Let's Cook!

Amanda Hoelting, Ellison Largent, Lily Logan, Will Marceau

Software Design Specification

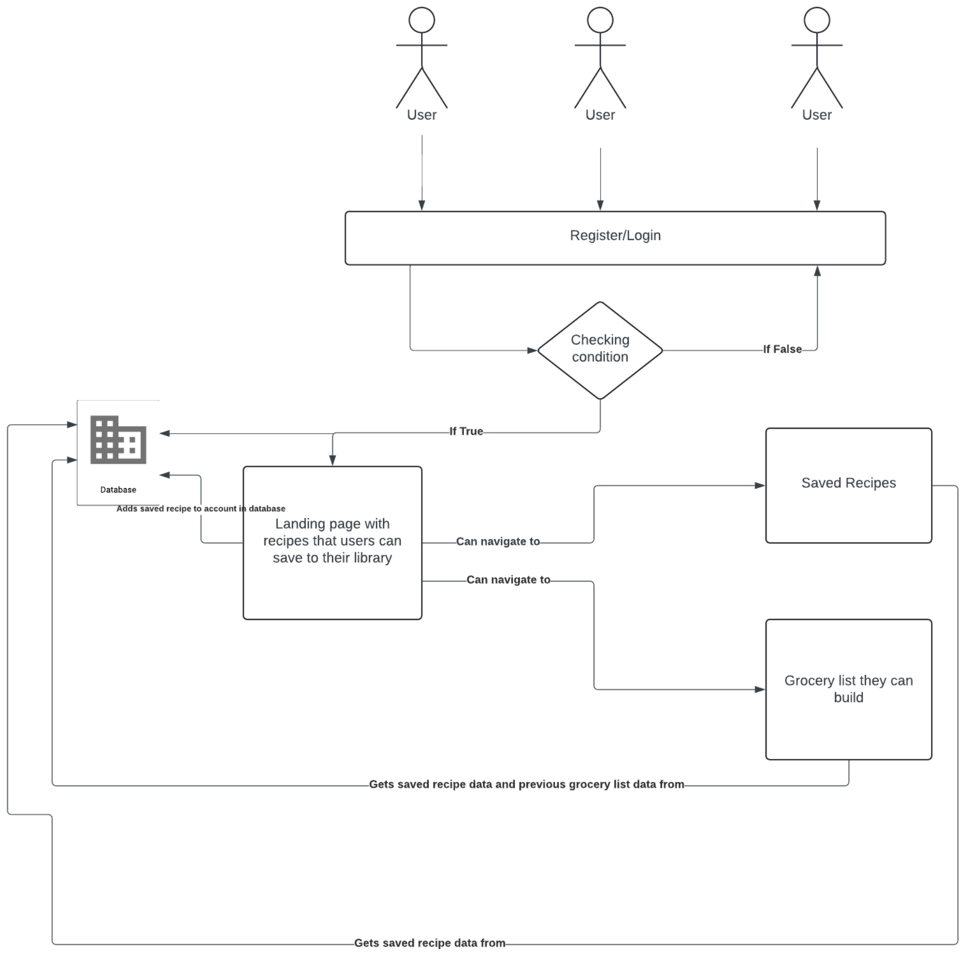
1. System Overview

1.1. Description

Choosing what meal to make is a common challenge, often leading to unnecessary takeout and overspending on food. *Let's Cook!* is a web application designed to simplify meal planning by offering a streamlined, intuitive solution to recipe selection. Rather than flipping through traditional cookbooks or endlessly scrolling through online recipes, users can respond with a simple "yes," "no," or "maybe" to suggested meals. This quick decision-making process not only saves time but also automatically tracks the ingredients needed for each selected meal. By swiping through options, users can easily curate a meal plan while concurrently building a personalized grocery list, all within one cohesive platform.

Let's Cook! consists of six main components: user account information, a login screen, a recipe landing page, saved recipes, meal plans, and grocery list. The user account and login screen manage profiles for individual users, storing essential data such as email, phone number, and name to maintain the user's active account. The recipe landing page is where users interact with meal suggestions, determining whether to choose, discard, or consider recipes. This decision-making process sends selected recipes to the user's saved recipes section and updates the meal plan with ingredients automatically added to the grocery list. These interconnected components work together to provide a seamless experience, with real-time updates between recipe decisions and meal planning, ensuring users can effortlessly manage their cooking and grocery needs.

1.2. Design Description


User
User
User
Register/Login
Checking condition
If False
If True
Database
Adds saved recipe to account in database
Landing page with recipes that users can save to their library
Can navigate to
Can navigate to
Saved Recipes
Grocery list they can build
Gets saved recipe data and previous grocery list data from
Gets saved recipe data from

List of components:

Login page - Provides a login page where users can register for an account or log in with their existing username and password.

Landing page - A page that will have one recipe on it and will allow users to swipe left if they do not want to save it to their saved recipes menu or swipe right if they do want to save it. Once they swipe, a new recipe will pop up afterwards.

Saved recipes - A page that will have a list of all recipes the user has saved. The user also has the option to create and add their recipe here. There will be search and filter options to allow the user to find exactly the recipe they are looking for.

Grocery list - A page that will allow users to add saved recipes to their weekly meal plan and then add their groceries to a list where they can check off the ingredients as they get them at the store.

2. Software Architecture

2.1. Software Architecture Components

2.1.1. Login

2.1.1.1. The login module allows user to gain access to the web app, maintaining their past data and preferences.

2.1.2. Sign up

2.1.2.1. The sign-up module facilitates the creation of a new user with access to the web app.

2.1.3. New recipe/discovery deck

2.1.3.1. The new recipe/discovery deck suggests new or unpreferenced recipe cards to the user which are interacted with and stored in the user's preferences within the user database.

2.1.4. Liked recipe deck/list

2.1.4.1. The liked recipe deck/list maintains the users' liked recipes in two different formats. The deck feature allows the user to swipe through each recipe while the list feature displays all recipes liked by the user.

2.1.5. Let's Cook home page

2.1.5.1. Let's Cook home page is the landing page for the user with navigation tools to other tabs of the web application.

2.1.6. Recipe meal plan/grocery list

2.1.6.1. Recipe meal plan/grocery list stores the user's chosen recipes for their meal plan and the grocery list is produced from the ingredients of each recipe in the meal plan.

2.1.7. Cookbook

2.1.7.1. The cookbook maintains all of the user's past and present likes and used recipes.

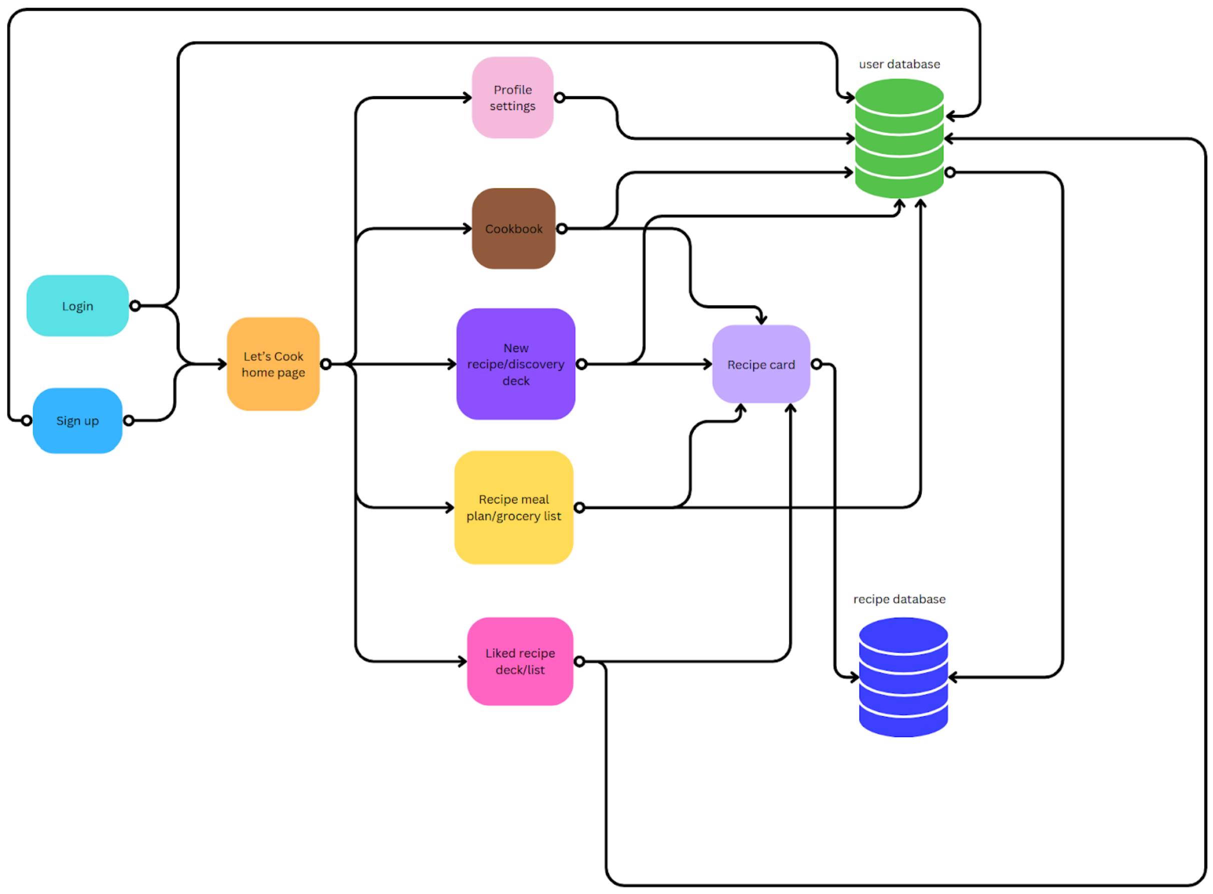
2.1.8. Profile settings

2.1.8.1. The profile settings store the user's basic information such as their username, email, and password.

2.1.9. Recipe card

2.1.9.1. The recipe card has two functions. It shows the user a minimized version of the recipe and it can expand to a full view of the recipe to its entirety.

2.2. Software Architecture Diagram



2.3. Software Architecture Interactions Between Modules

2.3.1. Login + user database

2.3.1.1. When the user logs in, the system must retrieve information about recipe preferences and other user data from the user database.

2.3.2. Login + Let's Cook home page

2.3.2.1. Once the user has logged in, the system connects the interface to the homepage.

2.3.3. Sign up + user database

2.3.3.1. When a user signs up with the application, a new entry is entered into the user database and ready to be changed according to the user's actions.

2.3.4. Sign up + Let's Cook home page

2.3.4.1. Once the user has signed up, the system connects the interface to the homepage.

2.3.5. Let's Cook home page + Profile settings

2.3.5.1. The homepage contains a navigation bar that can lead to the profile settings. This subsystem must display its contents.

2.3.6. Let's Cook home page + Cookbook

2.3.6.1. The homepage contains a navigation bar that can lead to the cookbook. This subsystem must request and retrieve the necessary data to then show the user.

2.3.7. Let's Cook home page + New recipe/discovery deck

2.3.7.1. The homepage contains a new recipe/discovery deck shown in the center of the user's screen.

2.3.8. Let's Cook home page + Recipe meal plan/grocery list

2.3.8.1. The homepage contains a navigation bar that can lead to the recipe meal plan/grocery list. This subsystem must request and retrieve the necessary data to display to the user within this tab.

2.3.9. Let's Cook home page + Liked recipe deck/list

2.3.9.1. The homepage contains a navigation bar that can lead to the recipe deck/list. This subsystem must request and retrieve the necessary data to display visually to the user like the deck in the main homepage.

2.3.10. Profile settings + user database

2.3.10.1. The profile settings must request the user's information from the user database. Once it's received, this module may continue with its UI actions.

2.3.11. Cookbook + user database

2.3.11.1. The cookbook is personalized to each user. To maintain the correct data for that user, the cookbook module must retrieve the user's saved and created recipes from the user database.

2.3.12. Cookbook + recipe card

2.3.12.1. The cookbook contains recipe cards within its subsystem to visualize each saved and created recipe for the user.

2.3.13. New recipe/discovery deck + user database

2.3.13.1. The new recipe/discovery deck is personalized to each user. To maintain the correct data, the deck must contain information about the user's recipe preferences from the user database.

2.3.14. New recipe/discovery deck + recipe card

2.3.14.1. The new recipe/discovery deck is a collection of recipe cards.

2.3.15. Recipe meal plan/grocery list + user database

2.3.15.1. The recipe meal plan/grocery list is personalized to each user; thus to maintain the correct user preferences, this module must retrieve the current user's data from the user database.

2.3.16. Recipe meal plan/grocery list + recipe card

2.3.16.1. The recipe meal plan will have a feature that expands recipes, displaying the recipe card.

2.3.17. Liked recipe deck/list + recipe card

2.3.17.1. The liked recipe deck/list contains recipe cards.

2.3.18. Liked recipe deck/list + user database

2.3.18.1. The liked recipe deck/list is personalized to each user. To maintain the correct user recipe preferences, this module must request and collect the current user's recipe preferences from the user database.

2.4. Rationale for Architectural Design: The system as a whole is large. There are nine modules that create the system; however, they are all interconnected in some way. For the system to be personalized to each user, the user database is a highly important and necessary subsystem that directly and indirectly connects with every module. In order to maintain a large recipe source and display important information within the recipe cards and grocery list, the recipe database must directly and indirectly connect with every module. Thus, the system is complex, it is necessary for each subsystem to work according to the web application needs.

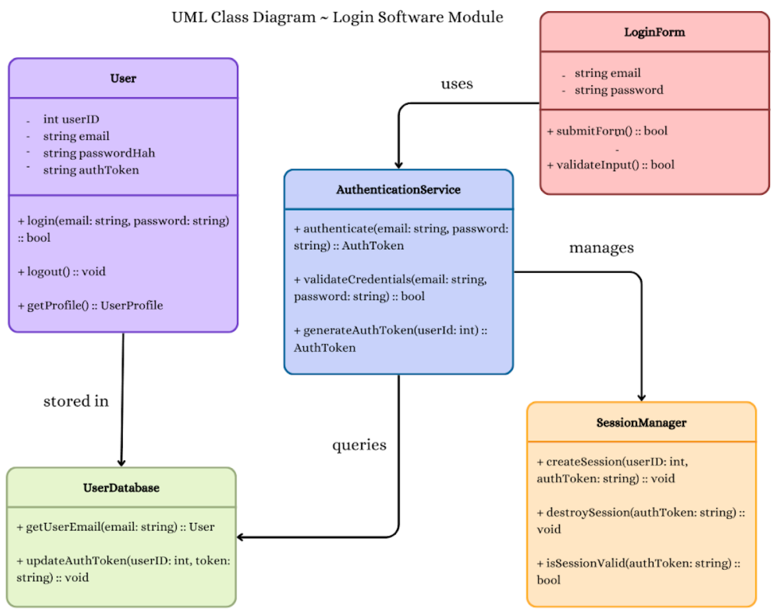
3. Software Modules

3.1. Login

3.1.1. Primary Role and Function: For the login software module, the primary purpose would be to allow users a method by which they can access the web app functions personalized to them allowing them to save, and discard recipes according to their interests. The login would require a user to input their email and password, which they have an account in the system to access the rest of the site. This would involve session management, user authentication, access control to personalized features, and protection of user data.

3.1.2. Interface Specification: The interface specifications for this can be split into two major components, user interface, and data passing. The user interface for this module would consist of a login form with fields for email and password, a login button, and a link to the registration page for new users to create an account. Data passing would include the input of username and password, and the output created would be an authorization token, user profile data, and error messages.

3.1.3. Static Model:



UML Class Diagram ~ Login Software Module
User
- int userID
- string email
- string passwordHah
- string authToken
+ login(email: string, password: string) :: bool
+ logout() :: void
+ getProfile() :: UserProfile
LoginForm
- string email
- string password
+ submitForm() :: bool
+ validateInput() :: bool
uses
AuthenticationService
+ authenticate(email: string, password: string) :: AuthToken
+ validateCredentials(email: string, password: string) :: bool
+ generateAuthToken(userId: int) :: AuthToken
manages
stored in
queries
UserDatabase
+ getUserEmail(email: string) :: User
+ updateAuthToken(userID: int, token: string) :: void
SessionManager
+ createSession(userID: int, authToken: string) :: void
+ destroySession(authToken: string) :: void
+ isSessionValid(authToken: string) :: bool

Figure 1: A UML Class Diagram modeling the connection between all the various classes such as user, user database, session manager, login form, and authentication service. Illustrates the relationships, methods, and attributes of each class.

3.1.4. Dynamic Model:

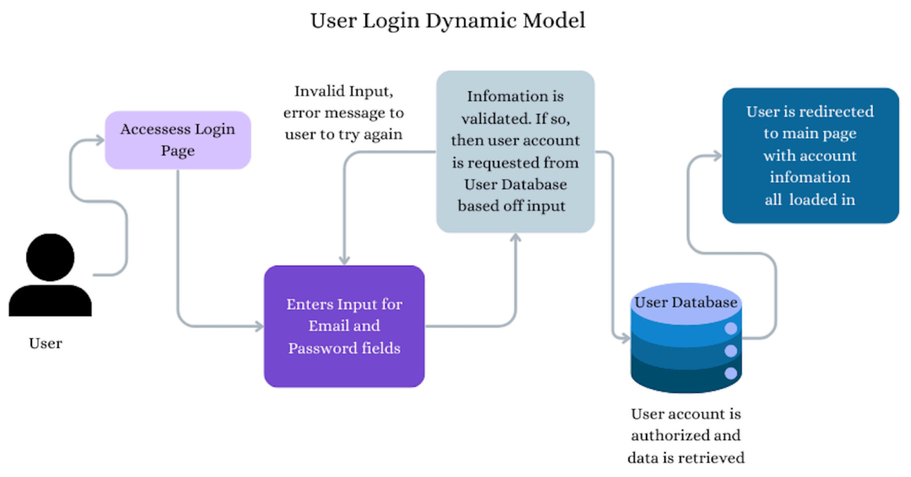

User Login Dynamic Model
User
Accessess Login Page
Enters Input for Email and Password fields
Invalid Input, error message to user to try again
Infomation is validated. If so, then user account is requested from User Database based off input
User Database
User account is authorized and data is retrieved
User is redirected to main page with account infomation all loaded in

Figure 2: A dynamic model diagram of the interaction between a user and the login module. From the accessing of the login page to the redirection to the main page, upon successful login.

3.1.5. Design Rational:

The login module was designed with a strong focus on security, modularity, and user experience. It separates the process into distinct classes for user interface (the login form), business logic (authentication service), data accesses (user database), and session management (session manager). This separation of parts allows for easier testing, maintenance, and the ability to add more or enhance the process in the future. The use of password hashing and authentication tokens aims to enhance security, while the modular structure allows for scalability and the potential of adding additional authentication methods in the future. The design also incorporates immediate feedback for users via input validation and error-handling messages or alerts.

3.1.6. Alternative Designs:

3.1.6.1. One possible alternative to the current design would be a stateless JSON Web Token-based authentication system, eliminating the need for server-side session storage. This could allow for easier implementation of microservices but would also increase complexity in token management and revocation.

3.1.6.2. Another possible design alternative would be a social media integration approach to authentication, allowing users to log in via existing accounts on platforms such as Google. This would simplify the user experience by reducing the need for new account creation and password management but would introduce dependencies on the third-party service and could even raise privacy concerns.

3.2. Sign Up

3.2.1. Primary Role and Function: The function of the sign-up module is to facilitate the creation of a new database entry for a new user to create an account. The sign up module will have new users input data, and create a new key for each user to be used for login and website access.

3.2.2. Interface Specification: The sign up module would require a unique email, a password, and a reconfirmation of the given password. This password will then be encrypted before being stored using Sha256, and the combination of the email and password will serve as a key for the user account in the database. The display page for this module will include a sign up form with inputs for email, password, and reconfirmation of password as well as a submit button, and a back button in case of a wrong entry. After creating an account by signing up with this module, users in the future will be able to use the login module to access their account and personalized website.

3.2.3. Static Mode

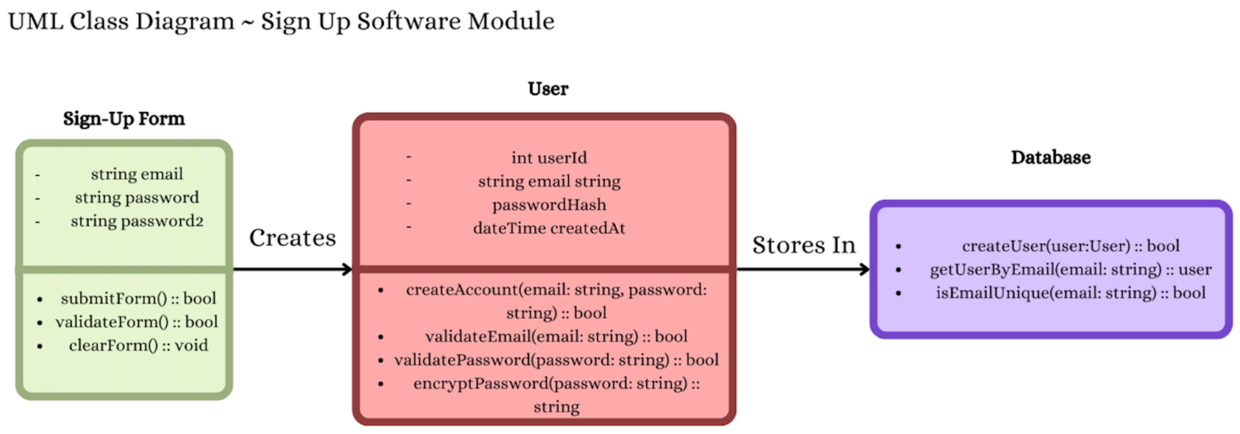

UML Class Diagram ~ Sign Up Software Module
Sign-Up Form
- string email
- string password
- string password2
• submitForm() :: bool
• validateForm() :: bool
• clearForm() :: void
Creates
User
- int userId
- string email string
- passwordHash
- dateTime createdAt
• createAccount(email: string, password: string) :: bool
• validateEmail(email: string) :: bool
• validatePassword(password: string) :: bool
• encryptPassword(password: string) :: string
Stores In
Database
• createUser(user:User) :: bool
• getUserByEmail(email: string) :: user
• isEmailUnique(email: string) :: bool

Figure 3: A UML Class diagram demonstrating the attributes and methods behind each class in the three-tier architecture as well as the relationships between the classes.

3.2.4. Dynamic Model:

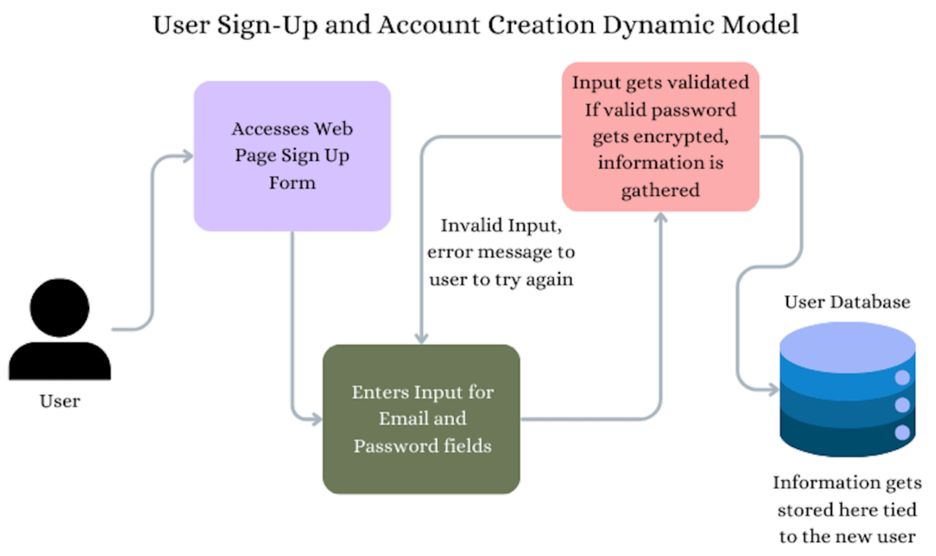

User Sign-Up and Account Creation Dynamic Model
User
Accesses Web Page Sign Up Form
Enters Input for Email and Password fields
Input gets validated If valid password gets encrypted, information is gathered
Invalid Input, error message to user to try again
User Database
Information gets stored here tied to the new user

Figure 4: A dynamic model of user interaction with the sign-up module, from the first access of the web page to account creation.

3.2.5. Design Rationale: The design for this module focuses on security, usability, and modularity. It employs a three-tier architecture, separating the UI of the Sign-Up Form, the business logic of the User, and data persistence with the Database. This separation of concerns enhances maintainability and allows for independent scaling and modification of each component. The module prioritizes security by implementing email uniqueness checks and password encryption using SHA256 before storage. User experience is improved through immediate form validation and clear error messaging. Overall, this design strikes a balance between robust security measures, intuitive user interaction, and flexible system architecture, making it well-suited for modern web applications requiring secure and efficient user registration processes.

3.2.6. Alternative Designs:

3.2.6.1. One alternative design our team has considered is implementing a Single Page Application (SPA) Design. In which we would use a JavaScript framework like React or Vue.js for the front-end. Handle sign-up logic in the browser, and send the data to a simple API backend. Then we would store the user data in a database. This would cause less server load and a faster user experience but require a more complex front end.

3.2.6.2. Another alternative design we have considered is to use a server-side framework like Express.js or Django. With that, we could handle all sign-up logic on the server. Then after that, we would render HTML pages and send them to the browser. This would make for a simpler front-end, but cause more server load and slower page updates.

3.3. New Recipe / Discovery Deck

3.3.1. Primary Role and Function: The primary role of the recipe deck module is to suggest recipe cards to the user that they either have not seen before, have used but not liked, or have decided that they might want to try at a later time. It then allows the user to interact with this card giving them 3 swiping options, a left swipe to dislike the recipe, a downward swipe to get a new recipe, and a right swipe to add the recipe to their meal plan.

3.3.2. Interface Specification: This module will enable the user to interact with the new recipe deck. It will get new recipes from the database to display to the user. When the user swipes a direction, the module will give this information to the other modules that need it like the database and meal planner, depending on the direction of the swipe.

3.3.3. Static Model:

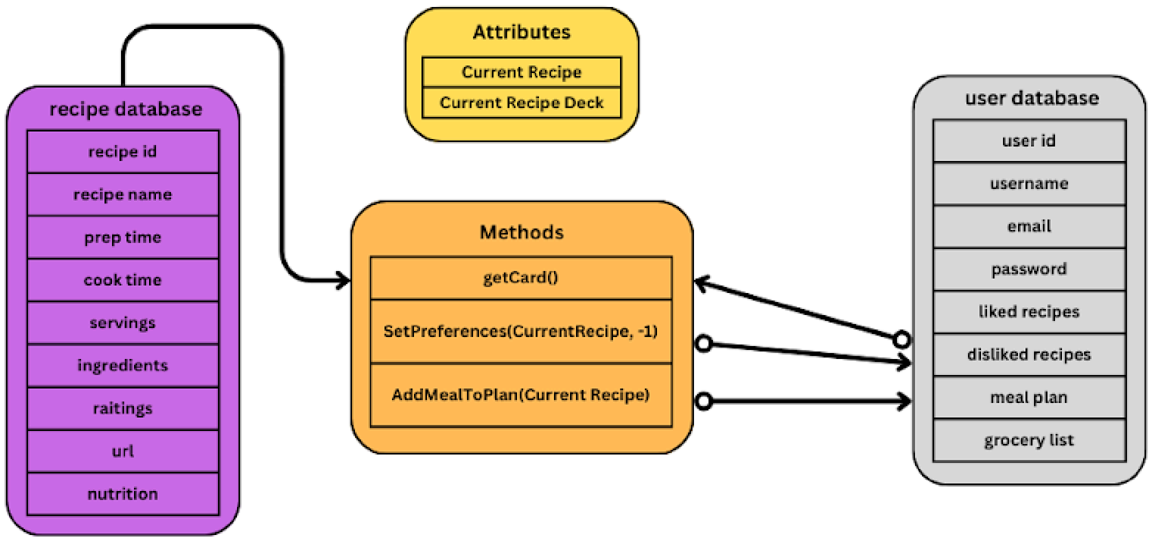
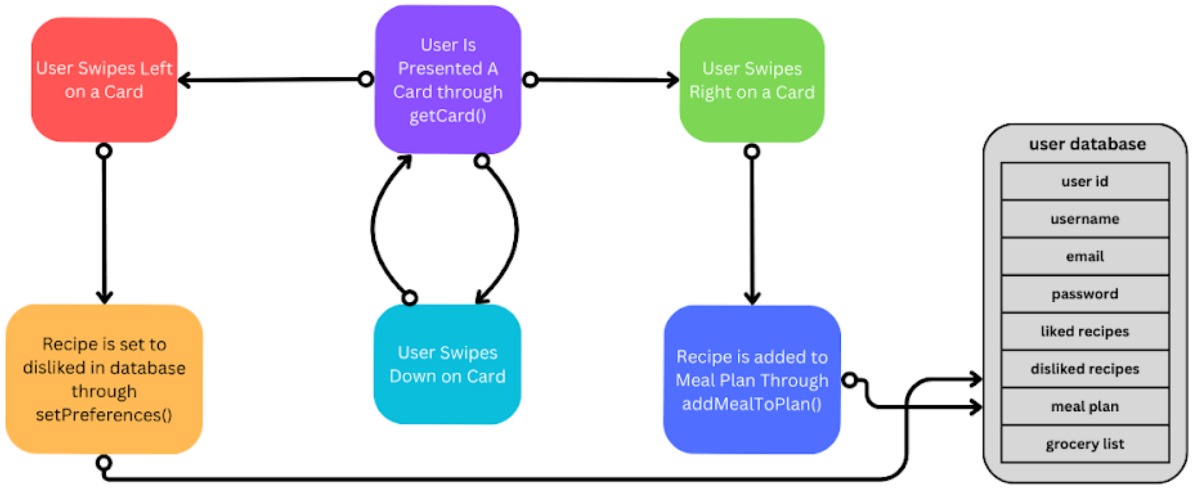


Figure 5: This is a chart of the methods and attributes contained in the Discovery Deck module. It also shows their relationship with our databases when they query or update the information stored in them.

3.3.4. Dynamic Model:



User Swipes Left on a Card
User Is Presented A Card through getCard()
User Swipes Right on a Card
Recipe is set to disliked in database through setPreferences()
User Swipes Down on Card
Recipe is added to Meal Plan Through addMealToPlan()
user database
user id
username
email
password
liked recipes
disliked recipes
meal plan
grocery list

Figure 6: This is a flow chart of how the various methods of the Liked Recipe Deck / List module interact with each other and the user database based on user input.

3.3.5. Design Rationale: The current design of the discovery deck allows us to cover all possible use cases without making the controls too complex. The user can easily add a new meal they want to try to the meal planner with just a right swipe. If a user does not like a recipe they just swipe left and it will not be suggested again, and if the user just wants to see a new recipe without getting rid of one or adding it to their meal plan, they just swipe down. Finally, we decided to have a unique button for liking a recipe to allow the user to add a recipe to their meal plan without liking it, so they can try it out before adding it to their list. This simple control scheme makes it very easy to interact with our app and for the user to find new recipes they might enjoy.

3.3.6. Alternative Designs:

3.3.6.1. One alternative design we considered is having buttons instead of swiping but decided that swiping would not only look better but be more conducive to both our computer and phone audiences.

3.3.6.2. Our first iteration of what each swipe does was a little different. Originally we only had two swipes, left for dislike and right for like. We then added the later swipe to make it so that you can get a new recipe without having to like or dislike the current recipe. We then changed the right swipe functionality to add to the meal planner and decided to move the like functionality to the recipe card to make it so the user can use a recipe without liking it first

3.3.6.3. Finally, we split the deck into the liked recipes deck and discovery deck to better accommodate our use cases. If you want to find new recipes then you can use the discovery deck, but if you just want help deciding what to eat for the week you can go through your liked recipes deck.

3.4. Liked Recipe Deck / List

3.4.1. Primary Role and Function: The primary role of the liked recipe deck is to suggest recipe cards from the users' liked list for the user to interact with. It allows the user to interact with these cards in two ways, a left swipe to get a new card from their liked recipe deck, and a right swipe to add to their meal plan for the week. Additionally, this module will display the liked recipes in list format for users on the liked recipes page of the software.

3.4.2. Interface Specification: This module will enable the user to interact with the liked recipe deck/list. It gets new recipes from the users' liked recipes from the database when needed. It receives input from the recipe cards to add or remove recipes to this list/deck when the user specifies. It then passes the meals that the user wants to add to the meal planner module.

3.4.3. Static Model:

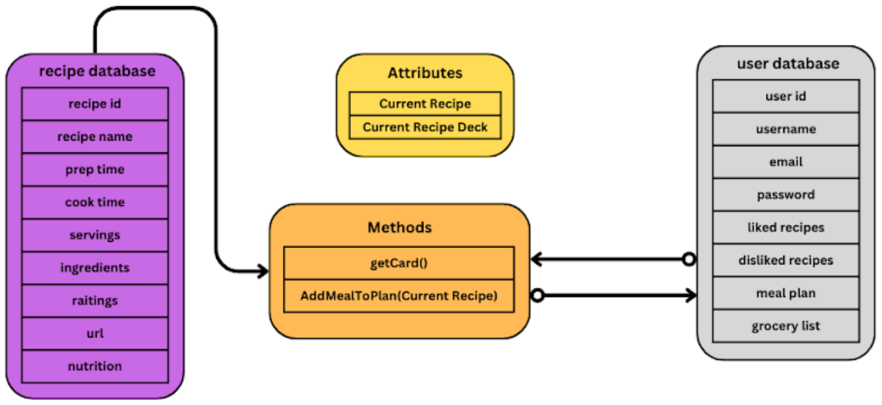


Figure 7: This is a chart of the methods and attributes contained in the Liked Recipe Deck / List module. It also shows their relationship with our databases, when they query or update the information stored in them.

3.4.4. Dynamic Model:

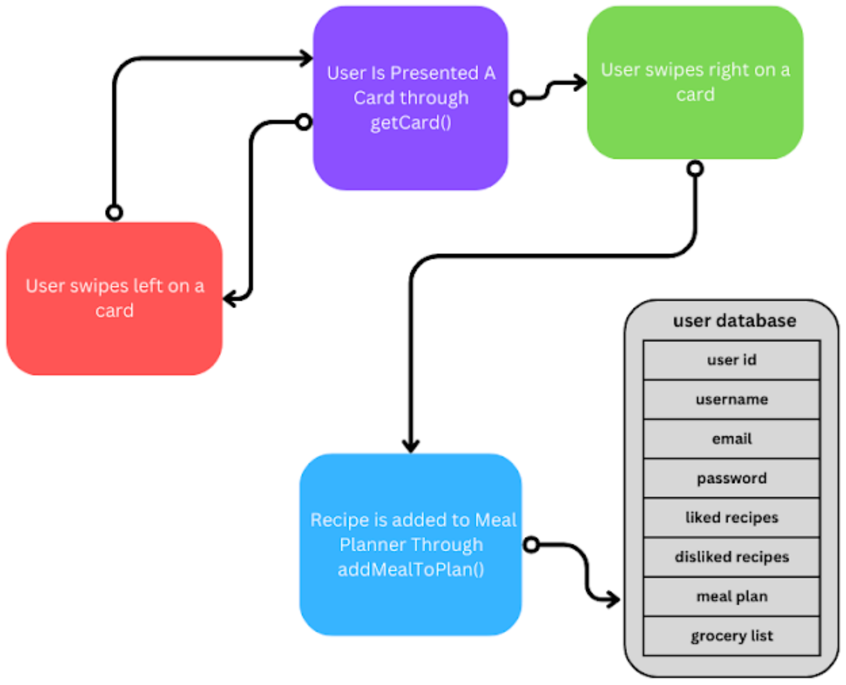

User Is Presented A Card through getCard()
User swipes right on a card
User swipes left on a card
Recipe is added to Meal Planner Through addMealToPlan()
user database
user id
username
email
password
liked recipes
disliked recipes
meal plan
grocery list

Figure 8: This is a flow chart of how the various methods of the Liked Recipe Deck / List module interact with each other based on user input. It also shows when the user database gets updated and what section of the database the method writes to.

3.4.5. Design Rationale: We designed the users' liked recipes in a deck format and list format to accommodate all the use cases we could think of for this portion of the app. One of the many reasons we wanted to make the app was to not only make an easy way for people to decide what to eat for the week but, to also help remove the need for long recipe books. The liked recipe deck was our solution to this as it gives you quick suggestions of recipes you like, with an easy way to add them to your meal plan for the week. However, with this implementation, we realized that it would be an issue if someone had a specific recipe in mind, and had to flip through the entire deck to find it. To solve this we decided to also have a liked recipes page with a search bar, to get that traditional recipe book feel but also make it easier to find the recipe you are looking for

3.4.6. Alternative Designs

3.4.6.1. We originally only had the discovery deck but decided to create this one as well to accommodate the people who just wanted to decide what they wanted to eat from meals they have already cooked and know they like

3.4.6.2. We then decided to change the swiping mechanics to better match the goals of this deck, changing the left swipe to pass, removing the down swipe, and keeping the right swipe as an add.

3.5. Let's Cook Home Page

3.5.1. Primary Role and Function: The primary role of the Let's Cook home page is not only to be a landing page for when someone logs into the app and allow for easy navigation between the other areas of the app but to display the recipe cards from the selected recipe deck to allow the user to play "recipe Tinder", which is the main draw of our app.

3.5.2. Static Model:

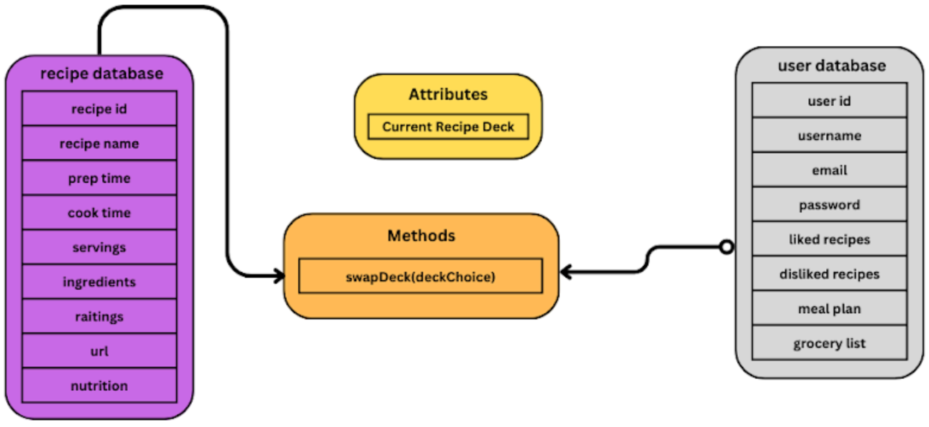


Figure 9: This is a chart of the methods and attributes contained in the Let's Cook Home Page Module. It also shows their relationship with our databases, when they query or update the information stored in them.

3.5.3. Dynamic Model:

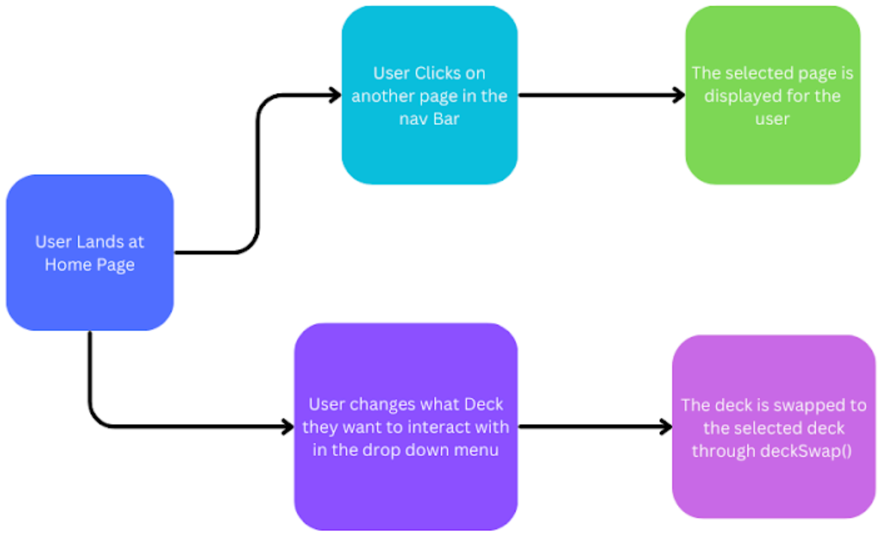

User Lands at Home Page
User Clicks on another page in the nav Bar
The selected page is displayed for the user
User changes what Deck they want to interact with in the drop down menu
The deck is swapped to the selected deck through deckSwap()

Figure 10: This is a flow chart of how a user would interact with the Let's Cook home page module.

3.5.4. Design Rationale: We decided to design our home page like this to highlight the most important part of our app. The "recipe Tinder" section is critical to all the other functions of our app and is what separates us from our competitors. Therefore, we wanted this to be front and center as soon as you enter the site. Additionally, easy navigation between sections of the app is critical to all web applications, as you will lose users if it is too convoluted or difficult.

3.5.5. Alternative Designs:

3.5.5.1. The design for the home page has not changed that much, but the first design only displayed cards from the discovery page as at that point we only had one deck

3.5.5.2. After deciding to have more than one deck, we chose to include a dropdown menu over the displayed recipe card for the user to select which deck they want to use

3.6. Recipe Meal Plan / Grocery List

3.6.1. Primary Role and Function: The primary role of the Recipe Meal Plan and Grocery List is to allow users to view their weekly meal plan and see the grocery checklist the site has generated for them.

3.6.2. Interface Specification: This module will call on the user database to retrieve the user's meal plan and recipes to display what they have planned to make for the week and will generate a checklist of their groceries. If they delete a recipe from their weekly meal plan, this will make a change to both their meal plan and grocery list in the user database.

3.6.3. Static Model:

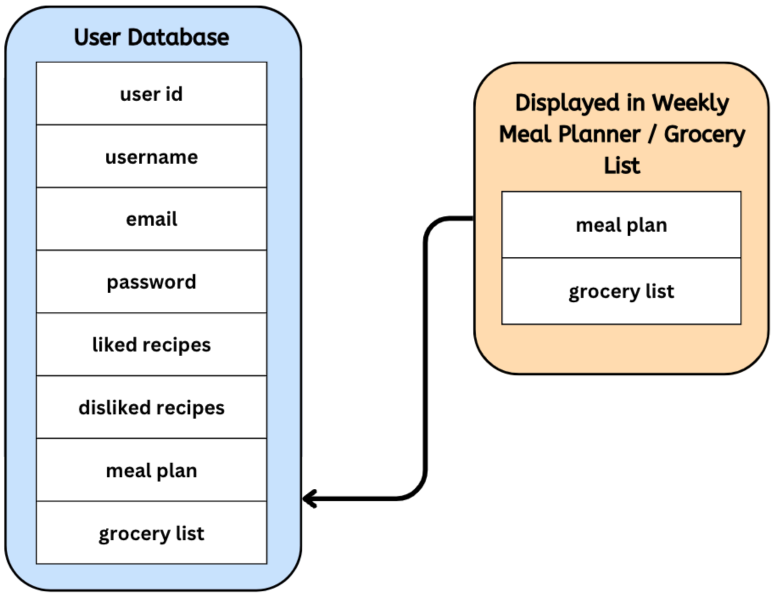


Figure 11: A chart modeling the connection between the information displayed in the weekly meal planner/grocery list and the user database.

3.6.4. Dynamic Model:

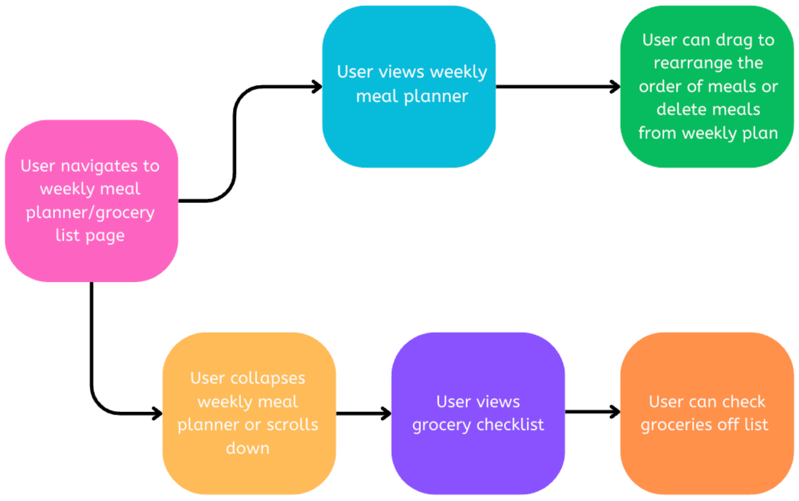

User navigates to weekly meal planner/grocery list page
User views weekly meal planner
User can drag to rearrange the order of meals or delete meals from weekly plan
User collapses weekly meal planner or scrolls down
User views grocery checklist
User can check groceries off list

Figure 12: A chart modeling the user flow from first navigating to the weekly meal planner/grocery list page.

3.6.5. Design Rationale: This page is for the user to view and organize the recipes that they have saved to their weekly meal plan. They can rearrange the order of their recipes for the week and delete recipes they no longer want in their weekly meal plan. The user can also scroll down or minimize the weekly meal plan panel to view the grocery list, which is generated from the ingredients needed for the recipes in their weekly meal plan. They can then check off items if they already have them or if they have bought them.

3.6.6. Alternative Designs:

3.6.6.1. Originally, we were not sure we wanted to put the weekly meal planner and the grocery list on the same page. However, we decided to include them both on the same page since the weekly meal planner directly influences what will be added to the grocery list. It will be easier for the user to delete a recipe from their weekly meal planner and then be able to view how that affected their grocery list.

3.7. Cookbook

3.7.1. Primary Role and Function: The primary role of the cookbook is to let users view all of the recipes they have liked and add their own personal recipes to the list. There will be a search bar that allows users to search for recipes by name.

3.7.2. Interface Specification: This module will call on the user database to display liked recipes in a list format. Users can also add their recipes in this tab. After adding the necessary information (instructions, ingredients, etc), it will be added to their liked recipes in the user database.

3.7.3. Static Model:

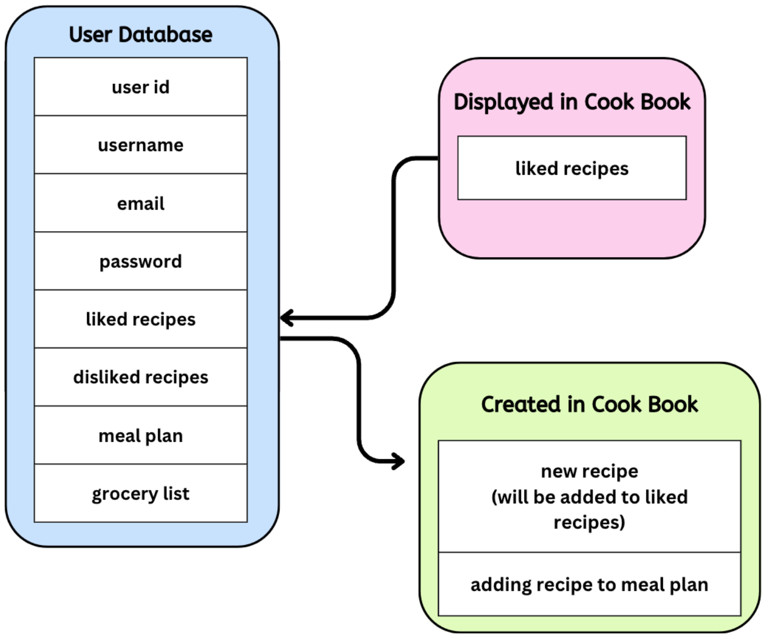


Figure 13: A chart modeling the passing of information back and forth between the user database and the cookbook page.

3.7.4. Dynamic Model:

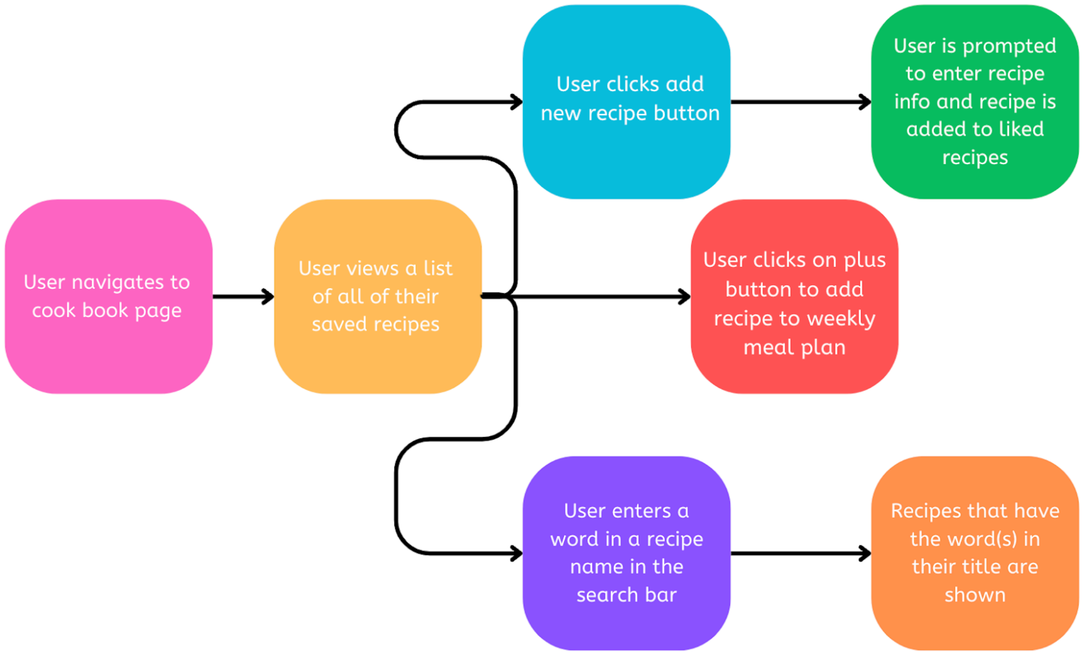

User navigates to cook book page
User views a list of all of their saved recipes
User clicks add new recipe button
User is prompted to enter recipe info and recipe is added to liked recipes
User clicks on plus button to add recipe to weekly meal plan
User enters a word in a recipe name in the search bar
Recipes that have the word(s) in their title are shown

Figure 14: A chart modeling the user flow from first navigating to the cookbook page.

3.7.5. Design Rationale: We wanted to make a list of all of their liked recipes in case they wanted to repeat a recipe they had made a few weeks prior. We also wanted to give users the option of adding their recipes or one that they found not on our platform that they can add to their weekly meal plan.

3.7.6. Alternative Designs:

3.7.6.1. We were originally planning to have this as the only way users could build their weekly meal plan, but we decided to give them the swiping option on the homepage so they didn't have to track down recipes on a large list.

3.8. Profile Settings

3.8.1. Primary Role and Function: The primary role of the profile settings is to display the specific characteristics associated with each user. This contains their username, password, and email. This will help store their personal recipe preferences and their user ID.

3.8.2. Interface Specification: The combination of username, password, and email ensures each entry in the user database is unique, creating a clean environment with minimal bugs when retrieving and updating user data.

3.8.3. Static Model

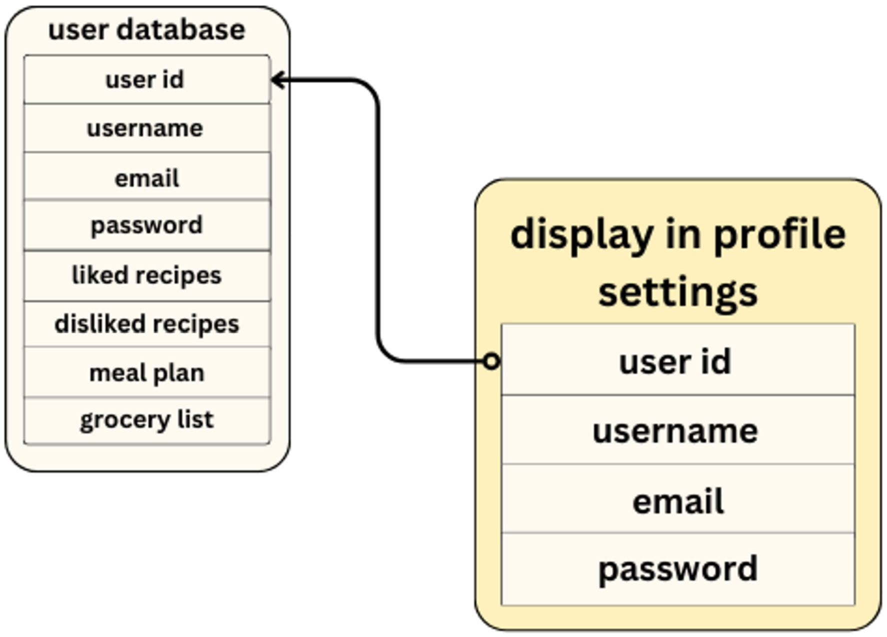


Figure 15: The profile settings within the web application interface display parts of the user's information held in the user database. The user database contains many more details allowing for other features of this application to run.

3.8.4. Dynamic Model

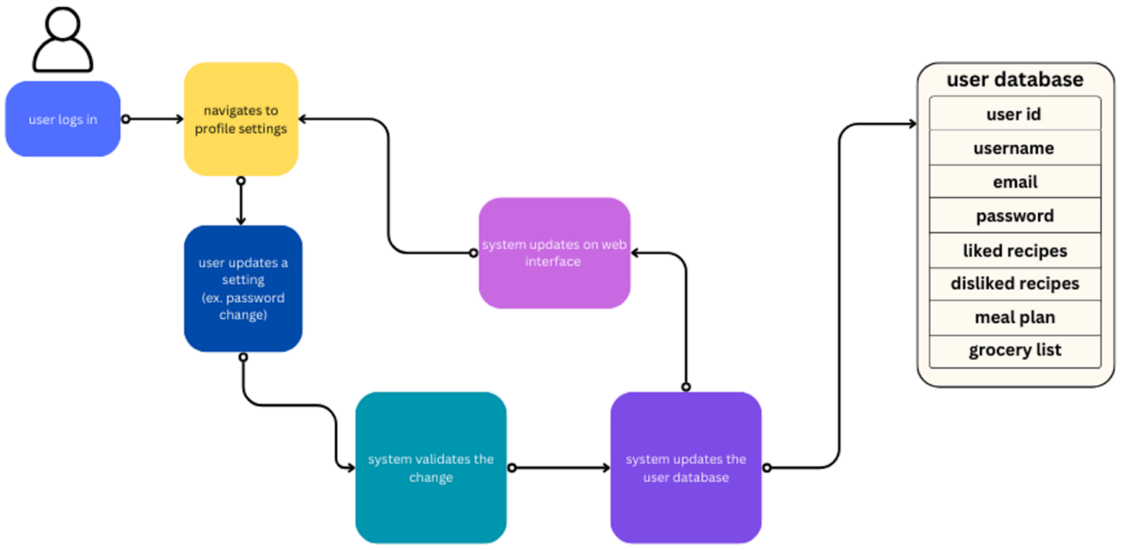

user logs in
navigates to profile settings
user updates a setting (ex. password change)
system validates the change
system updates the user database
system updates on web interface
user database
user id
username
email
password
liked recipes
disliked recipes
meal plan
grocery list

Figure 16: Whenever the user changes information held in the profile settings, the system must connect with the user database and update these changes.

3.8.5. Design Rationale: With this design, we can keep user information without displaying all data stored in this profile settings module. In the profile settings, only parts of the users' data will be visible. However, other areas from the user database will be able to be seen by the user in other modules. Such as meal plans and grocery lists in their respective tab. A user database allows us to keep track of each user and their preferences.

3.8.6. Alternative Designs: An alternative design may include separate databases, one containing the profile settings and the other containing their recipe preferences. However, this would potentially complicate the system and introduce potential bugs, making it less efficient and harder to maintain.

3.9. Recipe Card

3.9.1. Primary Role and Function: The role of a recipe card is to provide the user with essential information about a recipe. This includes but is not limited to prep time, cooking instructions, and ingredients. This card may be expanded to full view which contains all recipe attributes. This is activated when the user clicks on the "more info" button.

3.9.2. Interface Specification: The recipe cards present the information contained in the backend recipe database in a user-friendly manner. It connects directly to the backend, and maintains the stored recipes for this web application, enabling the full view option for the user.

3.9.3. Static Model

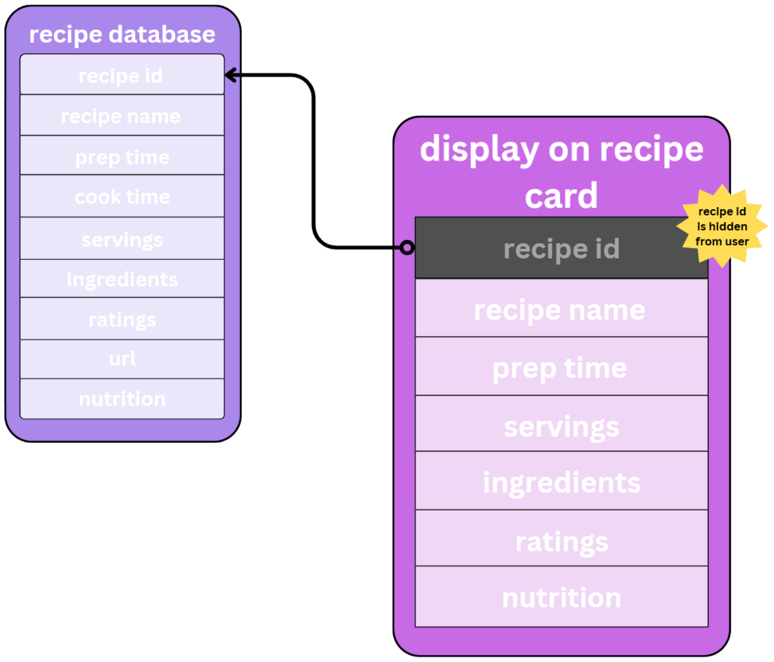


Figure 17: The recipe card holds all of the information stored in the recipe database in a user-friendly design. The only attribute that is hidden from the user is the recipe id which allows for easy retrieval from the database.

3.9.4. Dynamic Model

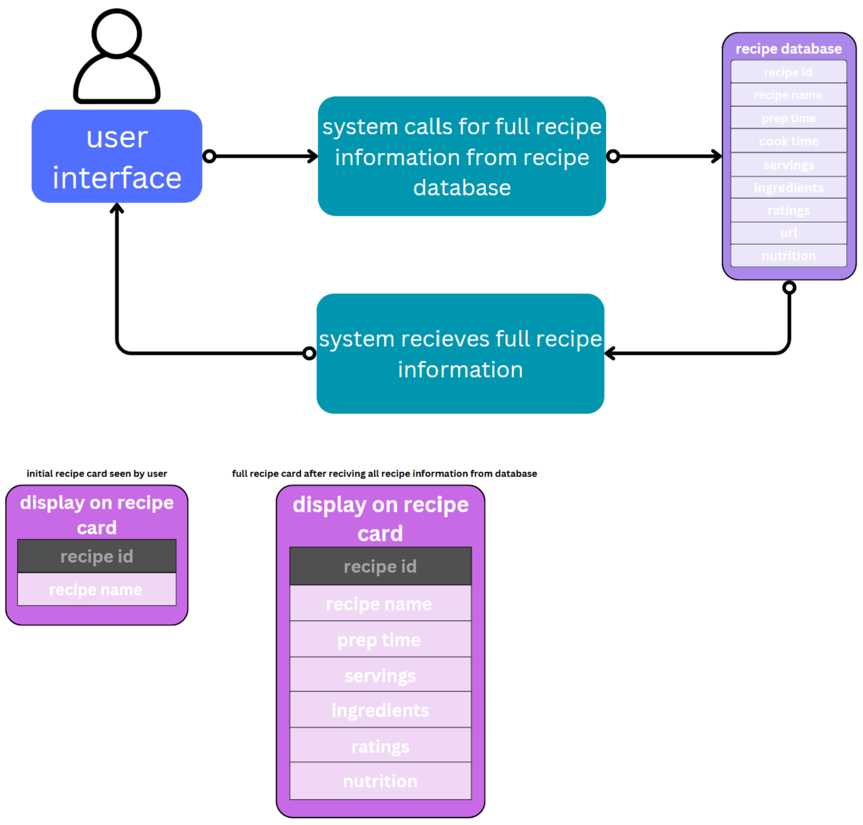


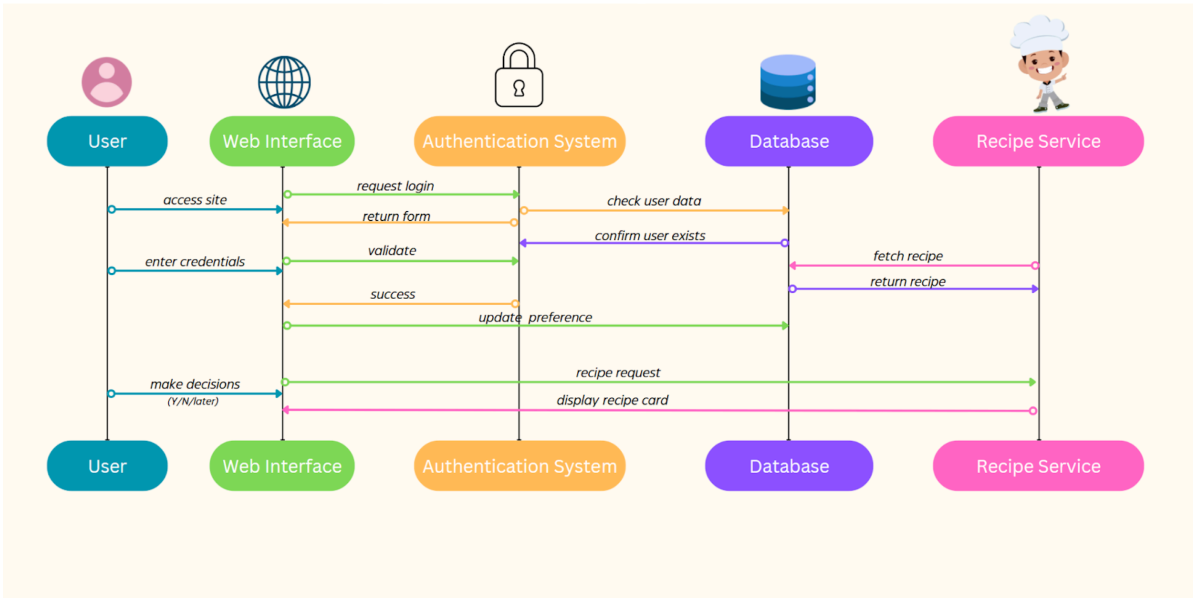
Figure 18: When the user expands the recipe in any way, all attributes, except the recipe id, will be available to view on the recipe card. The user must click the "more information" button which then signals the system to retrieve the full recipe information.

3.9.5. Design Rationale: This design helps reduce the possibility of overwhelming a user by initially presenting a concise view of a recipe. The user may choose to expand the recipe for more details, creating an intuitive and simple user experience.

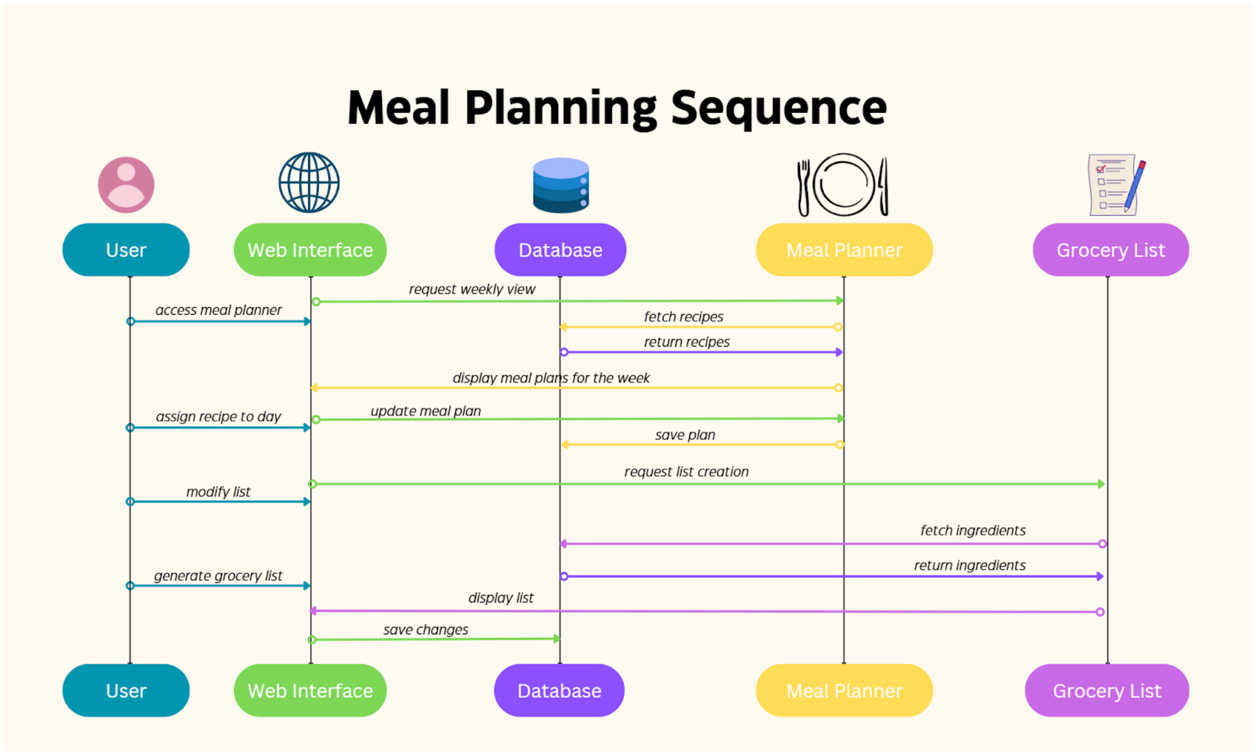
3.9.6. Alternative Designs: Displaying the entire recipe card which contains all the recipe attributes from the stored recipe database is an alternative design, but is not as user-friendly as the approach our web application provides.

4. Dynamic Models of Operational Scenarios (Use Cases)

4.1. Recipe Discovery Flow

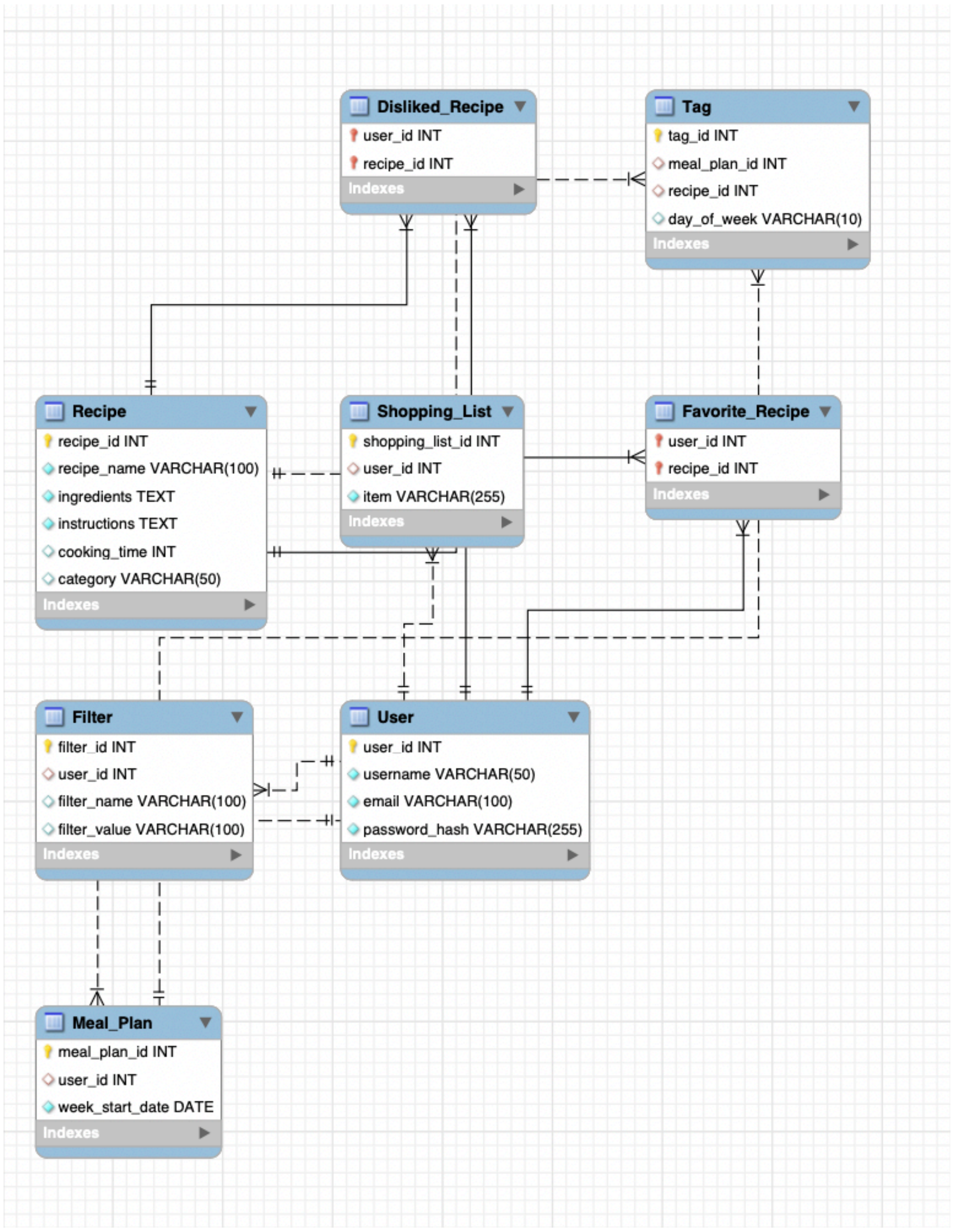


4.2. Meal Planning Sequence



5. Design Underlying Database

5.1. ER Diagram of SQL Database References



5.2. Design Rationale

The design of the "Let's Cook" app database works to simplify the meal planning process by organizing user interactions with recipes more efficiently and effectively. The model centers around users and their ability to interact with the recipes, marking them as favorites or dislikes, tagging them for specific days of the week, and adding ingredients to their shopping lists. This many-to-many relationship between users and recipes is efficiently handled through junction tables (e.g., Favorite_Recipe and Disliked_Recipe), allowing each user to have a personalized experience while minimizing data redundancy.

Additionally, the inclusion of meal planning and shopping list tables provides a practical mechanism for users to organize their weekly meals and streamline grocery shopping. Each user can create weekly meal plans, and within these plans, they can tag recipes to specific days, aligning to make decision-making easier. The schema design enforces relationships through foreign keys and cascading deletes, ensuring data integrity by automatically removing associated records if a user is deleted. By leveraging this relational model, the app maintains flexibility, scalability, and clarity, allowing possible stretch goal expansion in the future with features like custom recipe creation and advanced filtering without requiring major structural changes.

6. References

The Devastator, Recipes Dataset. Recipes Dataset (kaggle.com)

7. Acknowledgments