Final Report

COMP3900 2022 Term 2

Computer Science Project

Project Title: Dinner Party

Group name: NewWorld

Submission date: 05/08/2022

Team Members:

Antoinette Ayoub (z5254617) Scrum Master, Developer
z5254617@unsw.edu.au

Tom Killingback (z5256086) Developer
z5256086@unsw.edu.au

Tiger Liu (z5160350) Developer
z5160350@unsw.edu.au

Zico Chen (z5248252) Developer
z5248252@unsw.edu.au

Derek Chen (z5289988) Developer
z5289988@unsw.edu.au

Table of Contents

Overview

Dinner Party is a collaborative, mobile-first web application that aims to help individuals organise, manage and plan at home dinner party events. Dinner Party allows users to find and contribute recipes, create events, and also invite guests, who will then be able to collaboratively contribute ingredients. Dinner Party strives to provide both hosts and guests with a seamless event management experience, taking the fuss out of compiling ingredient lists, shopping lists, guest lists and keeping track of calendars by integrating all of these functionalities into a centralised experience.

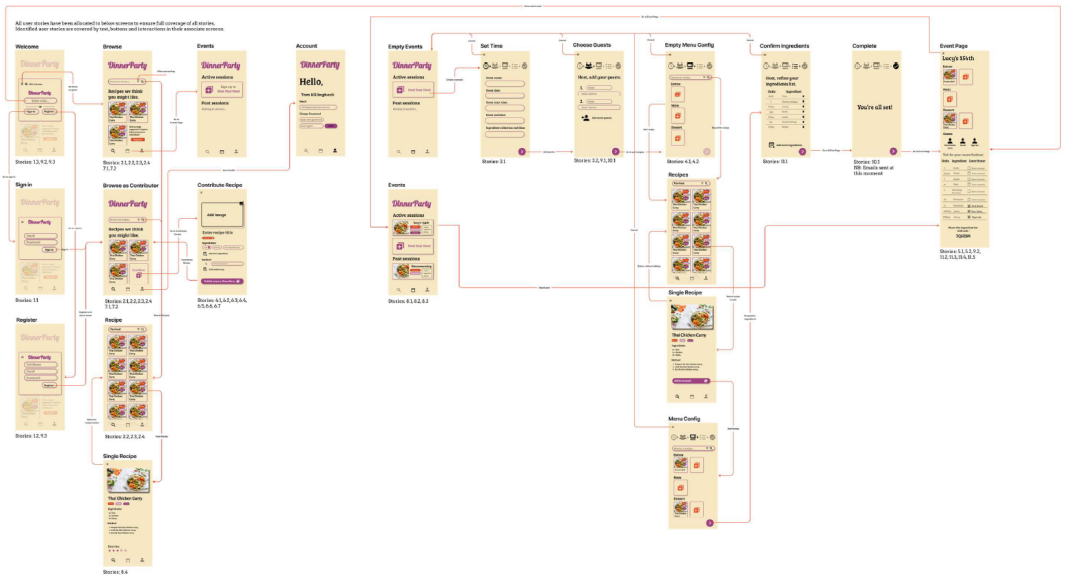
This application is the product of user-centred design, built by a team of agile developers with scalability and usability in mind. The below section explores both the overall system design and functionality, including the technology architecture and UX/UI design.

User Design

The user interface and experience of Dinner Party has been heavily styled based on the 1970's era of marketing material including colour palette, font and design aesthetic. This choice was made to invoke a nostalgic and homely feeling to the app, inviting users to engage and share the app on a 'novelty' basis.

Some specific instances of these design decisions include;

- **User Flow:** To ensure the user experience is intuitive and seamless, a full storyboard mockup was completed in Figma before work commenced in the frontend. Figjam was used in conjunction with this storyboard to map user flows and potential happy paths. By going through this process, the interface was well thought out and kept the user experience in mind.

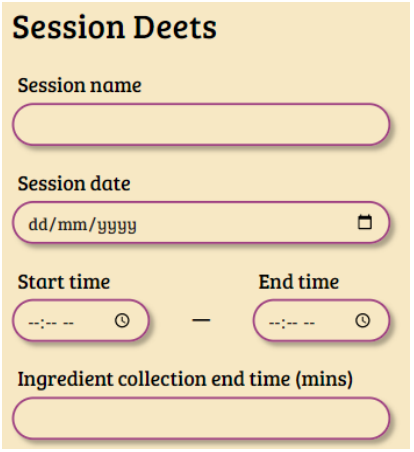


- **Colour Palette**: A playful colour theme of purple, beige and orange was selected as it is reminiscent of the faded plastic era (70's) with a hint towards it being related to plastic kitchen tools. Additionally, the use of purple and orange as highlight colours, adds an element of spontaneity and colour, evoking a sense of comfort to appeal to users for what is mostly a social application.

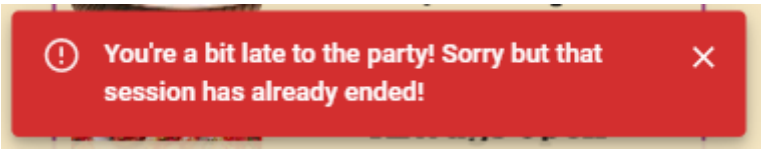
- **Accessibility & Contrast**: To cater to all users and provide an accessible user experience, the colour palette and all user elements were designed with accessibility in mind. To do this, the colour palette and use of colours was checked against guidelines and passed WCAG AAA standards. Further to this, general accessibility guidelines were obeyed including the use of alt-text on images and ARIA labels where required.

- **Typeface:** The use of Bree Serif as the primary font for all titles, information and subtext, also adds a playful and inviting tone to the application. As the application is intended to be used in a social context, this font choice was appropriately inviting and relaxed.

- **Radial design:** Rounded edges are consistently used throughout the application for input and containerised information. This decision was made as it reduces the clinical feel of data entry, and encourages the user to move from one entry to another.

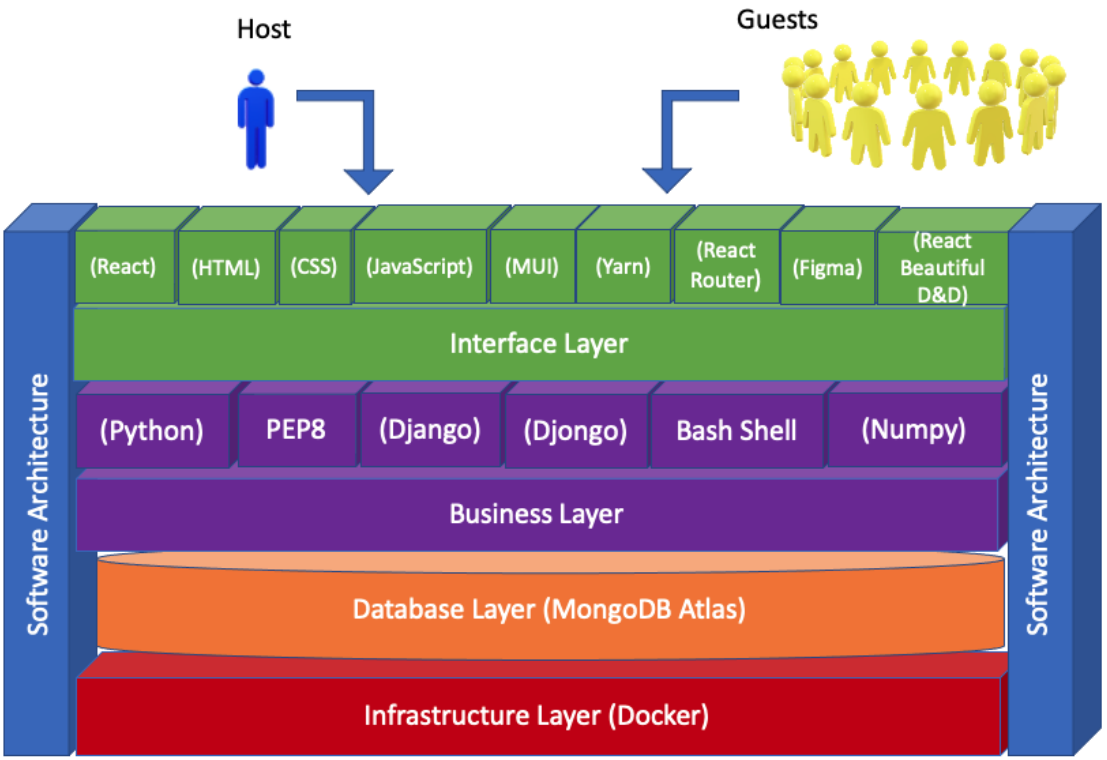


- **User friendly language:** Error messages are often sterile and harsh when exposed to the user. Although this is appropriate for some applications, for this context, it was seen to break the playful and creative theme. Therefore, all error messages introduced into the system are an attempt at humour and help the user understand what went wrong, why it went wrong, and how they can move to recover from the situation. This was designed to ease the user out of inconvenient situations.



Technology Stack & Architecture

The system architecture defines the structure of Dinner Party's software stack through the use of multiple frameworks, tools and languages, which contribute towards the presentation, business and data layers. The components of this architecture are deliberately selected to match the project objectives, requirements and resources of the team. Additionally, the team tried to push themselves to learn new and modern frameworks, whilst managing a tight timeline and advanced technical feasibility requirements.



Software Architecture Diagram

Presentation Layer

The interface and presentation layer consists of technologies, which a user sees and interacts with, forming part of the user experience. External actors interact with the system solely through the interface layer, which provides both a web-app experience for users, who can access it from any desktop or mobile client device with a web browser. It is important to note here that this is a mobile first application, therefore meaning all design decisions have been made with the mobile user in mind. External actors consist of unauthenticated users and authenticated users, who can act as hosts and guests. Both types of actors (hosts and guests) interact with the system through the same presentation layer and mostly the same user interface and experience. These actors interact with the presentation layer, which interfaces with the business layer, which in turn manipulates data in the data layer. This presentation layer allows all users to access the system over the internet with any client device browser.

Dinner Party is a single page web application (SPA), constructed using the ReactJS declarative framework, with associated technologies including HTML, CSS and Javascript.

Furthermore, key libraries such as Material UI (MUI) and React Router form part of the presentation layer. MUI provides primitive React components which form most parts of the web app whilst React Router manages navigation and routing within the app's interface. Finally, Django REST API is used alongside Python to interface between the frontend and backend through the use of basic CRUD operations and specific calculation endpoints.

Business Layer

The business logic layer implements business rules using programming logic. In this system, the business layer sits between the presentation and data layers, forming the backend of the system. This backend is required to receive requests from the presentation layer, apply some logic to it, respond back with some additional information and/or store the requested information to the data layer.

The business layer of the application was written in Python, utilising the Django framework in combination with the Django Rest Framework (DRF) to provide a quick, scalable and maintainable business layer for the developers to implement logic into. The Django framework abstracts data from the data later, into higher level and more usable internal objects which are simply manipulated. Similarly, DRF provides an abstraction from the actualities of the http request, exposing a high level request object, allowing the developers to focus more on the business logic rather than technical implementation.

Some important Python libraries also form the business layer, such as Djongo, which interacts with the data layer and a suite of utility libraries which are all outlined in the below 3rd part library section. NumPy is also utilised in conjunction with the data layer to implement the logic involved in the advanced custom recipe recommendation system.

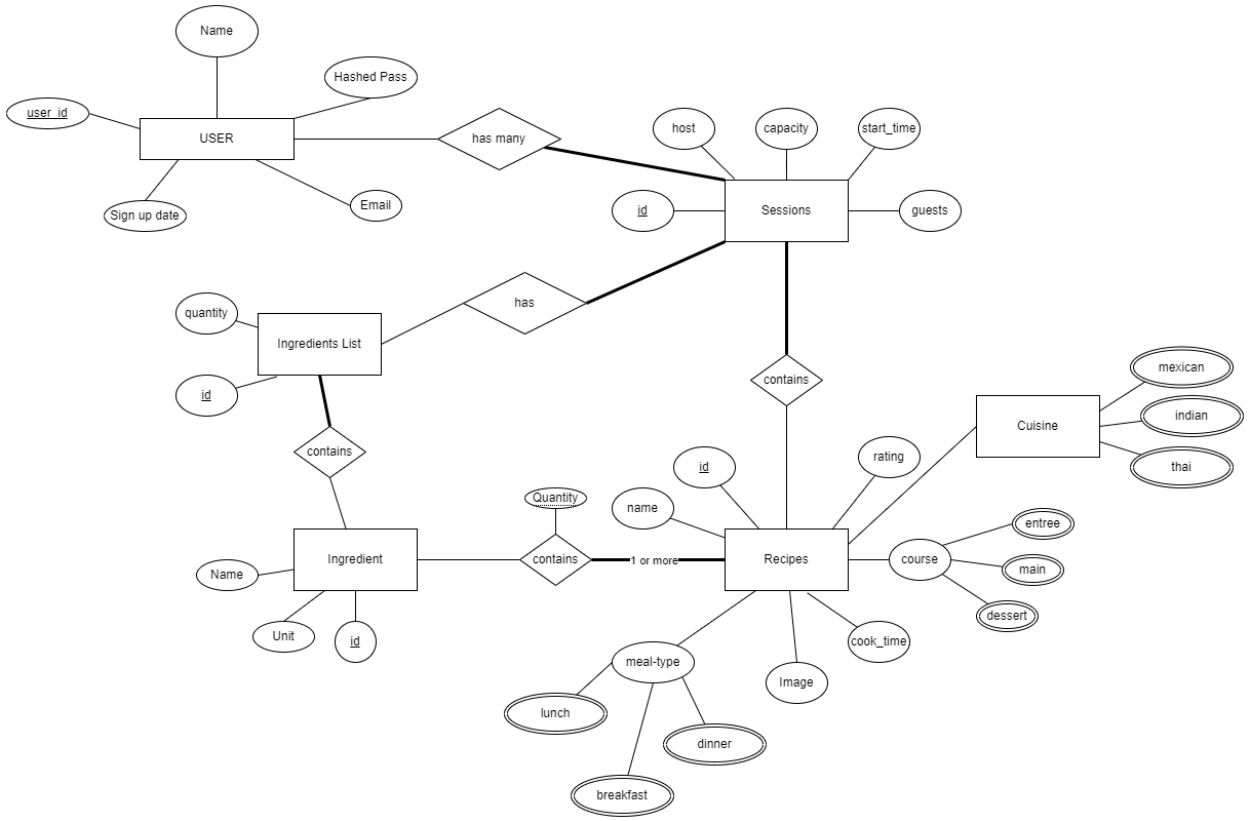
Additionally, there were some utility libraries used to accelerate common processes within the business later, namely uuid & django-iso8601. The UUID library is used to generate a universally unique identifier for the session code. This avoided the developers implementing this already solved problem. Furthermore, as dates and times are stored and operated as ISO8601 strings, it was convenient to use the django-iso8601 library to parse these strings into python date-time objects, to make logic and further operations more accessible.

Data Layer

The data layer is the system architecture layer concerned with data storage and retrieval. The selected database technology is MongoDB Atlas, which is a cloud based, NoSQL database.

Our dataset also stores a significant number of images as all recipes have an attached image. This image storage was implemented by serialising all images and storing them as text. This method was chosen as it is the most transferable and maintainable as it is independent of the specific data layer implementation.

Furthermore, the application context lended itself very well to a relational database schema, however mongoDB is not inherently a relational database. Therefore, a range of join tables were used to link various data objects between tables, such as users and their included sessions. Below shows the entity-relationship (ER) diagram as mapped for the data layer.



System Functionalities

The below section outlines the resultant functionalities of the Dinner Party app in production, and how they address the initially proposed project objectives.

Project Objectives / Requirements

A set of 11 objectives were formed in the Dinner Party project proposal to satisfy full coverage of the problem domain and all targeted functionality. **All core objectives have been successfully implemented**, and are mapped to the specific functionality present in the application at the time of submissions, as a demonstration of fulfilling the project proposal. Additionally, each functionality is accompanied by the specific user story(ies) that implemented this piece of functionality and their corresponding IDs on Jira.

New functionality which was not proposed in the original proposal, but has been implemented, is also clearly marked through the section, to make it explicit where extra work has been done in the interest of user centred design.

Firstly, some key terminology is defined below for the purpose of consistency:

Term	**Definition**
User	Non-authenticated user (everyone)
Host	Logged in user who has created the event (in context)
Guest	Authenticated and invited users.
Logged-in user	Authenticated and currently logged in user
Session	A Dinner Party event, consisting of time, menu and guests
Active session	A session until the scheduled event finish time
Contribution limit	Refers to the amount of time before the start of an event that invited guests can add, account for or adjust items in the shared ingredients list.

Objective 1: *User authentication*

Mapped Functionality	**Mapped User Story**
1.1 Users can log in with an email and password	*#25 #62 #63*
1.2 Users can register for an account with a name, an email and password	*#26 #64 #65*
Additional: Users can log out of their account. *This is core complementary functionality for user authentication, however was not specified in the project proposal.	*#110*

Objective 2: *Users can browse recipes*

Mapped Functionality	**Mapped User Story**
2.1 Users can view all recipes available in the database	#28 #83
2.2 Users can sort recipes by rating (ascending/descending)	#30 #86
2.3 Users can search for recipes by keyword	#31 #88
2.4 Users can filter recipes by meal-type (e.g breakfast, lunch, dinner, entree, main, dessert), cuisines, custom tags and cook time.	#29 #84 #101

Objective 3: *Hosts can create and manage a dinner party session*

Mapped Functionality	**Mapped User Story**
3.1 Hosts can specify the start and end times for a session	*#32*
11.2 Hosts can specify a contribution limit for the session as an amount of time before the event starts	*#32*
3.2 Hosts can choose number of attendees by adding guests individually	*#33*

Objective 4: *Hosts can add recipes to their session*

Mapped Functionality	**Mapped User Story**
Modification 4.1: Hosts can add recipes to one or more courses (entree, main, dessert) to create a menu *Included option to add to at least 1 course, or to as many as all 3 courses.	*#34*
4.2 Hosts can assign an infinite amount of unique recipes to each course	*#34*
Additional: Ingredients of a chosen recipe in a session are multiplied for the amount of guests invited *Implied but not specified in proposal	*#122*

Objective 5: *An ingredients list comprises of all the ingredients required for all selected recipes within the session*

Mapped Functionality	**Mapped User Story**
Modification 5.1 All ingredients across recipes in a session are combined into a single ingredients list before sharing with guests * Mention across recipes	#92
5.2 Ingredients of the same type across all selected recipes are collated	#36 #59

Objective 6: *Logged-in users can contribute recipes*

Mapped Functionality	**Mapped User Story**
6.1 Logged-in users can add recipes	#37 #69 #68 #80 #81
6.2 Logged-in users can attach the following metadata to recipes: 'Meal-type', 'Cuisine', 'Cook time', 'Photos' and customisable tags	#38
6.3 Recipes must have a list of ingredients. a) Existing ingredients should be able to be added through a search Function. b) Non-existing ingredients should be able to be defined and then added c) Ingredients should have specified unit(s) associated with them	#39 #40 #41
6.4 Recipes must have a chronological list of steps	#42 #78 #79
6.5 Logged-in users should be able to modify any recipe they contributed	#43

Objective 7: *A recipe recommendation system is able to give recommendations for hosts to select meals based on the trending meals and the host's historical choices*

Mapped Functionality	**Mapped User Story**
7.1 Users are able to see a list of recommended recipes	#44
7.2 Recommendations are based on the trending recipes, rating of recipes, host's historical choices	#45

Objective 8: *Logged-in users must be able to see a list of previous and current dinner party sessions*

Mapped Functionality	**Mapped User Story**
8.1 Registered users can view all events they have previously been invited to	#46 #53
8.2 Logged in users should not be able to edit any of their previous events	#47
8.3 A User can be a host and or guest in multiple active sessions	#48
8.4 Hosts can rate recipes that feature in past sessions	#50

Objective 9: *Hosts can invite guests to attend and contribute to the session*

Mapped Functionality	**Mapped User Story**
9.1 Host nominates guest names and emails	#33
9.2 Guests are emailed a unique code to access/read/write the share session	*#51 #52*
9.3 Guests are able to either login or register, but can access the information regardless	#53
Additional: An unregistered user who has been invited to a session, can register and view any previous sessions they've been invited to whilst unregistered *Strong incentive for a user to register	#53
1.3 Guests are invited to directly join the session via a code sent to their email	*#27 #51 #52*

Objective 10: *When the host creates a session and chooses guests, this creates a web calendar event that can be synced with a chosen calendar for each guest*

Mapped Functionality	**Mapped User Story**
10.1 The system should automatically send everyone a calendar invite for the time period of the event to the associated email address	#54

Objective 11: *All invitees to an active DinnerParty session can view and contribute to a shared ingredients list*

Mapped Functionality	**Mapped User Story**
Guests and Hosts can edit the ingredient list before the contribution limit	#55
11.1 Host can account for items before sharing with the group	#55
Additional: Guests and hosts can view who has been assigned to each ingredient **Not previously specified but important functionality	#57
11.3 Guests can view all un-accounted for ingredients	#57 #60
Modification: 11.4 A guest can assign themselves to an ingredient before the contribution limit	#55 #56 #58
11.5 At the contribution time limit, remaining ingredients form a shopping list	#6

Implementation Challenges

For a busy team consisting of many members with varying skill levels, expertise and availability, various implementation challenges arose due to unforeseen circumstances, technical complexities and a strict timeline.

Technical, Research, & Implementation Issues

Issue	**Action**
Data schema backward compatibility	Throughout the development of the system, the team often needed to reset the database back to a known state so they could test new / modified features. When doing parallel development with multiple developers trying to implement new features on the same dataset, they often conflicted with each other. This issue was resolved by ensuring that new features were all backwards compatible until everyone agreed to migrate and update the database schema.
Deadlines / time management	Due to the varying time schedules of all team members, there was often no time to do synchronous work. Better utilisation of time management, particularly allocating story points in Jira issues, may have yielded better allocation of the team's human resources and thus better use of time in the development stage.
Delegating responsibility	When implementing user stories & tasks, much of the responsibility assignment was split between the stack; database, backend and frontend. This presented integration issues during implementation, as issues within user stories had dependencies that stifled individual progress at times. Positively, this allowed for all team members to work within their expertise. A potential remedy would be for each person to implement features across the full stack, rather than concentrating on an end.
Dependency Issue - Python Packages	Challenges in setting up the development environment with Python packages. For example, the versions of packages need to be precise to be compatible with the project. This was not discovered until later in the development process when a package was updated and included breaking changes.
Django - Invitations Jazzband not implemented for v4.0+	When looking for a solution to send emails from DinnerParty with calendar attachments to invited guests, Django Invitations was the first option. However, this module was not compatible with our version of Django 4.0.5. The issue had been addressed in a github chain (https://github.com/jazzband/django-invitations/issues/182), but had not been released. The team resorted to a generic python email and smtplib package solution.
Gmail - less secure apps	Gmail limited their support in May of this year for 'less secure apps' which created a hurdle for the email and calendar invite solution. Dinner Party utilises a Gmail account that hosts the process of sending calendar invites to invitees when a session is created. However a persistent error led to generating an app password which exists as a secret in the code base.

Mongo Performance	MongoDB created serious, and persistent performance issues, which heavily impacted the development of this project. When testing full-stack features, and when running an instance of our program in general, the visual lag on the frontend was over 2 secs in some situations for DB information retrieval. This resulted in a frustrating user experience, so the team decided to utilise their credits and set up the full version for the final demo to maximise usability.
Frontend Drag & Drop	When implementing the create session functionality and the ability to reorder steps, internal user testing found a simple arrow re-ordering system of steps was onerous to the user. Therefore the frontend team looked to implement a drag and drop system to re-order method steps. This good concept was quite difficult to implement as it required the use of a completely new library to the team. This issue was resolved by reviewing a series of online tutorials and example implementations.
Django Serialisers	The Django framework which is being used for the backend embraces the MVC design pattern to link between the presentation layer and data layer. Unfortunately, none of the team members had used Django or any MVC-centred framework in the past. Therefore, the team found there was a significant learning curve to the Django Framework and the 'Serialisers' which are used as part of the MVC pattern implementation. This issue was overcome by the team teaching each other about how the backend operates internally.

Non-trivial Algorithms & Special Architectures

The below section outlines any non-trivial feature implementations and some special architectures that the team think are of note.

Recommendation System

Dinner Party uses a recommendation system to rank recipes in the browsing page, for both personalised (logged-in) and non-personalised (guest) users.

For non-personalised recommendations, the rank (recommendations) of a recipe will be based on its average rating, number of rating and number of views, and use a formula inspired by Bayesian inference.

For personalised recommendations, the team adopted item-to-item collaborative filtering technique, which is widely used by e-commerce websites such as Amazon (Greg Linden et al, 2003).

Each recipe in the backend has a hidden recommendation score for generic recommendations, forming the non-personalised recommendations, which is used to rank the recipes shown in the frontend. Recipes shown in the frontend will be sorted by recommendation score by default, the recipe with highest recommendation score will appear in the top.

Non-personalised recommendation score - Bayesian Inference

Dinner Party's recommendation score will consider average rating, number of rating, and number of views, and combine these factors to use a formula inspired by the Bayesian method, with assumptions on the number of a typical recipes's rating, views and a default score for an unrated recipe.

Ratings in Dinner Party can have a maximum score of 5, and users can rate a recipe with an integer between 1 and 5.

The following constants are defined:

R: the default recommendation score of a recipe, the initial belief.
W: a constant based on the number of ratings of a typical recipe.
V: a constant based on the number of views of a typical recipe.
M: the maximum view score.
α: view score coefficient.

And variables:

sum_ratings: the sum of ratings of a recipe.
num_ratings: the number of ratings of a recipe.
C: the number of views (click) of a recipe.

The recommendation score is calculated as follows:

$$score = \frac{R * W + sum\_ratings}{W + num\_ratings} + min(log_V C\ ,\ M)$$

The logarithm is used to smooth the view score, so that the recommendation score weighs more for the first hundred views but less so as the number goes larger.

The team compared the empirical results from most e-commerce websites where R **= 2.3**. That is; an item with 3 out 5 rating is generally better than an item without ratings as the default rating 2.3 is lower than it.

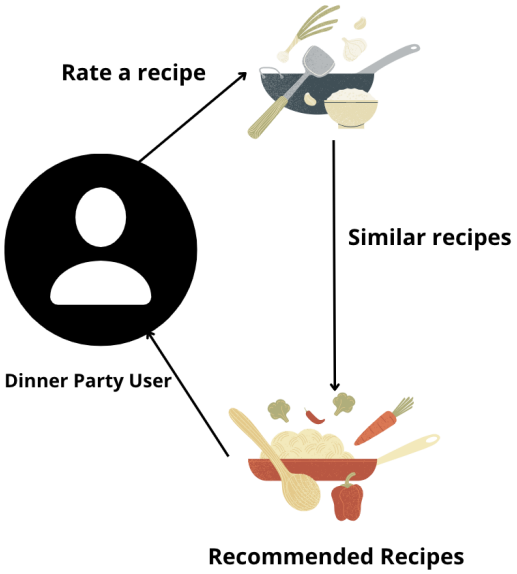
Given the current scale of Dinner Party, we set **W = 5**, and **V = 20**. This means we expect an item should have around 20 ratings and 100 views. These are two main assumptions we made in the calculation of recommendation score, and it should be adjusted as the application scales the number of users.

Additionally, we chose **M = 2** as an empirical result, meaning the recommendation score for a recipe should have a weight no more than 40% compared to ratings factors. Therefore, based on our setting, the recommendation score will be between 0 and 7.

Three examples:
An item has 5x 5-ratings, with 100 views, recommendation score is 5.197.
An item has 10x 4-ratings, with 300 views, recommendation score is 4.479.
An item has 1x 5-ratings, with 3 views, recommendation score is 3.117.

Personalised recommendation score - Collaborative Filtering

The personalised recommendation score will be used for logged in users to recommend personal recommendations based on what users have rated in the past. The recommendation system will also learn and recommend similar recipes to the user.



To achieve this functionality, we used the well-known collaborative filtering method from the recommendation system field. Collaborative filtering is a method of making automatic predictions about the interests of a user by collecting preferences or taste information from many users. It works by searching a large group of people and finding a smaller set of users with tastes similar to a particular user. It looks at the items they like and combines them to create a ranked list of suggestions.

There are two types of collaborative filtering, one is item-based collaborative filtering, which makes recommendations based on the similarities of items, and another one is user-based collaborative filtering, which makes recommendations based on the similarities of users.

We chose to use item-based collaborative filtering, as it achieves better results than user-based collaborative filtering in actual use. For personalised recommendations, we have set a personal score for each recipe in the backend, which represents the extent of recommendation, only when a recipe has a personal score higher than a set threshold, it will be listed as a recommended recipe and rank on the top in the recipes browsing page.

The personal recommendations will start by calculating the similarity score with each user rated item. We used the Pearson correlation coefficient to calculate the similarity score, which is a widely used equation to calculate the similarity score of two users or items. Then based on the similarity score and user's rating on those items, we calculate a personal score,and set a threshold, any recipe passing the threshold will be recommended, and will be ranked based on the personal score calculated.

When getting a personal recommendation, the user has to rate a recipe with at least 3-stars, and the recommendation system will calculate each unrated recipe's similarity score with the recipe, then recommend the most similar recipe on the top.

The formula is as follows:

Pearson correlation coefficient:

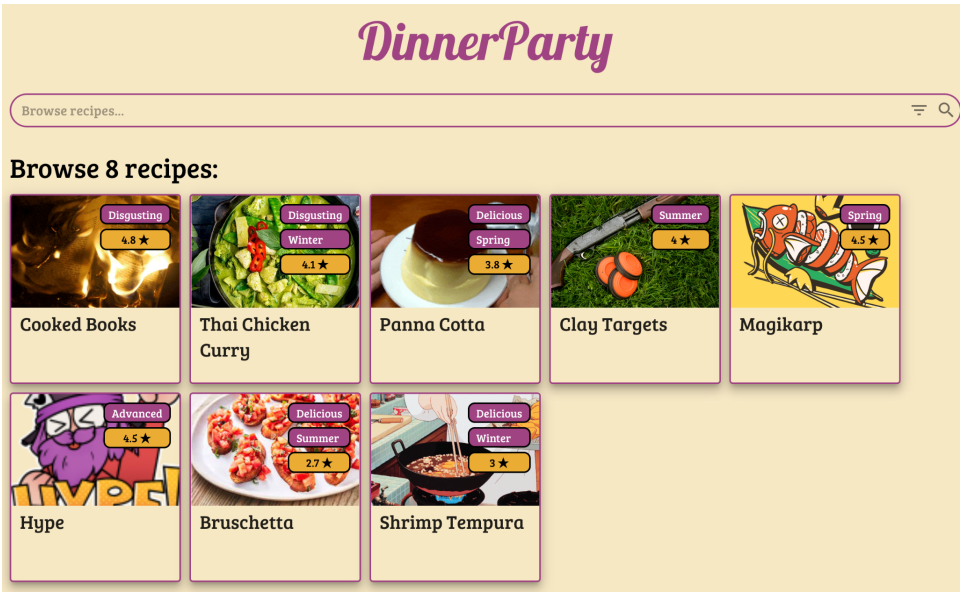
$$\text{simil}(x,y) = \frac{\sum_{i \in I_{xy}} (r_{x,i} - \bar{r_x})(r_{y,i} - \bar{r_y})}{\sqrt{\sum_{i \in I_{xy}} (r_{x,i} - \bar{r_x})^2} \sqrt{\sum_{i \in I_{xy}} (r_{y,i} - \bar{r_y})^2}}$$

Collaborative filtering personal score:

$$Personal\ Score(recipe,\ user) = \frac{\Sigma_{i \in ratedItems(user)}\ simil(recipe, i) * r_{user,i}}{\Sigma_{i \in ratedItems(user)}\ |simil(recipe, i)|}$$

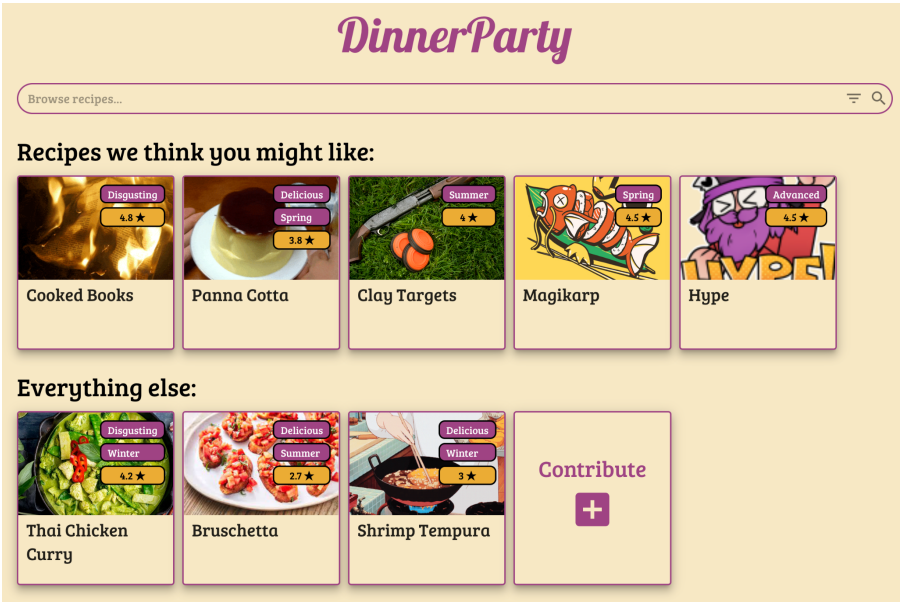
Recommendation system examples

1. View from a new user or non-login user page (non-personalised recommendations):



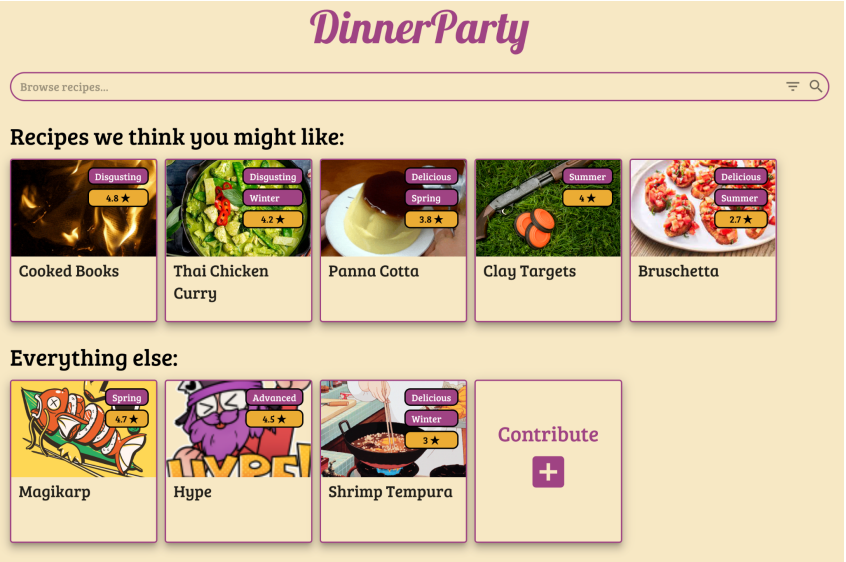
The priority of recommendation will be based on recommendation score (number of ratings, average ratings and click count). Cooked Books get the highest recommendation score because of its high average ratings, number of ratings and popularity. Though Hype has a 4.5 average ratings but it only has 2 ratings (5 and 4), its recommendation score will not be higher than Panna Cotta which has much more popularity despite an average rating of 3.8.

2. View from a new user A that have a strong preference on Thai food and has rated Thai Chicken Curry a 5-star:



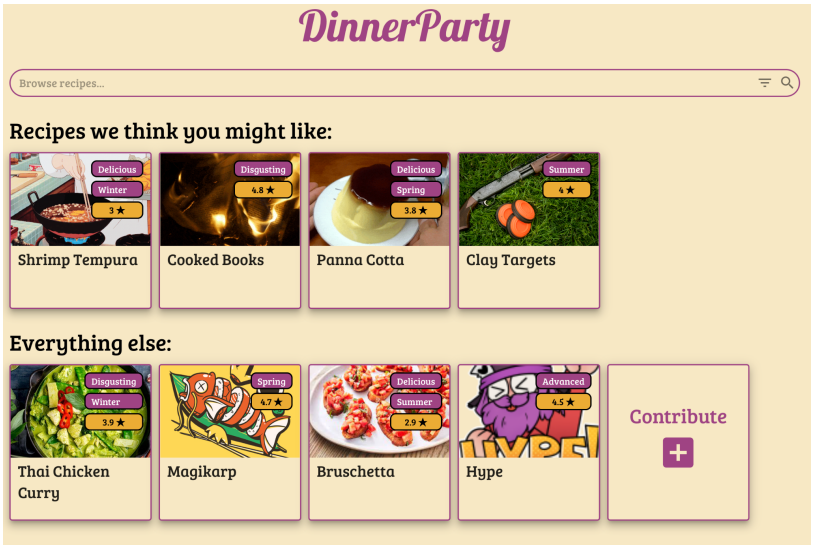
Based on the user's preference, the recommendation system will recommend most similar recipes in the database based on its personal score (calculated from item-based collaborative filtering and similarity score between items). Because there are many users both like and rated high a score for Thai Chicken and Cooked Books, the recommendation system will consider them as similar recipes and recommend Cooked Books for users that like Thai Chicken Curry. The rank of recommended recipes will be based on their similarity score with Thai Chicken Curry. Also if a user has rated a recipe with a low score, then similar recipes will have a smaller chance of being recommended.

3. View from a new user B that have a strong preference on fishes and has rated Magikarp a 5-star:



It will have a different view from user A, which will recommend recipes based on their similarity with Magikarp.

4. View from a new user C that rated 1-star for Thai Chicken and 5-start for Bruschetta:

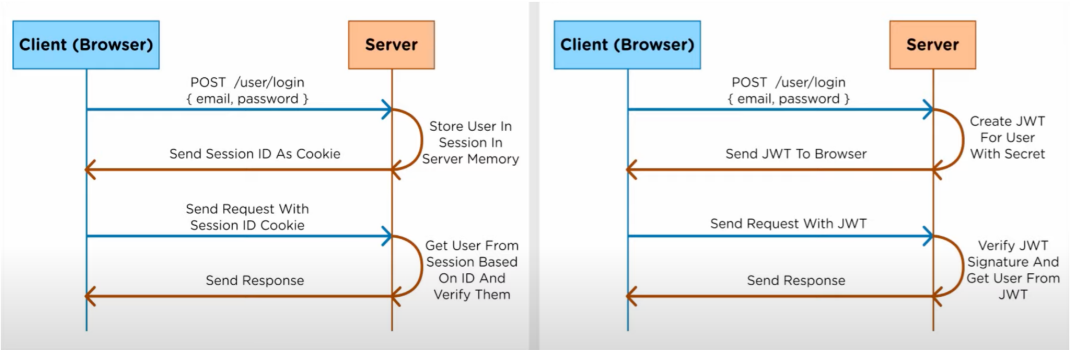


Magikarp has a high average rating but it is similar to Thai Chicken, which user C does not like, so it won't be recommended. Despite Shrimp Tempura's low average ratings and popularity, it's very similar to Bruschetta that user C has given a 5-star, so it will appear on the top of the recommended recipes. Also, recall that a recipe that has been already rated by a user does not appear as a personalised recommendation, since they are already aware of it.

JWT Authentication

We used the JWT authentication method to authenticate users. JWT is a lightweight and widely used authentication method. Unlike traditional authentication methods, JWT is a client side token, in which the server does not need to store and track the token, the token will expire itself based on a set time in the token body. This is particularly powerful and useful for

applications that do not need top level security, and subsequently we determined Dinner Party does not need a high level of security. The properties of JWT help save time and space for the server on maintaining the login and logout stages of each user. The below diagram shows the circular difference between traditional token and JWT token.



Traditional authentication versus JWT authentication
image source: YouTube, Web Dev Simplified

When a user login or register, the server will return a JWT token with expiration time and user id to identify the user. Every time a user sends a request from the frontend which needs authentication, the token should be included, and the backend is able to identify the user.

Autofilling previous session for new users

As part of the originally proposed features and objectives in the project proposal, we specified “An unregistered user who has been invited to a session, can register and view any previous sessions they’ve been invited to whilst unregistered”.

This feature meant there was a need to keep track of all sessions every unique email has ever been invited to within the system, such that, if that email was ever used to register an account, it is possible to link all previous sessions to the new account.

When considering this task, we assessed 2 main options;

1. At session creation, create a user for every guest email added to the session, but do not attach credentials to it. This user would also not be able to login until the user registers.
2. At user registration, query every session ever made and if the registering user exists in any session, create a link between the user and session.

Ultimately, we decided to go with option (1) mainly because querying and searching all sessions ever created is not a scalable approach to registering new users and this will drastically increase the time to complete the registration especially as the number of sessions and users grow.

As such, in the current implementation, at the time of creating a session, all guests are reviewed and if there is not a user that exists with that email address, one is created. In the case a user is created, this user is not able to login as they are registered as a ‘ghost user’

with a special value indicating this within the data model.All users are linked to sessions in the same way, whether they are a ghost user or normal user.

When this user comes to register, the system will first review the database of users and if one exists with that user (and it is a ghost user) it will simply set their new password as per the registration.

Live Updating of Collaborative Ingredients

To improve the collaborative experience when dealing with sessions and shared ingredient lists, live updating of contributions was implemented, similar to Google Docs. This was implemented through coordination between the frontend and backend using AJAX, which allowed for a background interval or clock to periodically check for updates and re-render the DOM accordingly.

This was a technical challenge as it involved the implementation of a specialised backend endpoint. To conserve data and maintain efficiency, the schema for this endpoint was created to only provide minimal data about changes, rather than resending the full ingredient list every time. This also required a special function to be written on the frontend to look through the list and insert these changes in place. Furthermore, the frontend has to deal with timer synchronisation, as this feature adds to the amount of asynchronous behaviour, which initially caused state conflicts and rendering problems. By refactoring the relevant code and restructuring it using the correct useEffect and useState hooks, these problems were overcome. As a result, the user gets to experience a seamless collaborative experience.

Styled Components

In the interest of code style and good practices, global CSS and inline styles or ‘sx’ were not used. This is due to the problematic nature of global CSS in React, where all files share the same CSS, leading to potential clashing. As a result, styled-components were used to define component styles, which were then dynamically mapped to CSS styles. In conjunction with typical CSS usage, special selectors and media queries were nested inside these styles to provide a dynamic and responsive design.

A trick that was employed was the definition of a theme and then the use of a useTheme hook to apply this theme to default MUI components. By doing this, many components were automatically styled according to the central theme. For styled-components, this added to the complexity as the theme had to be passed into the styled component as a prop. This was a difficult implementation and was a learning curve as the CSS could no longer be a normal string, but had to take in arguments.

Finally, due to the heavy custom theming required, the use of styled-components was made complex by the use of certain MUI elements. As additional customisation was required, this meant that each component had to be manually styled using a series of special selectors, which are defined in the MUI API documentation. For a component such as a TextField, this

meant that an average of 4 special selectors had to be used to achieve the desired aesthetic. This severely complicated the use of styled components.

React Router

To allow easy navigation through the browsers built in navigation and to allow links to be shared and saved, React Router was utilised. Whilst this is somewhat trivial and normal, two special tricks were used to achieve certain effects, which added to the complexity of the project.

Firstly, for routes for recipes and sessions, as an ID was involved, useParams had to be used to extract parameters from the path and parse it correctly. This implementation means that every recipe has a custom URL, which could be directly accessed, allowing users to be able to share recipes by a link, despite the Single Page Application (SPA) design.

Secondly, many pages or interfaces required nesting of modals or screens for multiple layers. For instance, when the user creates a session, they are on the session creation screen and the corresponding route. When they click on the recipe browser inside the session creation screen to add a recipe, they must be presented with a different screen. However, the route should not change as they haven't navigated away. Therefore, instead of using the conventional route, a built-in view had to be created to display the recipe browser within the same route. This continues, as a user is able to click on a recipe and should see the recipe details, but this should not change the route. This means that many components that were used inside their normal route, but also as an external viewer had to be refactored and designed such that they would be reusable under different circumstances. This required a lot of effort and added to the complexity of the design, as many overlays had to be supported, as well as the stacking of overlays and tracking of this state.

Local Storage

To allow for saved sessions, meaning the user does not have to manually log in every time they use the app, local storage was utilised. Whilst the use of local storage is trivial, due to the use of useStates, desynchronisation issues could occur, where different states conflict with the local storage save. Therefore, a custom useLocalStorage hook was created on the frontend to allow for local storage to be used similarly to a useState. This hook acted the same way, but allowed for a default value and also saved the local storage key every time the state was changed, ensuring the local storage was always up to date with the app's state.

Third Party Services

Within a limited timeframe, DinnerParty was developed with balancing educational outcomes, production efficiency and product quality in mind. These constraints justified the inclusion of third-party services within the build as specified below:

Frontend

Figma

Figma was used extensively in the prototyping and proposal sections of this project to help us guide our ideation creation and user stories and user story paths. This meant when it came to developers implementing specific features, there was no question around what needed to be built. Additionally, Figma helped us to style our actual development, allowing us to export specific CSS features in advance of the actual development. Figma holds the open-source creative commons licence, meaning anything created on the Figma platform is owned and can be used by the creator.

Figjam

Like Figma, FigJam was used on the free starter plan, with access to unlimited personal whiteboards and 3 shared whiteboards with unlimited collaborators. Fig-jam was used to map ideas and Figma story boards together in a collaborative way.

React

React was used as it is the most common Javascript framework used in industry and was most commonly known within the developers. Additionally React has a significant number of open source libraries which also made it an attractive choice over Vue or Angular JS. Reat uses a MIT licence, meaning it is free to use the code base as long as the licence is included.

Javascript

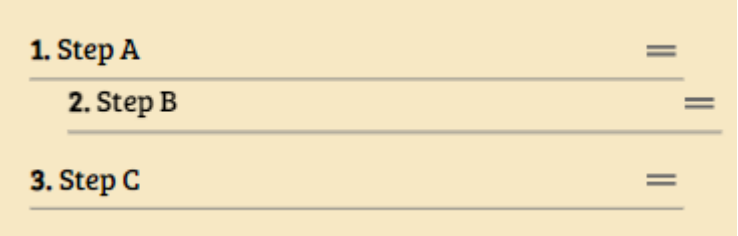
Javascript was the obvious choice for frontend development as it is the ubiquitous web language for advanced functionality. Javascript uses the Apache Licence 2.0
Licensed under the Apache License 2.0 meaning we are free to use the language and any parent covered by the software.

Material-UI

Material UI (MUI) is a react library that provides a large suite of commonly used components and building blocks for frontend web applications. MUI was used as it has been extensively used by the team's developers before and it is a very common library in industry. Additionally, MUI takes out much of the 'template' code needed to build advanced applications and was therefore preferable for its time saving. MUI uses its own standard licence which explains that MUI can be used for personal development but cannot be directly sold in a product. MUI offers a paid version for more commercial applications and this would need to be used if Dinner Party was scaled in the future.

React Beautiful D&D

Additionally, A process of internal usability testing determined that when ordering steps for a new session (see below for functionality), the ordered drag-and-drop system was the most usable. Therefore, the library 'React Beautiful D&D' was used to simply and time effectively implement this feature. This library also uses the Apache 2.0 licence meaning the library can be used under almost any circumstances.



Yarn

Yarn was the chosen package manager for this frontend application since it was most commonly know by the developers who were going to use it. In addition, yarn instals large packages in parallel compared to npm which instals sequentially. This can be a very large time savings for rapid development. Yarn uses the BSD 2-Clause License, meaning the application can be used for any application, however the original licence must be included in any redistribution.

Backend

Python3 & built-in libraries

Python was a reliable and scalable choice for the DinnerParty backend to be built. All developers were familiar with the language which made core logic development simpler to start, with minimal barrier to entry. Python core built in libraries, including datetime (REF) and email (REF) and the object-oriented nature were all attractive features for this full-stack project.

Bash Shell Scripting
Bash shell scripting is utilised within this project for tools that aggregate build and run procedures across dependencies, modules and frameworks into single programs to simplify setup during development and production environments.

Django 4.0.5
DinnerParty backend utilises Django-REST API. Whilst Flask was the initial choice as a web server framework, primarily for the familiarity factor amongst all team members, Django provided an opportunity to work with a new framework beyond Flask. Django is an older framework, known for its use in rapid development and prototyping, which was preferred within the time constraints of this project over the simplicity of Flask. Further favourable features utilised include the object-oriented mapper, clear layout of the models structure and no-code generation.

Djongo v1.3.6
Djongo was used as the Django framework to connect Django to the MongoDB Atlas database. Djongo was chosen over something like PyMongo as it provided more flexible data types to be used in the Djongo models - namely, Djongo provides a list field and advanced JSON fields where PyMongo did not.

Database

MongoDB Atlas
When deciding on a database service, MongoDB was selected due to its ease of use, convenience and novelty, which is in line with the requirements of the project. This database stores all persistent platform data and facilitates updates and queries. Our preference was a cloud-based NoSQL database. Similarly, in the early stages of development, we valued the “JSON-like document data model maps to the objects in your application code”.

The Atlas web-UI, viewable on any web-device requires minimal setup and is easily usable. Additionally, this simple cloud hosted data layer, allowed all developers to have an inherently synced state, to ensure we were working on the same data structures. This also helped for user testing as it was simple to spin up the front and backend server on a remote machine, without moving the data layer.

Full-Stack / Production

Docker
To simplify the deployment process, and cater to functionality across machines and binaries, the build and deployment of DinnerParty utilises Docker.

For the frontend deployment, a nginx server is used to host the frontend source code. This nginx server uses a stripped down, production build of the react app. This improves load times and server loading as there is minimal extra debugging information. The frontend is then built into its own docker image named ‘dinnerparty-frontend’. The frontend is deployed on the standard non-secure port 80.

For backend deployment, a docker image is built to isolate all backend logic code and required deps into a constant environment. The server is deployed and exposed from the container on port 81.

A docker compose file is used to set-up and configure the two docker images to build and then run together in a single ‘stack’ which includes frontend and backend.

A simple run script has also been created to ensure the host machine has docker installed before attempting to run the system, and if not, will attempt to install it or prompt the user to install themselves - based architecture.

User Documentation

How to Build, Run & Configure

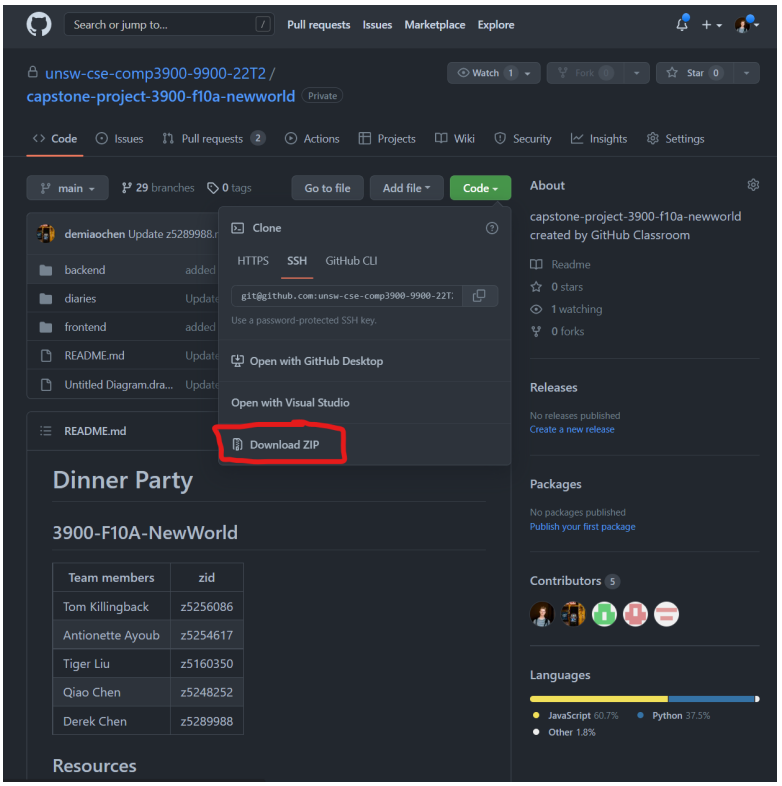
The following section outlines how to run the system.

This submission uses the Virtual Machine submissions method.

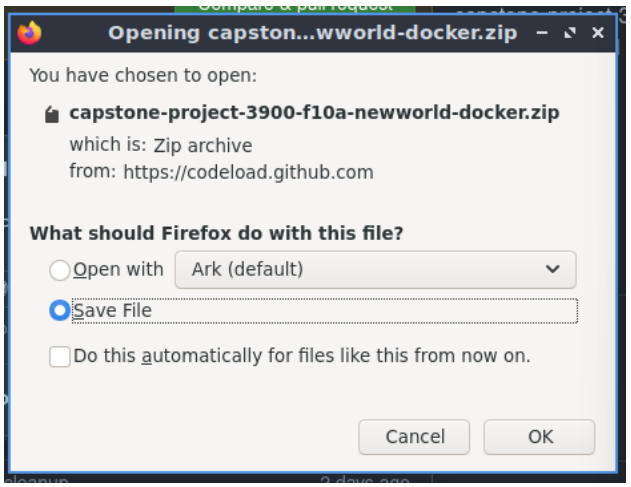
Virtual Machine Setup Guide

Virtual Machine will be referred to as ‘VM’ for the remainder of this report.

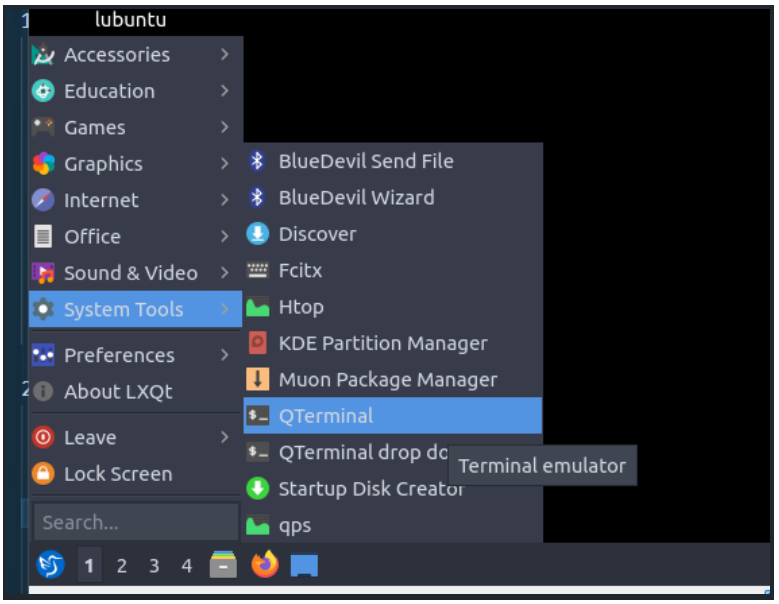
1. Sign into github using your authenticated account for this project.
2. Open the source code repository here: https://github.com/unsw-cse-comp3900-9900-22T2/capstone-project-3900-f10a-newworld
3. Download the source code as a zip file. *Note: we are downloading a zip file here to avoid setting up ssh keys. The image below shows how to do this.*



4. Select ‘Save File’ and ‘ok’ in the pop-up



5. Open a terminal. In lubuntu this can be done by selecting the QTerminal as shown below:



6. Navigate to the downloads folder on your machine.

```
cd ~/Downloads/
```

7. Unzip the source code by typing the following command:

```
unzip capstone-project-3900-f10a-newworld-main.zip
```

8. Navigate into this repo's root level folder:

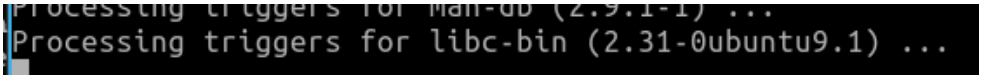
```
cd capstone-project-3900-f10a-newworld-main/
```

9. Run the system.

```
./run_system.sh
```

NB: You may be prompted for a password, if so, enter the computer user's password. For the provided lubuntu vm, the password is `lubuntu`. Type this password and press Enter.

NB: The system may hang on the following line for a few minutes, this is normal

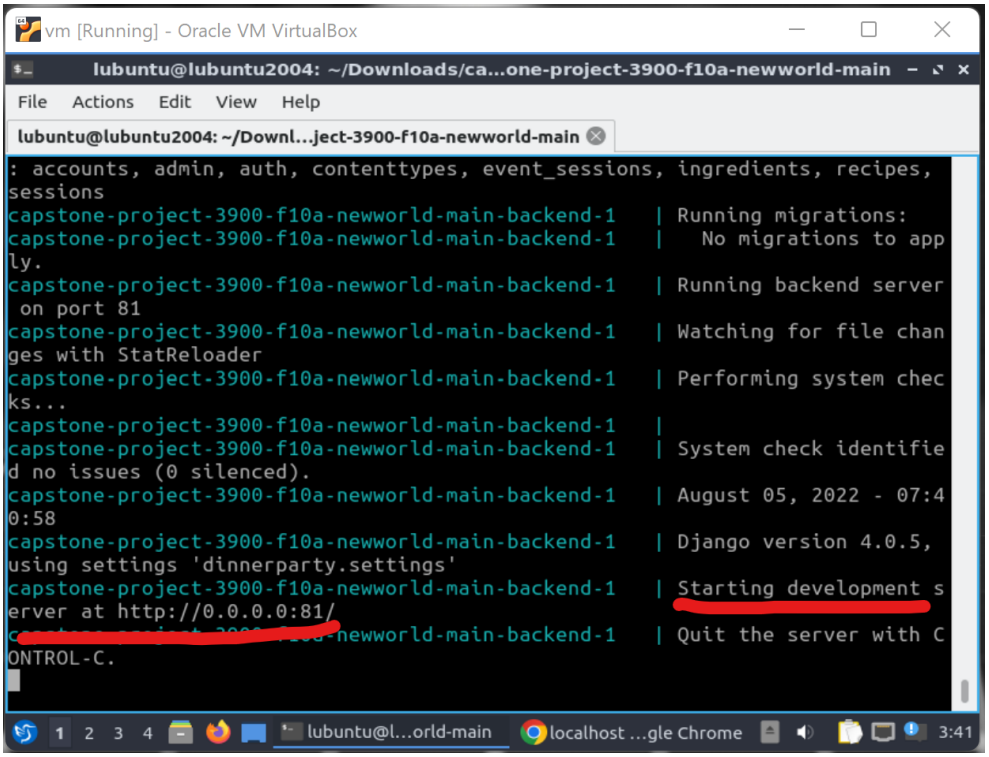


NB: The docker errors shown during the build are normal and you can ignore them. These errors are docker attempting to download the image from the public docker registry, however since the project is private, it must compile them.

NB: The entire system build should take approximately 10 minutes on the vm.

10. When prompted to set a default browser, select 'ok'. This will open a google chrome window and show 'no connection' or similar whilst the system continues to build. Go back to the terminal window.

11. Wait for the docker container to complete its startup process. The below image shows the output when the docker container has completed the startup process.



12. Visit the web user interface on your web browser of choice (Chrome is recommended) at http://localhost/.

Local Machine Setup Guide

This setup guide assume 3 things:

- You have docker installed and available on the command line. Instructions for installing docker can be found here.
- You have git installed and available on the command line. Instructions can be found here.
- You have a ssh-key setup between your local terminal and github account. Instructions for generating and then setting up ssh keys are linked.
- Your github account is authorised to access this team's github repo.

Instructions.

1. Open your terminal of choice and navigate to where you would like to store the source code. We recommend your home folder. Get there by doing the following

```
cd ~
```

2. Clone the source code with the following command.

```
git clone git@github.com:unsw-cse-comp3900-9900-22T2/capstone-project-3900-f10a-newworld.git
```

NB: This command should be entered as a single line, however has rolled over to two lines for the purposes of fitting on this report.

13. Navigate into the project source code

```
cd capstone-project-3900-f10a-newworld/
```

14. Run the system

```
./run_system.sh
```

15. You may be prompted for a password, if so, enter the computer user's password.
16. Visit the web user interface at http://localhost/.

How to Use System Functionality

The below section outlines how to use all implemented functionality as well as plenty of extra information. Sample data and a testing user has been provided for your convenience. The information can be found below.

Email: eric@eric.com
Password: 3900

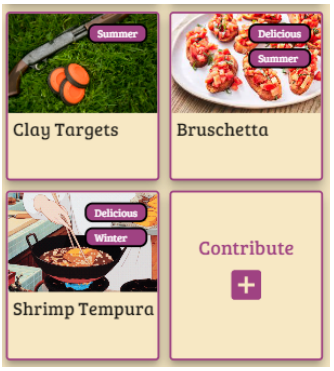
Recipes

The following section outlines all available functionality for interacting with the recipe entities throughout the application.

Create Recipe

1. To create and contribute a recipe, first the user must be logged in. Details of this process can be seen below. Once logged-in, navigate to the browse screen by selecting the magnifying glass in the bottom navigation bar.

2. If logged in, the contribution card will appear at the end of the page of recipes. Select this card.

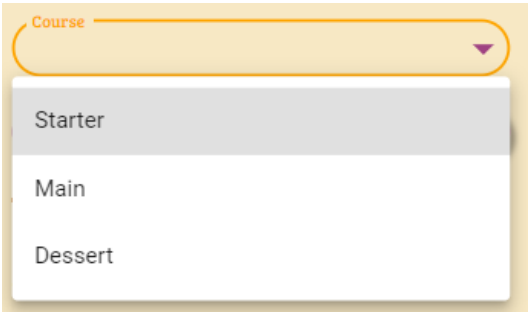


3. The following page should appear after selecting the contribute card

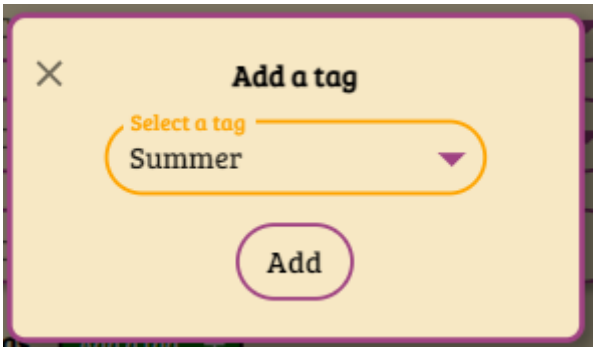
4. Select an image for the recipe but clicking the ‘Add image’ card at the top. This will prompt the user to select a png or jpeg image from their local machine.

NB: Supported image types are: .png .jpeg, .jpg, .gif

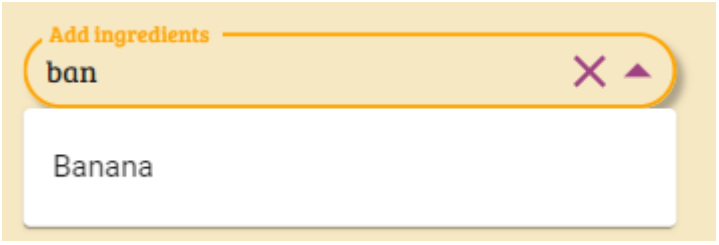
5. Use the drop down menus to select the Cuisine & Course.



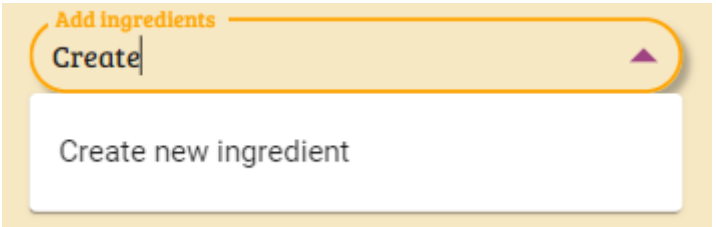
6. Enter a cook time in minutes.
7. (Optional) Select ‘Add Tag’ and use the popup to select one or more tags to add to your recipe.



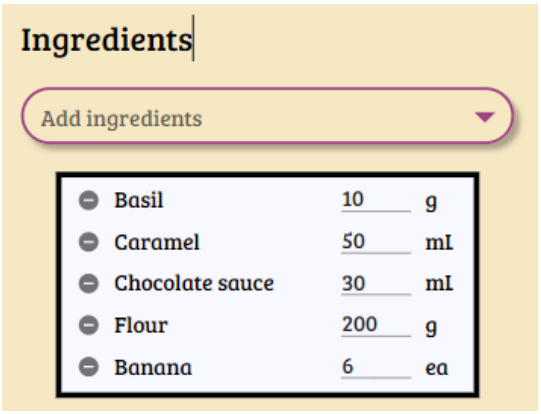
8. Add Ingredients
    a. Add an existing ingredient know to the system by searching for it in the ingredients selection box



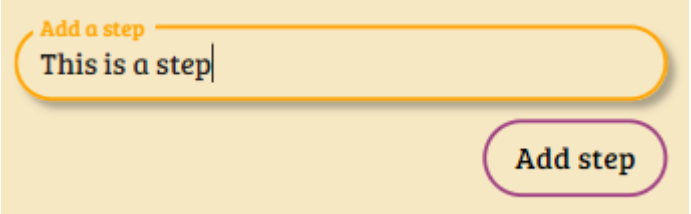
    b. Create a new ingredient not known to the system by selecting ‘Create an ingredient’ from the list of options. Use the pop-up to enter the details of the new ingredient type.



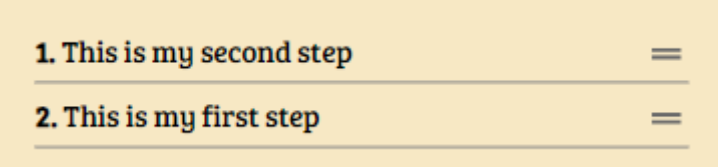
9. Give the ingredients a quantity by putting a number next to each unique ingredient. Ingredients can only be added once. All ingredients must have a quantity.



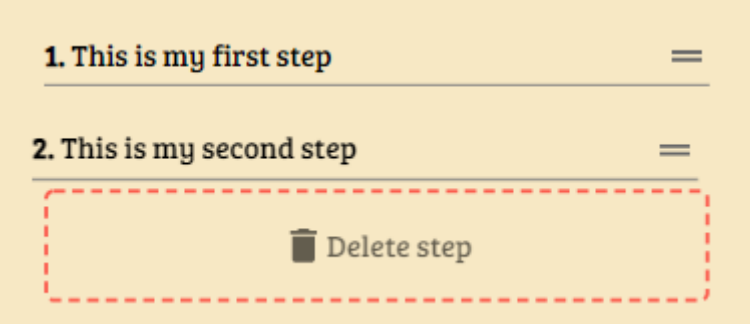
10. Add the method steps one by one by entering the step description and pressing ‘Add Step’



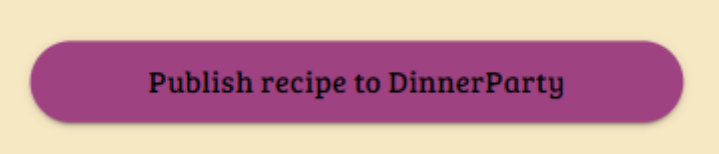
11. Steps can be re-arranged by selecting the handle on the right hand side and dragging them to the correct order



12. Steps can be deleted by dragging them to the ‘Delete step’ area. This area will appear once you have started to drag the step.

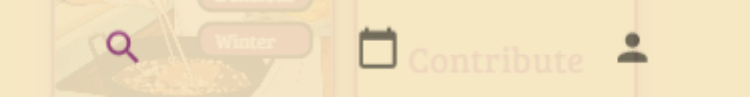


13. Select ‘Publish recipe to DinnerParty’ once the recipe is ready.

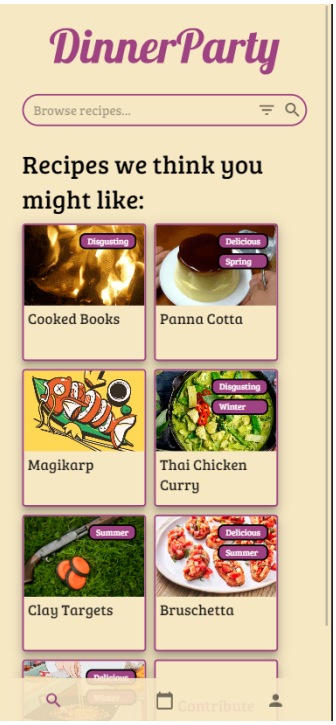


Browse Recipes

1. Browse recipes is done from the far left tab of the bottom navigation bar. Select the magnifying glass icon to see all recipes.



2. All recipes are shown on this home page.



Note: Recipes shown here are ordered based on their overall recipe ratings.

3. Recipes can also be filtered and sorted based on any field, to sort a recipe enter the keyword in the search bar located at the top of the page. To filter or sort results of a search or the browse screen, select the filter icon on the right side of the search bar

   a. From within the filter & sort page, select the desired filters and click ‘Apply’. Filtering can be removed at any point by selecting the ‘Reset’ option from this page.

Update Recipe

1. Select the ingredient from browse page
2. Change any entity as required
3. Select the ‘Update recipe’ button from the bottom of the page

Sessions

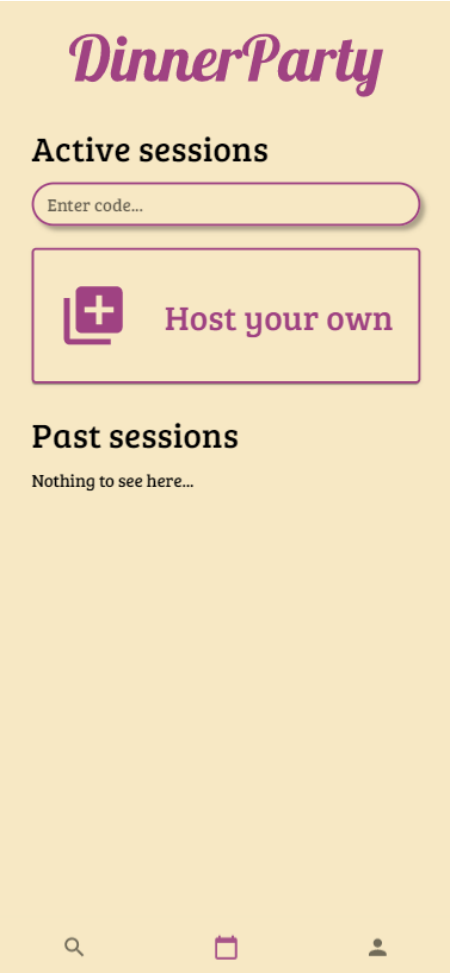
The following section outlines all available functionality for interacting with the dinner party session entities throughout the application.

Host a session

The core functionality of the application is to be able to host and attend dinner party sessions. This section will outline how any logged in user can host their own, assign recipes, invite guests and configure the session.

1. Navigate to the sessions screen by selecting the middle icon of the navigation bar.

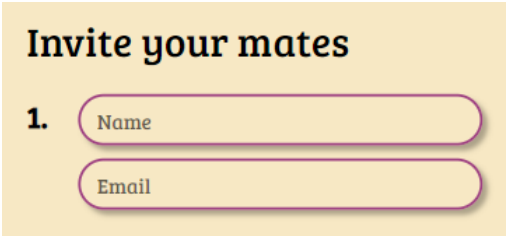
2. Once on this page, a user will see all of the sessions they have been invited to or have hosted. For setting up a new session, select the ‘Host your own’ card at the top of the page



3. The following page shows the process at the top of the page, allowing the user to navigate through the process, as well as the first set of data entry for the session details.

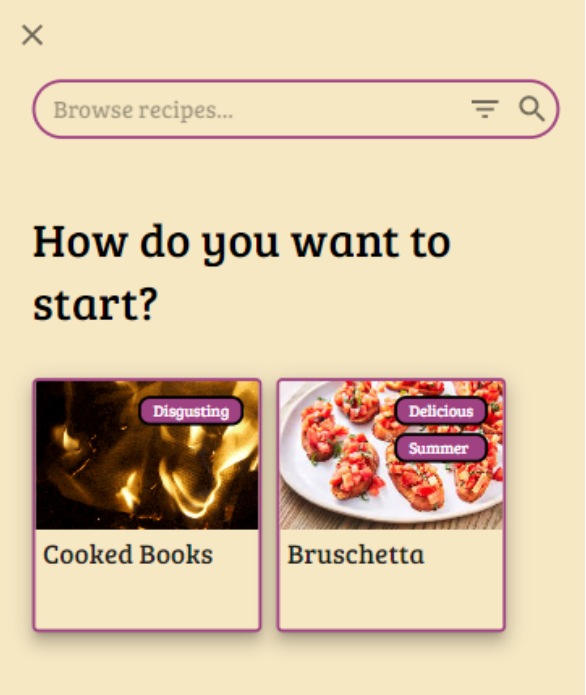
4. Enter the session details including name, date, start and end time and the ingredient collection end time.
    a. If the end time rolls over 11:59 of the session date, it is assumed the session will end the next day.
    b. Select the clock icon within the session time selection to use the user-interface to select a time
    c. Ingredient collection time refers to the number of minutes before the start of the session that the system will close ingredient contribution. This can be used to reserve time for shopping.

5. Invite others. Enter the name and email address of guests. After one guest's details have been entered, the UI will automatically prompt the option to add another. Select the arrow in the bottom right corner when ready to continue.

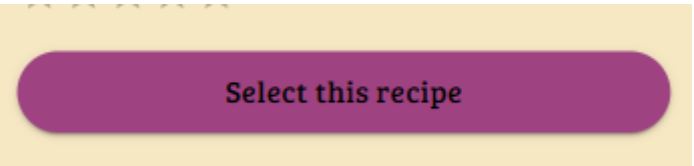


6. Select Meals. For each course, the user has the option to select as many meals of that type as desired. This can be done by clicking the ‘Select’ card in the desired course. For example, the starter options are shown below.

    a. *Note: It is possible to filter and search in this section also*

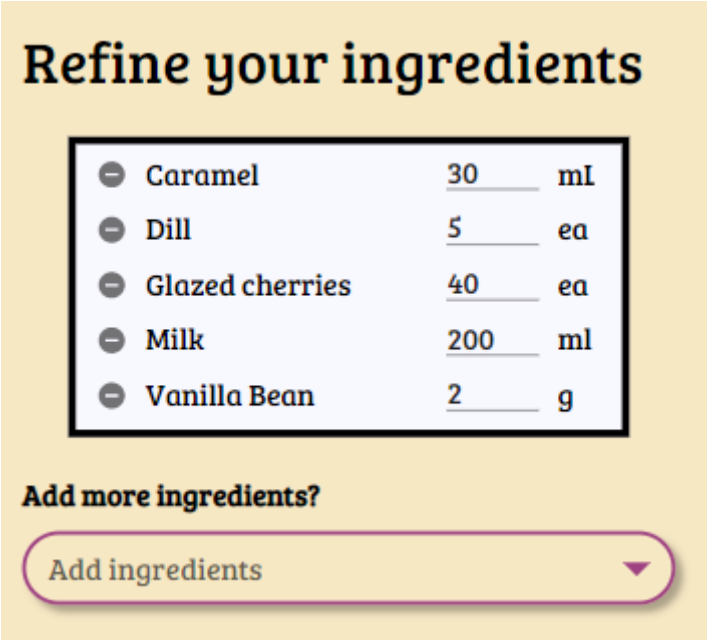


7. Once the desired recipe is found, select the car and once the detailed view appears, select the ‘Select this recipe’ button at the bottom of the card.

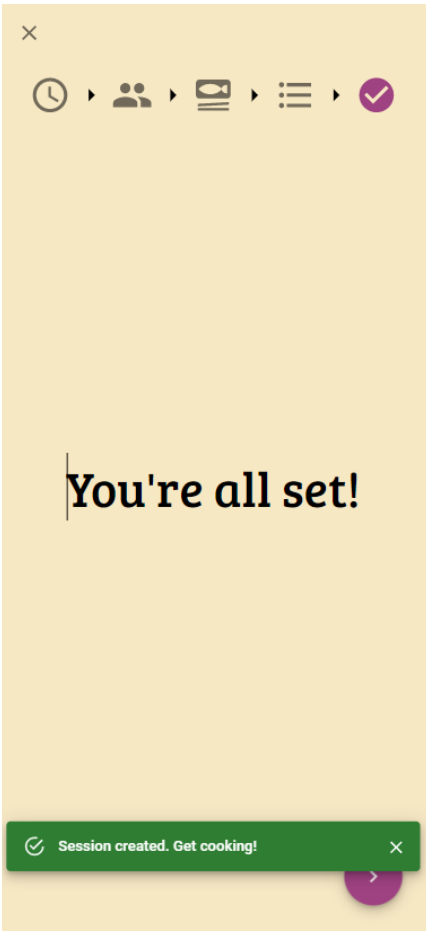


8. Repeat steps 6 & 7 for all desired courses and recipes. Once complete, select the next arrow in the bottom right corner of the ‘Choose your menu’ page.

9. Next, review the ingredient list provided. This ingredient list is calculated based on the number of guests attending (plus the host) and the ingredients between all ingredients. This ingredient list can be modified, added to and removed from as per what the host desires.
    a. Modification - Change the quantity of ingredient next to its name
    b. Delete - Select the minus logo to left of the ingredient name
    c. Add - To add ingredients, start typing the ingredients name in the bottom section, once the ingredient selected it will appear in the ingredient list where you can assign a quantity to it.



10. When happy with the ingredient, select the next arrow in the bottom section of the screen. If all was successful, a success message and screen will be shown as depicted below. The session is now open!



View session via account

It is possible for any logged in user to view any sessions they have coming up, or any session they have been to in the past.

1. Login to the application as described in the section below

2. Select the browse page from the middle of the bottom navigation bar.

3. This page will display all the necessary information related to upcoming and past events. Each card represents a session displaying information such as the number of people attending, the number of recipes on the menu and the total cook time of all recipes. Additionally, sessions which have not yet been looked at, will be shown a ‘new’ badge to invite the user to look at the new session.

4. To view a session, simply select the session card and all details will be displayed.

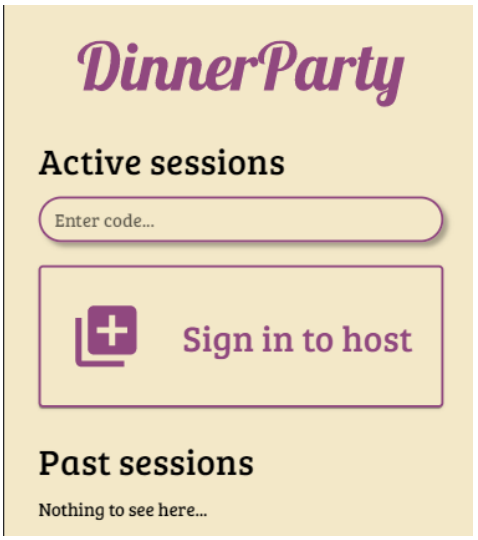
View session via code

Any logged in or non logged in user can view a session by entering a unique code which is created for each session. This code is emailed to all guests and can also be seen on the session page and then shared directly. To view a session when given a code;

1. From the browse screen pop-up when first launching the application, enter the code into the code box and hit enter.

OR

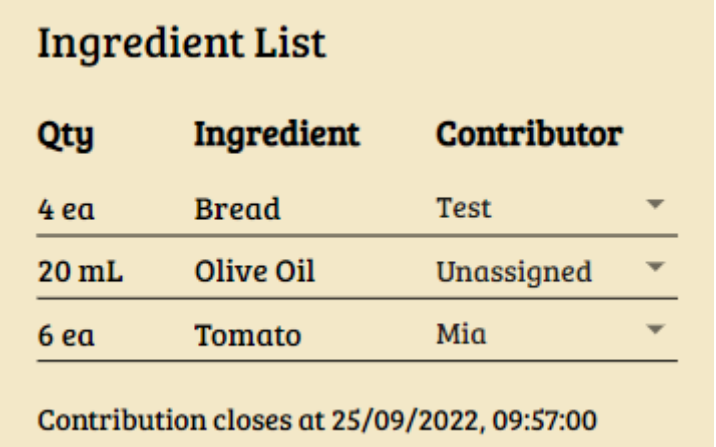
From the session screen, enter a code in the top text box and hit enter



2. Either of these options will display the session details page

Assign Ingredients

1. From the session details page (see above methods), scroll down to the ingredient area



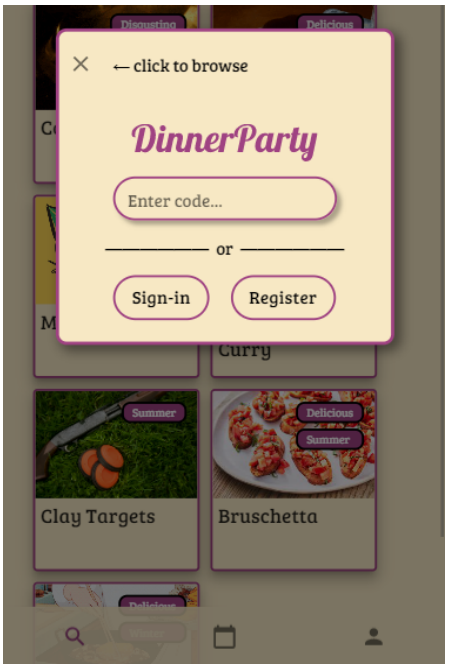
2. Select the drop down for the ingredient you wish to assign and select the name of the user you wish to assign to the ingredient.

Accounts

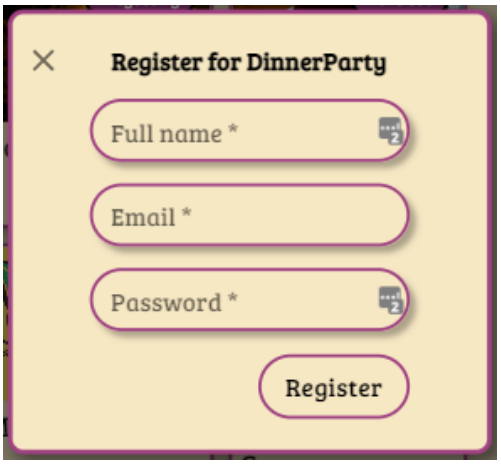
The following section outlines all the functionality available with the accounts within the application.

Register

1. Select Register from the initial pop up or from the user profile page. (Far right icon on the bottom nav bar).

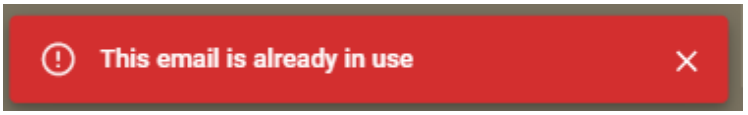


2. Enter the desired details, including name, email and password in the prompt. Select “Register” at the bottom.



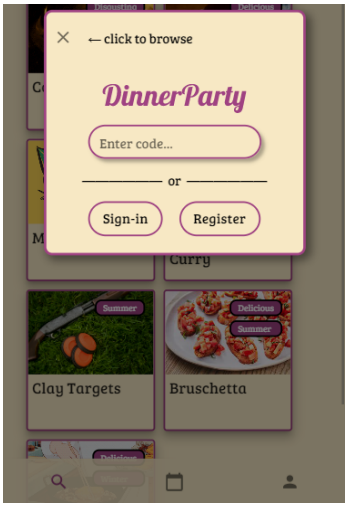
3. If the registration was successful, a welcome message will be displayed and the new user will be logged in.

4. If the registration was unsuccessful for some reason, the appropriate error message will be displayed and the form will remain present ready for the user to make adjustments.



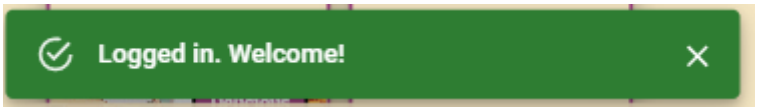
Sign-in

1. Select sign-in from the initial pop-up or from the user page. (Far right icon on the bottom nav bar).

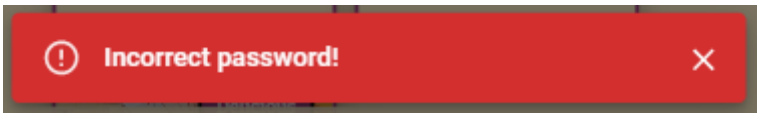


2. Enter the sign-in details in the provided pop-up. Select Sign-in when complete.

3. If successful, a welcome message will be displayed



4. If unsuccessful, an appropriate error message will be displayed and the user will be prompted to retry.



Logout

1. From a logged in user, select the profile page (far right) from the bottom nav bar.

2. Select the “Logout” button in the centre of the page

3. If successful, a success message will be displayed

References

Alex Reardon, *React Beautiful D&D*, accessed 1 Aug 2022, <https://www.npmjs.com/package/react-beautiful-dnd>.

Creative Commons 2016, *Creative Commons — Attribution 4.0 International — CC BY 4.0,* Creativecommons.org, accessed 5 August 2022, <https://creativecommons.org/licenses/by/4.0/>.

Django Software Foundation 2022, *Django documentation*, accessed 1 August 2022, <https://docs.djangoproject.com/en/4.0/intro/overview/>.

Django Software Foundation 2022, *Django overview*, accessed 1 August 2022, <https://www.djangoproject.com/start/overview/>.

Django Software Foundation 2022, *Django Trademark License Agreement*, accessed 1 August 2022, <https://www.djangoproject.com/trademarks/>.

Doableware AB 2022, *Djongo Get Started*, accessed 1 August 2022, <https://www.djongomapper.com/get-started/> .

Doableware AB 2022, *Djongo Mappe*r, accessed 1 August 2022, <https://www.djongomapper.com>.

Docker Inc., Docker Subscription Service Agreement, accessed 1 August 2022, <https://www.docker.com/legal/docker-software-end-user-license-agreement/>.

Docker Inc., Docs, accessed 1 August 2022, <https://docs.docker.com>.

Docker Inc., Home, accessed 1 August 2022, <https://www.docker.com>.

Figma, Inc., Getting Started, accessed 1 Aug 2022, <https://help.figma.com/hc/en-us/categories/360002051633-FigJam#Get-started>.

Figma, Inc., Learn and Level Up, accessed 1 Aug 2022, <https://www.figma.com/>.
Greg Linden et al, 2003 Amazon.com Recommendations Item-to-Item Collaborative Filtering.

License n.d., MUI Store, accessed 5 August 2022, <https://mui.com/store/license/#i-standard-license>.

Material UI SAS 2022, License, accessed 1 Aug 2022, <https://mui.com/store/license/>.

Material UI SAS 2022, Mat UI, accessed 1 Aug 2022, <https://mui.com>.

Material UI SAS 2022, Material UI - Overview, accessed 1 Aug 2022, <https://mui.com/material-ui/getting-started/overview/>.

MongoDB, Inc 2022, MongoDB Atlas, accessed 1 August 2022, <https://www.mongodb.com/atlas>.

MongoDB, Inc 2022, MongoDB Documentation, accessed 1 August 2022, <https://www.mongodb.com/docs/atlas>.

MongoDB, Inc 2022, MongoDB Licensing, accessed 1 August 2022, <https://www.mongodb.com/community/licensing>.

Meta Platforms, Inc. 2022, Getting Started, accessed 1 Aug 2022, <https://reactjs.org/docs/getting-started.html#learn-react>.

Meta Platforms, Inc. 2022, MIT License, accessed 1 Aug 2022, <https://github.com/facebook/react/blob/main/LICENSE>.

Meta Platforms, Inc. 2022, React, accessed 1 Aug 2022, <https://reactjs.org>.

Pessé, S & O'Mullan, A 2014, *Learn Javascript*, GitHub, accessed 5 August 2022, <https://github.com/GitbookIO/javascript/blob/master/LICENSE>.

Reardon, A 2019, *react-beautiful-dnd (rbd)*, GitHub, accessed 5 August 2022, <https://github.com/atlassian/react-beautiful-dnd/blob/master/LICENSE>.

The Python Software Foundation 2022, datetime - Basic date and time types, accessed 1 August 2022, <https://docs.python.org/3/library/datetime.html>.

The Python Software Foundation 2022, Email - An email and MIME handling package, accessed 1 August 2022, <https://docs.python.org/3/library/email.html>.

The Python Software Foundation 2022, History and License, accessed 1 August 2022, <https://docs.python.org/3/license.html>.

The Python Software Foundation 2022, Python 3 documentation, accessed 1 August 2022, <https://docs.python.org/3/>.

Yarn, BSD 2-Clause "Simplified" License, accessed 1 Aug 2022, <https://github.com/yarnpkg/berry/blob/master/LICENSE.md>.

Yarn, Introduction, accessed 1 Aug 2022, <https://yarnpkg.com/getting-started>.

Yarn, Yarn, accessed 1 Aug 2022, <https://yarnpkg.com>.

yarnpkg/yarn 2022, GitHub, accessed 5 August 2022, <https://github.com/yarnpkg/yarn/blob/master/LICENSE>.