null; // Invalid category name
  }
};
```

```
//category route
app.get('/category/:categoryName', isAuthenticated, (req, res) => {
  const categoryName = req.params.categoryName;

  // Clear the array for category results when switching categories
  loadedCategoryRecipeIds = {};

  const categoryColumnName = getCategoryColumnName(categoryName);
  const query = `SELECT * FROM recipes WHERE ${categoryColumnName} = 1 LIMIT 5`;

  db.all(query, (err, categoryResults) => {
    if (err) {
      console.error(`Error fetching ${categoryName} recipes:`, err);
      res.render('error');
    } else {
      console.log("error");
      // Add the initially loaded recipe IDs for category results to the global array
      addLoadedCategoryRecipeIds(categoryResults);

      // Render a category page with the matching recipes
      res.render('category-results', { categoryResults: categoryResults });
    }
  });
});
```

Figure 6.1.1.1 (Left): categorisation function Figure 6.1.1.2 (Right): category route

The category route follows 'CategoryName' where it corresponds to the category selected. An SQL query selects recipes where the category column is '=1'. This process retrieved matching recipes and rendered the 'category-result.ejs' page.

This continues and processes the recipes that have unique recipes and checks to fill in the page. Additional recipes are fetched to populate the pages.

```
$(document).ready(function () {
let page = 1;
const categoryRecipesContainer = $('.category-recipes-container');

function loadCategoryRecipes(category) {
    categoryRecipesContainer.empty();
    page = 1;

    $.ajax({
    url: `/category/${category}`,
    method: 'GET',
    success: function (data) {
        if (data.categoryResults > 0) {
            data.categoryResults.forEach(function (recipe) {
                $.get(`/category-results`, { recipe }, function (recipeHtml) {
                    categoryRecipesContainer.append(recipeHtml);
                });
            });
            page++;
            }

    },
    error: function (error) {
        console.log(`Error loading ${category} recipes:`, error);
    },
    });
}

// Event handler for category buttons
$('.category-button').click(function () {
        const selectedCategory = $(this).data('category');
        loadCategoryRecipes(selectedCategory);
    });
```

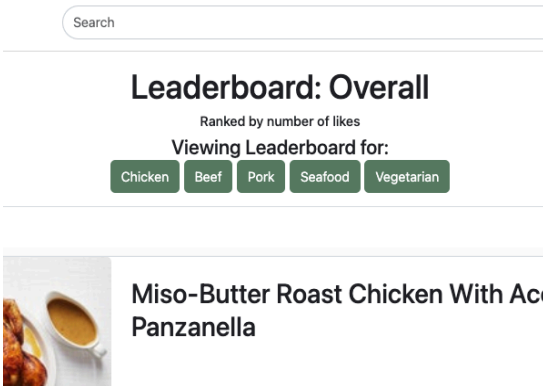

Figure 6.1.1.3 (Left): Load recipe to page with ajax Figure 6.1.1.4 (Right): Intended leaderboard page

We used Ajax (Asynchronous Javascript and XML) for making HTTP requests without refreshing the entire page, allowing the continuous retrieval and processing of recipes appended to an element in the EJS page.

However, to append the likes to the categories required extra configurations such as creating more routes for each category. Given that our existing routes were functioning well and to avoid potential errors and time used, we opted not to undertake these modifications.

6.1.2. Technical issues (Front-end)

- **Unavailability of data**

Early in our development process, we encountered backend data issues that posed significant hurdles. Given the data-centric nature of our website, these setbacks impacted progress. Delays emerged as we worked to retrieve essential information, such as recipe images, names, and steps. These data intricacies temporarily hindered web page development.

- **Displaying of the data and recipes**

To manage an extensive database comprising over 13,000 recipes and enhance user interactions, we adopted a strategy involving gradual content loading. Initially, our application fetches only five recipes from SQLite. As users scroll down, an additional batch of five recipes dynamically loads. This approach significantly enhances application speed and responsiveness. By loading a limited number of recipes at a time and progressively adding more during scrolling, we minimise initial loading times, ensuring efficient access to a vast content repository.

- **Leaderboard function implementation not working as intended**

Our current functionality allows users to like recipes, with liked recipes appearing on the profile page under 'Likes.' Our goal was to expand this feature to include aggregating total likes for

each recipe, ultimately populating the leaderboard. This leaderboard would rank recipes based on their accumulated likes. Users would also have the option to "unlike" a recipe, removing it from their liked section. However, we encountered an issue where unliking a recipe failed to trigger an update in the leaderboard rankings. This discrepancy could potentially lead to an inaccurate representation of recipe rankings.

6.2. Back-end

6.2.1. Conceptual issues

- **Search for recipe dataset**

While we were building the front end of the webpage, we did not confirm a dataset to use yet. We were confident that we could find a recipe that met our objectives. There were many datasets available on Kaggle but many lacked integrity. We considered an option to scrape recipes from other recipe-sharing websites, however, most online recipes were protected by copyright. The search was difficult, however we managed to find a dataset sourced from Epicurious that included the title, ingredients and instructions of recipes.

- **Storage issue**

The data we used was large and has >13,000 images and recipes. For this, we had to carefully select an appropriate database that fit our project specifications. We considered between NoSQL and SQL databases to store our data such as MongoDB and MySQL respectively. Both have cloud storage options which would allow the server to run at lower costs as compared to physical disk storage, and high customisability. To ensure we would be able to meet project deadlines, we chose to use SQL which most of our team was more familiar with. We used SQLite over MySQL as it was server-less and embedded in our codes made it easier to store and modify across computer systems.

- **Lack of advanced security measures**

In the final iteration of this project, we have allowed newly created recipes to be inserted into the database immediately. This comes at the risk of malicious and low-quality content being put on the website for other users in the community to see. Ideally, a website that allows user content creation should have basic moderation standards in place (Hetler, 2022). Some common ways for large platforms to moderate content are:

1. Hire a content moderation team to manually review and approve the legitimacy of user-submitted recipes before adding the information to the database
2. Utilise an algorithm that employs machine learning to automatically evaluate the legitimacy of user-submitted recipes
3. Rely on user reports to detect bad content

Such large investments are unfavourable for our independent project at the moment, thus these extensions were set aside.

6.2.2. Future implementations (Back-end)

Ranking system: Enhancing the ranking system to consider preferences across all users, rather than solely relying on a single user's ranking. A more advanced approach could involve creating user-specific ranking data. For instance, introducing a rating scale of 1 to 5 would provide valuable insights into users with similar tastes. This process would involve identifying recipes rated 4 out of 5 by a user, considering additional data such as a user's favourite recipes, and refining the selection to identify the top 10%. Implementing this personalised ranking system could utilise Python, along with tools like TF-IDF matrices and cosine similarity calculations.

Recommendation system: When entering a query into the search bar, it currently returns exact or closely related words based on specific semantic matches. However, it may overlook food items associated with distinct cuisines and cultures. For instance, a search for 'Asian' cuisines might yield recipes with 'Asian' in the title, but might not include specific dishes like 'Nasi Lemak' or 'Cold Soba'.

Developing an advanced search system might necessitate employing Natural Language Processing (NLP) techniques for text matching, ensuring a clear understanding of user intent and the relationships between food-related queries. This could be achieved through query understanding and creating a comprehensive food-knowledge database (Uber Eats, 2018).

Improving security: To safeguard user data and defend against potential threats, strong features like two-factor authentication, data encryption, and other security mechanisms will be implemented. This proactive approach to security will increase user confidence and guarantee the platform's long-term integrity, creating a safer environment for foodies to interact and share their passion for their favourite foods.

6.2.3. Technical issues (Back-end)

- **Risk of data loss**

Our project, centered around 'Epicurious' recipes, faces a challenge due to limited available users for constructing the recipe page at this stage. Typically, articles require an author who has created the recipes. However, since the provided CSV file lacks random users, we've associated each recipe with an 'Epicurious' account. Consequently, if this user account is compromised, there exists a potential risk of data tampering, management, or deletion. While associating a dummy account ensures comment and like functionality for a specific user, it may not be the most secure approach to link all recipes to a single user, as seen below.

```
// Call the functions sequentially(this must be done or it will load at the functions
//at the same time which will cause some recipes to fail when connecting to default user)
insertDefaultUser(insertCSVData);
```

Figure 6.2.3.1 linking author to default user

Possible solutions include implementing two-factor authorisation, locking the 'Epicurious' account from taking any new actions and periodically creating backups of information on Food Haven.

- **User base is too small**

The lack of user data makes it difficult to curate datasets based on relevant user behaviours. We are unable to obtain and refine dataset containing pertinent user interactions as such there is slow progress to understand the correct algorithm and evaluation metrics to create a personalised recommendation system. This required us to rethink our testing process. The compromisation of the proposed recommendation system also rendered our decision to make the actual content of the website available to authenticated users redundant.

- **Developed only for desktop**

As we are unable to host the website on the Internet, we only developed Food Haven for deployment on a desktop. As a result, users accessing the website on smartphones or tablets may encounter certain elements that do not display as seamlessly or aesthetically as they do on larger screens.

For example, some content cards or interface elements may not appear as visually appealing or may require excessive scrolling on mobile devices. This limitation can affect the overall user experience for individuals who prefer or need to access the website on their mobile devices.

- **Website downtime due to errors**

The hosted website faces downtime issues due to unhandled errors and bugs in the code, causing crashes and malfunctions. Restarting the server is a temporary solution due to a lack of experience with web hosting. This instability affects user experience and the platform's reliability, potentially leading to reduced user experience. Error handling and testing are essential to address this limitation and maintain consistent website availability.

7. Conclusion and summary

7.1. Front-end

Our recipe-sharing website's frontend development focuses on user-friendliness and aesthetics. We've enhanced user experience with fluid navigation, dynamic content loading, and interactive

elements. Features like user profiles, a powerful search bar, and infinite recipe scrolling are available.

Collaborative coding requires coordination to avoid conflicts. Technical issues, including slow data retrieval and error handling, were challenges. Despite these, we've made significant progress toward our front-end goals.

In conclusion, our front-end coding lays the foundation for an attractive and user-centred recipe-sharing site, fostering a thriving culinary community.

7.2. Back-end

Backend development ensures platform usability and data management. We've built a robust pipeline, handling essential features. This journey involved deep insights into data processing, SQLite3 database management, Passport.js for authentication, and user recommendations via collaborative filtering.

We successfully implemented user authentication, database integration, and dynamic content creation. Data handling and availability challenges led to creative solutions.

Connecting the backend to the frontend was constrained by time and complexity. Still, we've showcased backend capabilities through example output. Our team gained expertise in managing tasks like data parsing, database management, web hosting, and server-side logic. Despite difficulties, we've created a solid foundation for a seamless user experience.

7.3. Overall

The recipe-sharing website "Food Haven" is the result of our dedicated team's hard work, creativity, and cooperation. We faced and overcame a variety of obstacles during the project, displaying our resiliency and problem-solving skills. Through meticulous planning, extensive research, and dedicated teamwork, we successfully developed a dynamic platform with user-friendly features, robust authentication, and personalised recipe recommendations.

8. Individual Reflection

In this project, my primary role was front-end development, which involved coding web pages and certain features and writing certain sections of the report. Coordinating with teammates with varying schedules proved challenging but was mitigated through online communication. Managing an extensive task list prompted us to establish specific deadlines and milestones for efficient progress tracking. Despite these hurdles, collaborative efforts led to our success. This experience notably improved my teamwork skills. It provided invaluable lessons in full-stack web

development and the significance of effective collaboration, leaving me pleased with our achievements as a team.

9. References

1. *Body-parser*. npm. (n.d.). https://www.npmjs.com/package/body-parser
2. Hetler, A. (2022) *6 content moderation guidelines to consider*, *WhatIs.com*. Available at: https://www.techtarget.com/whatis/feature/Content-moderation-guidelines-to-consider (Accessed: 15 September 2023).
3. Hamad, F., Lui, I., Zhang, X., (2018) *Food Discovery with Uber Eats: Building a Query Understanding Engine, Uber.com.* Available at: https://www.uber.com/en-SG/blog/uber-eats-query-understanding/

10. Appendix

10.1. Work history

July 18, 2023

- Discussion of feedback given from midterm user testing
- Updating necessary changes to the high-fidelity wireframes

July 24, 2023

- Searched for a dataset and cleaned it
- Researched on type of user recommendation and selected one

August 1, 2023

- Report writing - Abstract, Outcomes and Introduction (completed)
- Setting up necessary libraries and coding environment

August 8, 2023

- Created a homepage card template
- Added a basic footer
- Implemented basic navigation without a sidebar

August 9 and 10, 2023

- Linked recipe cards to the recipe template page
- Cleaned up the template pages and routing
- Made basic footer additions
- Report writing - Planning and Research (started)
- Improved navigation and sidebar

August 13, 2023

- Made changes to button styles
- Updated like and comment design
- Created leaderboard routes and functionality

August 14 to 17, 2023

- Added leaderboard elements to the database
- Made improvements to navigation and the sidebar
- Created an About page
- Added a Settings page
- Created login and register pages
- Linked home page elements to the database
- Made various other fixes and improvements
- Report writing - Planning and Research (almost completed - left progression log)

August 18, 2023

- Implemented fully working user authentication and session
- Connected likes to user and recipe
- Created user_comments table
- Updated the comments table and package JSON

August 20 to 25, 2023

- Reconfigured database likes and comments
- Connected the recipes from CSV to default user Epicurious
- Created a profile page and updated its functions
- Added likes button for recipes
- Fixed the sidebar
- Report writing - Design and Development (started)

August 28 and 29, 2023

- Fixed the like button
- Fixed comment route and added comment anchor
- Fixed login button click

August 31, 2023

- Updated navigation hrefs and profile page layout
- Fixed infinite scrolling for the home page
- Added profile navigation
- Updated the schema for post count by user
- Added a search bar function
- Made the search bar work with unlimited scrolling

- Report writing - Design and Development (almost completed - left user testing, recommendation algorithm and some minor parts)

September 4, 2023

- Updated settings page functionalities
- Updated the schema
- Refined the infinite scrolling feature
- Made view engine configuration for Mac users
- Worked on the profile-page sub-navigation

September 5, 2023

- Edited the profile page functionality
- Designed the profile page card
- Added collaborative filtering function (user recommendation algorithm)
- Report writing - Evaluation (started)

September 11 and 12, 2023

- Updated settings page
- Added bio functionality and schema updates
- Added settings page functionality
- Re-fixed the infinite scrolling

September 16, 2023

- Removed categorisation parts
- Worked on the ranking part of the leaderboard page
- Styled elements in settings
- Report writing - Evaluation (completed - needs refinement)

September 15, 2023

- Added a route for updating settings
- Removed the log-out button from the settings page
- Updated index.js
- Implemented a fully functional "Delete Account" feature
- Fixed previous bad merges
- Enabled the ability to create recipes
- Worked on alt text for all images

September 16, 2023

- Continued development by removing categorisation parts
- Worked on styling elements in the settings
- Made Google Forms and sent out to our peers for evaluation
- Report writing - User testing (started)

- Report writing - writing the missing parts of the report

September 17, 2023

- Report writing - User testing (completed)
- Report writing - Refined the report (completed the report)
- Refined the code