**CS/SE/CE 3354 Software Engineering**

**Project Deliverable #2**

**DishDoc: Recipe Manager**

## Github Repository

- https://github.com/tteeoo/3354-DishDoc

## Delegated Tasks by Member

*Deliverable #1*

Aryan KC: I will make the first GitHub commit, then address the proposal feedback and work on the explanation for why we chose our software process model.

Evan Biggins: I will create sequence diagrams for the use cases.

Amrita Thapa: I will create the use case diagrams and class diagrams.

Shivank Singh: I will create the sequence diagrams.

Aima Salman: I will define the functional and non-functional software requirements

Riley Johnson: I will define the product scope and choose the software process model we use

Theodore Henson: I will create the GitHub repository, invite everyone, and create the architectural design diagram

*Deliverable #2*

Aryan KC: I'll do the final report comparison with similar designs, the conclusion and my individual presentation slides, and the overall flow and cohesion of the report.

Evan Biggins: I will contribute to the final report and presentation slides, particularly the Conclusions and Reflections sections

Amrita Thapa: I will work on the final report, presentation slides, project scheduling, and the conclusion and reflections.

Shivank Singh: I will work on my part of the final report and presentation slides.

Aima Salman: I will work on the final report, presentation slides, and the pricing estimation.

Riley Johnson: I will be working on implementing the code and test case.

Theodore Henson: I will work on the test plan and implement the tests, collaborating with Riley. I will also work on the final report, focusing on the aspects of the project to which I contributed, and create the slides for the presentation on these topics.

**Proposed Implementation**

We will be implementing a recipe manager website. Users will be able to create an account or log in in order to create, view, edit, or delete their recipes. Every recipe will have a list of ingredients along with the instructions. Additionally, recipes can be searched for by name, ingredients, or various tags assigned to each recipe (e.g., breakfast, lunch, dinner, quick, instant pot, etc.). Users will also be able to create a list of recipes and generate a shopping list from their ingredients, summarizing the ingredients used in multiple recipes. Finally, users will be able to export and import recipes from other users.

**Motivation**

We chose this project because it seemed like a realistic project that we could not only plan, but also create. At its core, a recipe manager is a create, read, update, and delete (CRUD) project; however, more features can be added to make it the right level of challenge. Also, as everyone cooks and nobody has time to write out shopping lists anymore, a recipe manager is a must-have.

**Project Proposal Feedback**

The positive feedback we received on our initial proposal confirmed that our project was viable and realistic; therefore, we are proceeding with the original structure. We are confident that our scope will enable us to successfully plan the architecture and potentially deliver a fully functional, high-quality minimum viable product (MVP).

**Software Process Model**

For our software process model, we're choosing to use the iterative process model. We like this in our software development lifecycle because it will let us deliver the core Client-Server data, like CRUD functions, in the first iteration - effectively allowing us to address the most significant risk in our architecture (reliable, synchronized, multi-user data). This iterative model will enable us to address more complex features after the first iteration has solidified the core features of the app. The iterative approach will also enable us to break the project into smaller cycles, improving manageability and reducing the cost of changing requirements later in the development timeline. Finally, functional increments ensure a solid schedule for delivering requirements and will allow us to demonstrate measurable progress after each iteration.

We also considered other models, with the most thought given to the prototyping model, which would also be a suitable model for our project. However, it lacks discipline and structure, whereas the iterative approach enables us to outline a more effective schedule for delivering requirements. As to why we didn't choose the other models, the waterfall and spiral are both models that are more geared towards larger projects, and as such, add overhead that can slow down development and increase costs without much added benefit for our project.

**Software Requirements**

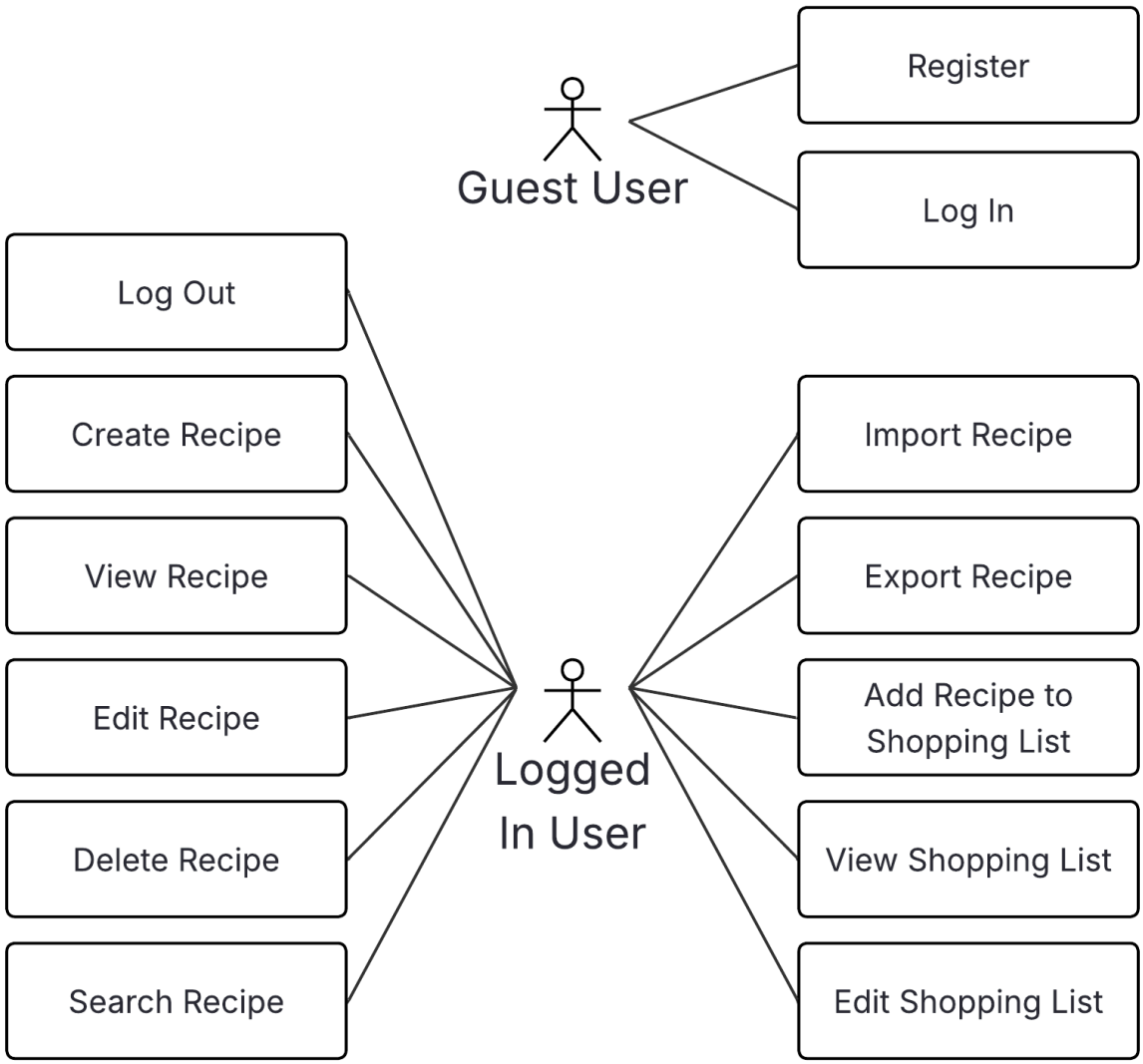
*Functional Requirements*

- The system shall enable users to create an account.
- The system shall enable users to log in securely.
- The system shall enable users to log out securely using unique credentials.
- The system shall allow authenticated users to create their recipes.
- The system shall allow authenticated users to view their recipes.
- The system shall allow authenticated users to edit their recipes.
- The system shall allow authenticated users to delete their recipes.
- The system shall enable users to create tags for categorizing their recipes.
- The system shall enable users to manage tags for categorizing their recipes.
- The system shall allow users to search for recipes by different criteria (e.g., “breakfast,” “lunch,” “dinner,” “chicken).
- The system shall allow users to automatically add a recipe’s ingredients to their shopping list.
- The system shall automatically sum ingredients from multiple recipes in the shopping list.

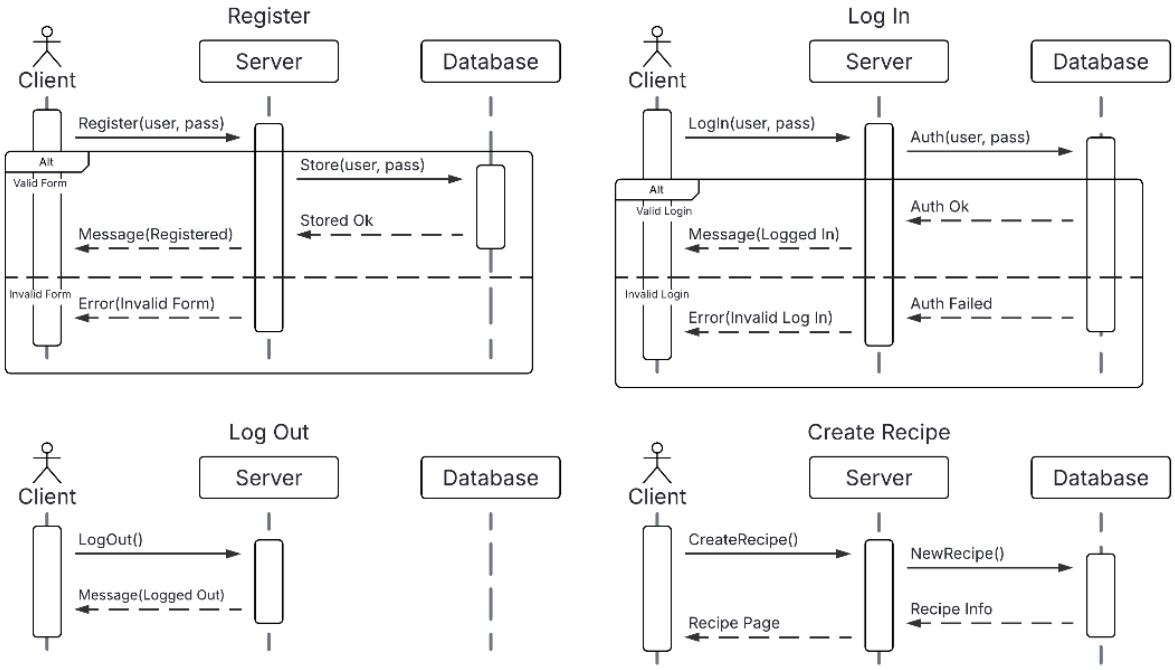
*Non-Functional Requirements*

- Usability Requirement: The system shall provide an interface that allows users to complete any actions within five clicks from the homepage.
- Performance Requirement: The system shall return recipe search results within 2 seconds.
- Space Requirement: The system shall have a storage capacity of less than 1TB.
- Dependability Requirement: The system shall be available 95% of the time.
- Security Requirement: The system shall protect against common web vulnerabilities.
- Environmental Requirements: The system shall be compatible with major operating systems.
- Operational Requirement: The system shall operate across all major browsers.
- Developmental Requirement: The system shall be developed using a modern web stack (e.g., React, Node.js/Express).
- Regulatory Requirement: The system shall comply with data protection and privacy laws.
- Ethical Requirement: The system shall ensure that harmful, discriminatory, or illegal material is not shared.
- Accounting Requirement: The system shall record and report metrics such as the number of active users, created recipes, and search queries for administrative review.
- Safety/Security Requirement: The system shall protect user data and alert when user data is breached, allowing users to be contacted

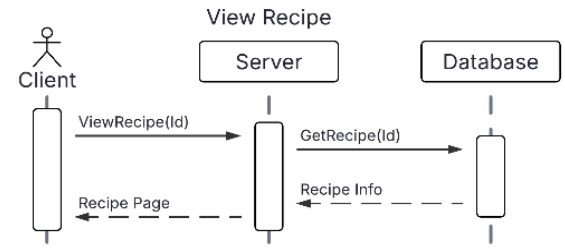
**Use Case Diagram**

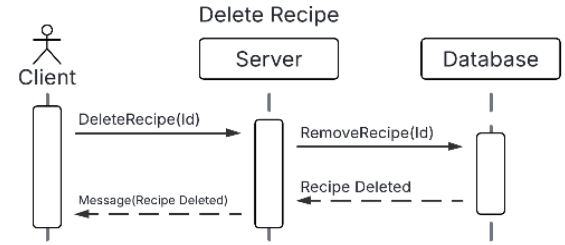


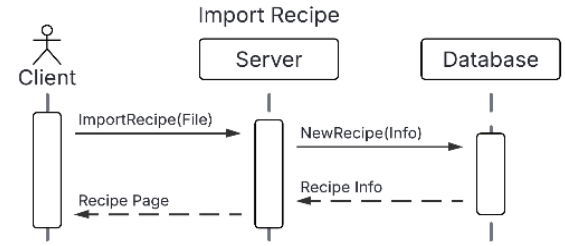
## Sequence Diagrams

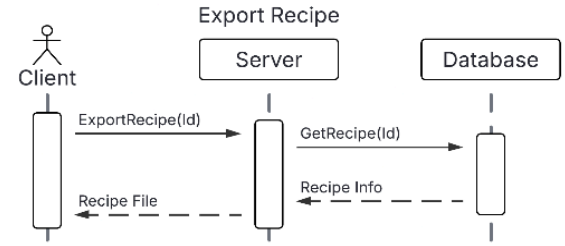


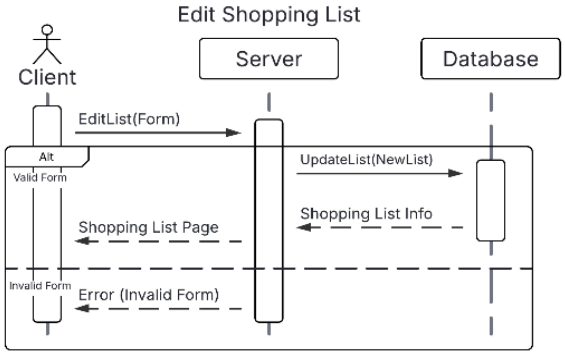

(More below)

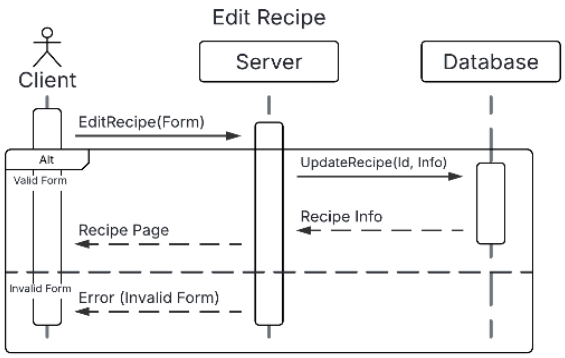

View Recipe
Client
Server
Database
ViewRecipe(Id)
GetRecipe(Id)
Recipe Info
Recipe Page

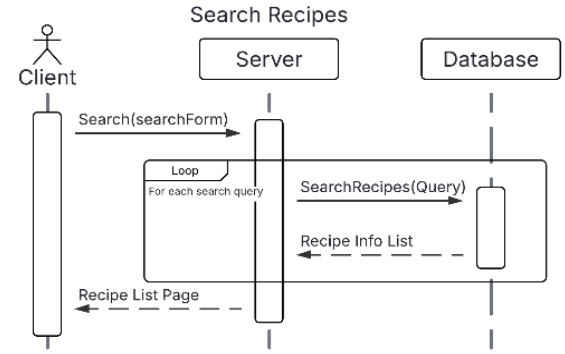

Delete Recipe
Client
Server
Database
DeleteRecipe(Id)
RemoveRecipe(Id)
Recipe Deleted
Message(Recipe Deleted)

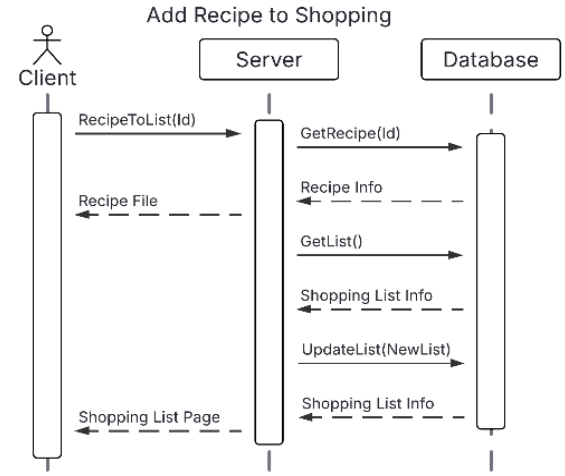

Import Recipe
Client
Server
Database
ImportRecipe(File)
NewRecipe(Info)
Recipe Info
Recipe Page

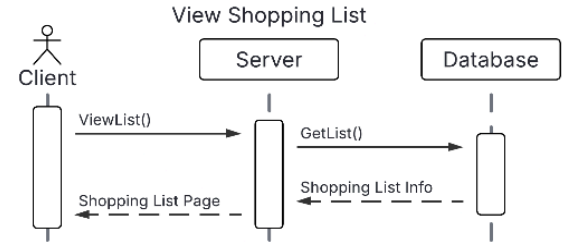

Export Recipe
Client
Server
Database
ExportRecipe(Id)
GetRecipe(Id)
Recipe Info
Recipe File


Edit Shopping List
Client
Server
Database
EditList(Form)
Alt
Valid Form
UpdateList(NewList)
Shopping List Info
Shopping List Page
Invalid Form
Error (Invalid Form)

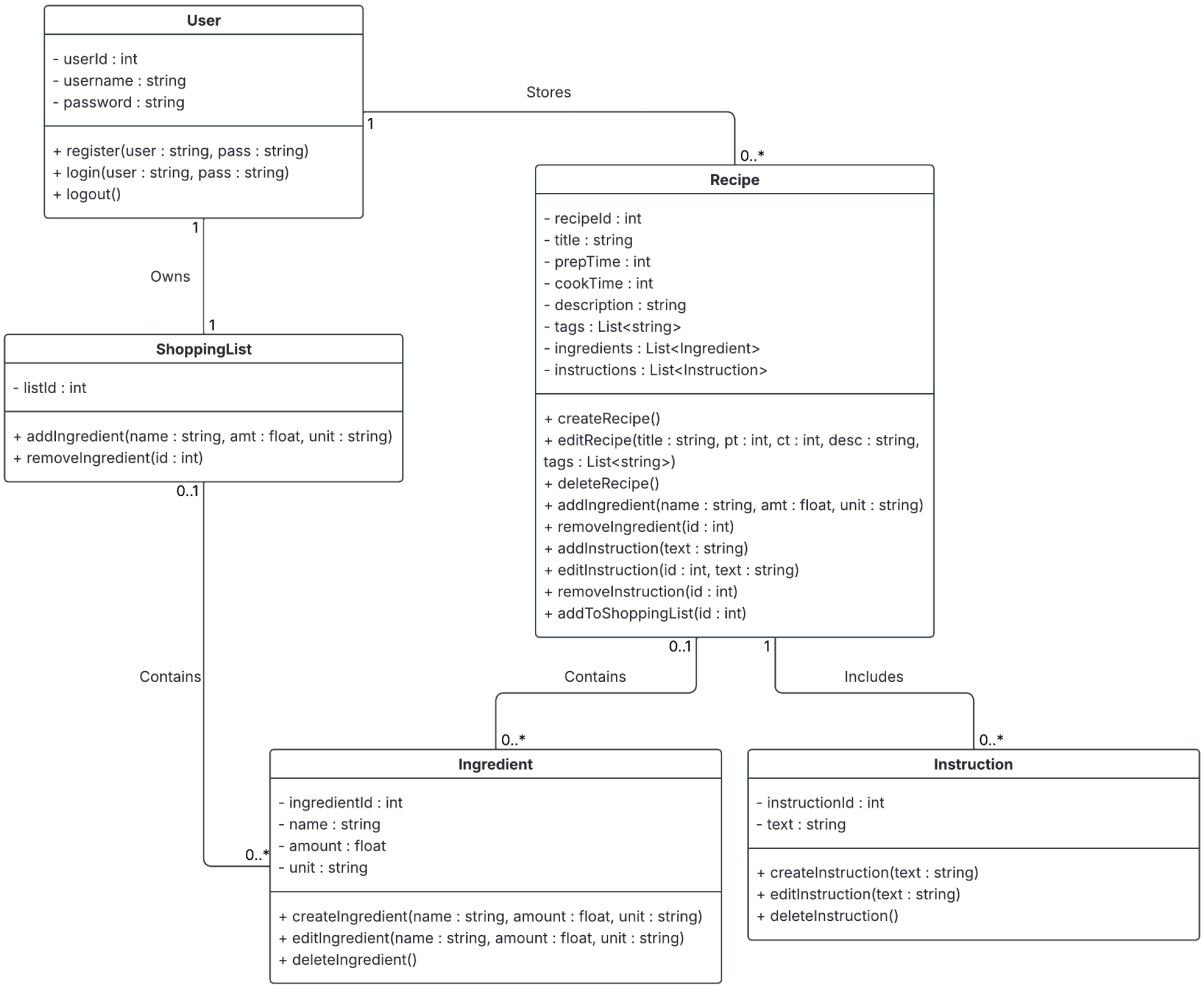

Edit Recipe
Client
Server
Database
EditRecipe(Form)
Alt
Valid Form
UpdateRecipe(Id, Info)
Recipe Info
Recipe Page
Invalid Form
Error (Invalid Form)


Search Recipes
Client
Server
Database
Search(searchForm)
Loop
For each search query
SearchRecipes(Query)
Recipe Info List
Recipe List Page

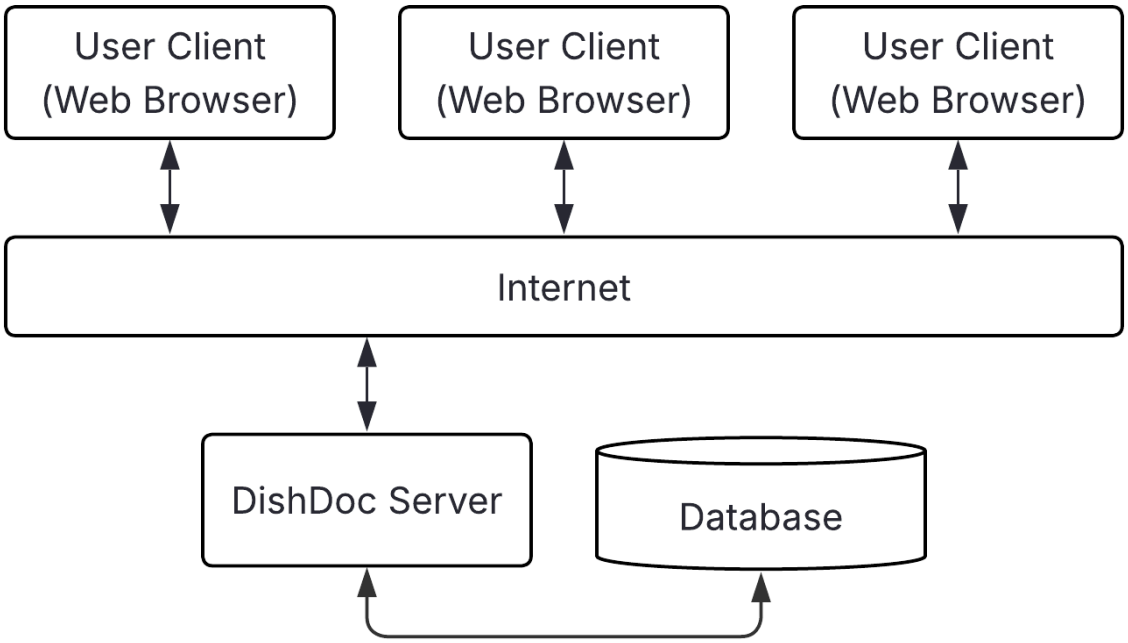

Add Recipe to Shopping
Client
Server
Database
RecipeToList(Id)
GetRecipe(Id)
Recipe Info
Recipe File
GetList()
Shopping List Info
UpdateList(NewList)
Shopping List Info
Shopping List Page


View Shopping List
Client
Server
Database
ViewList()
GetList()
Shopping List Info
Shopping List Page

# Class Diagram



**Architectural Design**



**Project Scheduling**

Given a team of three developers, we expect this project to take approximately three weeks, from December 1st to December 19th, with developers working eight hours per day, excluding weekends. Throughout this period, we will employ an iterative design model to develop our software.

*Iterable 1: Deliver a Functional Recipe Manager App (1.5-2 Weeks)*

- Front end with pages for user information, all recipes, one recipe, and a shopping list
- Database for storing user information, recipes, ingredients, and shopping lists
- Ability to perform CRUD actions on user information, recipes, and shopping lists from the front end

*Iterable 2: Linking Recipes to Shopping List (0.5-1 Week)*

- Able to automatically add ingredients from a recipe to the shopping list
- Shopping list gets simplified so that repeated ingredients are merged with their amounts converted into the same units and combined

*Iterable 3: Better Organization (0.5 Weeks)*

- Recipes can now be tagged using a variety of preset or custom tags
- Recipes can be searched with multiple queries, such as name, tag, ingredients, and cooking time
- Previously viewed or commonly viewed recipes are remembered and displayed earlier in the list

*Estimated Total: 2.5-3.5 Weeks*

**Cost, Effort, and Pricing Estimation**

We will be using the Function Point (FP) method for cost estimation. We believe that this system works best for our system because it's a web system that has well-defined user functions (login, recipe management, search, etc.). The benefit of using this method is that we can easily count inputs, outputs, and files from the requirements. The cost estimation is outlined below.

*Parameter Settings*

- Team size = 3 developers
- Efficiency = 15 fp/person-week

*Assumptions*

- User load is moderate, not enterprise-level.
- Infrastructure is cloud-hosted but small-scale.

*Function Point Estimation Table*

| | **Category** | **Count** | **Complexity** | | | **Count * Complexity** |
|---|---|---|---|---|---|---|
| | | | **Simple** | **Average** | **Complex** | |
| 1 | Number of user inputs | 12 | **3** | 4 | 6 | 36 |
| 2 | Number of user outputs | 4 | 4 | **5** | 7 | 20 |
| 3 | Number of user queries | 3 | **3** | 4 | 6 | 9 |
| 4 | Number of data files and relational tables | 5 | 7 | **10** | 15 | 50 |
| 5 | Number of external interfaces | 1 | **5** | 7 | 10 | 5 |
| | | | | | **GFP** | 120 |

*Processing Complexity*

| PC | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Score | 4 | 4 | 2 | 2 | 1 | 4 | 3 | 4 | 3 | 2 | 2 | 3 | 1 | 5 |

*Calculations*

PCA = 0.65 + 0.01(4+4+2+2+1+4+3+4+3+2+2+3+1+5) = 1.05
FP = GFP * PCA = 120 * 1.05 = 126
E = PCA / Efficiency = 126 / 5 = 8.4 person-weeks
D = E / Team size = 8.4 / 3 = 2.8 weeks

*Estimated cost of hardware products*

Laptops: $1,000/dev = $3,000
Total: $3,000

*Estimated cost of software products*

IDE: Free
Claude Pro Subscription: $20/dev = $60
Tech Stack (React, Node.js, Express): Free
Database: $60/month = $720/year
Cloud Hosting: $20/month = $240/year
Domain: $20/year
Total: $60 + $980/year

*Estimated cost of personnel*

Developer: $40/hr = $12,600
Total: ~$12,600

*Total and yearly cost*

Total: $3,000 + $60 + $12,600 = $15,660 + $980/year

**Test Plan**

Testing throughout the iterative design process is important to verify that individual components as well as the entire system works during and at the end of each cycle. To do this, we will implement automated testing frameworks to verify intended functionality while avoiding unintended side effects. As our back end will be built in JavaScript, we will use the Jest testing framework to handle these automated tests. Below is an example of test cases for a function that simplifies the shopping list by combining and converting repeated ingredients.

*Test 1 - Passed*
Input: [{ingredient: 'salt', amount: 1, unit: 'tsp'}]
Expected: [{ingredient: 'salt', amount: 1.0, unit: 'tsp'}]

*Test 2 - Passed*
Input: [{ingredient: 'sugar', amount: 1, unit: 'tsp'}, {ingredient: 'sugar', amount: 3, unit: 'tsp'}]
Expected: [{ingredient: 'sugar', amount: 1.3, unit: 'tbsp'}]

*Test 3 - Passed*
Input: [{ingredient: 'flour', amount: 1, unit: 'cup'}, {ingredient: 'flour', amount: 6, unit: 'tbsp'}]
Expected: [{ingredient: 'flour', amount: 1.4, unit: 'cup'}]

*Test 4 - Passed*
Input: [{ingredient: 'milk', amount: '1 1/2', unit: 'cup'}, {ingredient: 'milk', amount: '1/2', unit: 'cup'}]
Expected: [{ingredient: 'milk', amount: 2.0, unit: 'cup'}]

*Test 5 - Passed*
Input: [ {ingredient: 'pepper', amount: 1, unit: 'tsp'}, {ingredient: 'pepper', amount: '1/2', unit: 'tbsp'} ]
Expected: [{ingredient: 'pepper', amount: 2.5, unit: 'tsp'}]

*Test 6 - Passed*
Input: [ {ingredient: 'salt', amount: 1, unit: 'tsp'}, {ingredient: 'pepper', amount: 2, unit: 'tbsp'}, {ingredient: 'oregano', amount: 3, unit: 'tsp'} ]
Expected: [ {ingredient: 'salt', amount: 1.0, unit: 'tsp'}, {ingredient: 'pepper', amount: 2.0, unit: 'tbsp'}, {ingredient: 'oregano', amount: 1.0, unit: 'tbsp'} ]

*Test 7 - Passed*
Input: [ {ingredient: 'vanilla', amount: 0, unit: 'tsp'}, {ingredient: 'vanilla', amount: '0', unit: 'tsp'} ]
Expected: [{ingredient: 'vanilla', amount: 0.0, unit: 'tsp'}]

*Test 8 - Passed*
Input: [ {ingredient: 'a', amount: 1, unit: 'cup'}, {ingredient: 'a', amount: 0, unit: 'tsp'} ]
Expected: [{ingredient: 'a', amount: 1.0, unit: 'cup'}]

*Test 9 - Passed*
Input: [ {ingredient: 'ginger', amount: 1, unit: 'tsp'}, {ingredient: 'ginger', amount: 1, unit: 'tsp'}, {ingredient: 'ginger', amount: 1, unit: 'tsp'} ]
Expected: [{ingredient: 'ginger', amount: 1.0, unit: 'tbsp'}]

*Test 10 - Passed*

Input: [ {ingredient: 'honey', amount: '3/4', unit: 'tbsp'}, {ingredient: 'honey', amount: '1/4', unit: 'tbsp'} ]
Expected: [{ingredient: 'honey', amount: 1.0, unit: 'tbsp'}]

*Test 11 - Passed*
Input: [ {ingredient: 'Sugar', amount: 1, unit: 'tsp'}, {ingredient: 'sugar', amount: 2, unit: 'tsp'}, {ingredient: 'SUGAR', amount: 3, unit: 'tsp'} ]
Expected: [{ingredient: 'sugar', amount: 2.0, unit: 'tbsp'}]

*Test 12 - Passed*
Input: [ {ingredient: 'a', amount: 1, unit: 'tsp'}, {ingredient: 'b', amount: '1/2', unit: 'tbsp'}, {ingredient: 'c', amount: '3/4', unit: 'tsp'}, {ingredient: 'a', amount: 2, unit: 'tbsp'}, {ingredient: 'd', amount: 1, unit: 'cup'}, {ingredient: 'b', amount: '3/4', unit: 'tbsp'} ]
Expected: [ {ingredient: 'a', amount: 2.3, unit: 'tbsp'}, {ingredient: 'b', amount: 1.3, unit: 'tbsp'}, {ingredient: 'c', amount: 0.8, unit: 'tsp'}, {ingredient: 'd', amount: 1.0, unit: 'cup'} ]

*Test 13 - Passed*
Input: [ {ingredient: 'x', amount: 1, unit: 'cup'}, {ingredient: 'x', amount: 1, unit: 'cup'} ]
Expected: [{ingredient: 'x', amount: 2.0, unit: 'cup'}]

*Test 14 - Passed*
Input: [ {ingredient: 'lem', amount: '1/3', unit: 'tsp'}, {ingredient: 'lem', amount: '2/3', unit: 'tsp'} ]
Expected: [{ingredient: 'lem', amount: 1.0, unit: 'tsp'}]

*Test 15 - Passed*
Input: [ {ingredient: 'p', amount: 0.1, unit: 'tsp'}, {ingredient: 'p', amount: 0.2, unit: 'tsp'}, {ingredient: 'q', amount: 0.15, unit: 'tsp'} ]
Expected: [ {ingredient: 'p', amount: 0.3, unit: 'tsp'}, {ingredient: 'q', amount: 0.2, unit: 'tsp'} ]

*Test 16 - Passed*
Input: [{ingredient: 'egg', amount: '1 1/4', unit: 'tbsp'}]
Expected: [{ingredient: 'egg', amount: 1.3, unit: 'tbsp'}]

*Test 17 - Passed*
Input: []
Expected: []

*Test 18 - Passed*
Input: [ {ingredient: 'a', amount: '1/4', unit: 'cup'}, {ingredient: 'a', amount: '1/4', unit: 'tbsp'}, {ingredient: 'a', amount: '1/4', unit: 'tsp'} ]
Expected: [{ingredient: 'a', amount: 4.3, unit: 'tbsp'}]

*Test 19 - Passed*
Input: [ {ingredient: 'z', amount: 1.05, unit: 'tsp'}, {ingredient: 'z', amount: 1.05, unit: 'tsp'} ]
Expected: [{ingredient: 'z', amount: 2.1, unit: 'tsp'}]

*Test 20 - Passed*
Input: [ {ingredient: 'corn', amount: 2, unit: 'tbsp'}, {ingredient: 'corn', amount: 1, unit: 'cup'}, {ingredient: 'salt', amount: 3, unit: 'tsp'}, {ingredient: 'pepper', amount: 1, unit: 'tbsp'} ]
Expected: [ {ingredient: 'corn', amount: 1.1, unit: 'cup'}, {ingredient: 'salt', amount: 1.0, unit: 'tbsp'}, {ingredient: 'pepper', amount: 1.0, unit: 'tbsp'} ]

## Comparison With Similar Designs

There are similar apps available on the market, including Paprika, Tandoor, and Recipe Keeper. However, DishDoc has key differences such as processing ingredient data, prioritizing user control, and a highly intuitive UI. These differences contribute to a better, more streamlined user experience that reduces cognitive load and maximises utility for users.

First and foremost, DishDoc's primary differentiator lies in its ability to process ingredient data across recipes and create an optimized shopping list. This addresses the consumer pain point of recipes listing items multiple times (such as ½ cup flour, 3 cups flour, 1 tbsp flour). For example, Paprika, Tandoor, and Recipe Keeper [2] [4] all use basic aggregation, listing every instance of an ingredient, whereas DishDoc automatically sums ingredients across recipes.

In addition, DishDoc prioritizes user control for categorization. While the competitors offer categories such as "Main Dish" or "Dessert", DishDoc allows users to create and manage their own tags and have personalized search criteria such as preparation method, dietary restrictions, and even sources that all meet the usability requirement of efficient content retrieval within 2 seconds.

Last but not least, while DishDoc is not open source like Tandoor [3], it offers a crucial advantage in its UI and UX. The aforementioned competitors have dated interfaces that feel cluttered or visually heavy to the user. DishDoc however, is designed from the ground up using a modern web stack (React/Node.js) to deliver a smoother, more aesthetic, and highly intuitive UI that's consistent across all browsers. This means a reduced cognitive load for users, enhancing satisfaction and making the core CRUD operations effortless.

## Conclusion

In this project, our team planned and designed DishDoc, a recipe management system built to be practical and easy to use. We established a solid foundation for developing a comprehensive and reliable application by defining clear requirements, outlining the system architecture, and dividing tasks among ourselves. Our decision to use the iterative process model helps ensure that core features, such as recipe creation, editing, and searching, can be developed in manageable steps and continually improved over time. The cost and effort estimates we came to indicate that the project is realistic within the planned schedule, and the successful test results confirm that key features, such as ingredient consolidation, function correctly. When compared to similar applications, DishDoc stands out by providing users with better control, utility, and convenience. However, we made one specific deviation during this analysis. Our original comparison point was data portability (focusing on import/export functionality), yet further research into our direct competitors (Paprika, Recipe Keeper, and Tandoor) revealed that their platforms also offer robust import/export functionality. Therefore, to maintain our edge we shifted the focus to an area where DishDoc outperforms the market, our smoother and highly intuitive UI/UX. Overall, DishDoc is a well-planned and thoughtfully designed project, with our deliverable laying a strong foundation for our application, and a focus on continued development and implementation over time.

**References**

[1] Jest, “Jest · 🃏 Delightful JavaScript Testing,” *Jestjs.io*, 2017. https://jestjs.io/

[2] “Paprika Recipe Manager for iOS, Mac, Android, and Windows,” *www.paprikaapp.com*. https://www.paprikaapp.com/

[3] “Tandoor: Smart recipe management,” *Tandoor.dev*, 2025. https://tandoor.dev (accessed Nov. 21, 2025).

[4] “Recipe Keeper App for iPhone, iPad, Android, Windows and Mac,” *recipekeeperonline.com*. https://recipekeeperonline.com/