ENSE 375 - Software Testing & Validation

Recipe Management System

Ramsha Naeem (200507574)
Kristina Langgard (200323529)
Pratik Gadhiya (200518183)

Table of Contents

1 Introduction

This project focuses on designing and developing a Recipe Management System, a Java-based desktop application that allows users to create, manage, and search their personal recipes.

We aim to simplify the process of storing and accessing culinary information with mindfulness of a user-friendly experience. In traditional alternatives, there is a lack of organization and limited accessible tools for individuals who enjoy cooking and want to manage their recipes. Existing solutions are often either overly complicated or too limited in functionality. We wish to modernize the experience to be convenient and approachable.

2 Design Problem

This project focuses on designing and developing a Recipe Management System, a Java-based desktop application that allows users to create, manage, and search personal recipes. The goal is to simplify the process of storing and accessing culinary information through a user-friendly platform. The motivation for this project stems from the lack of organized and accessible tools for individuals who enjoy cooking and want to digitally manage their recipes. Existing solutions are often either overly complicated or too limited in functionality.

2.1 Problem Definition

Challenges with Current Recipe Organization Methods:

1. Lack of Centralized Storage:
    - There are no established ways to consolidate handwritten recipes, printed cookbooks and bookmarked websites for easy viewing and retrieval.
2. Complexity of Existing Platforms:
    - Popular recipe organizing programs often provide more features than are necessary for simple users, leading to increased complexity and difficulty.
3. Limited Customization Options:
    - Existing platforms often don’t allow users to fully control their data or customize their recipe entries, such as adding personalized tags or categorizing based on meal type.

Advantages of Implementing a Digital Recipe Management Tool:

1. Streamlined Organization:
    - A dedicated desktop application allows recipes to be categorized, tagged, and searched quickly, enabling easy access to meal ideas.
2. Intuitive User Experience:
    - A simplified interface focused solely on recipe management avoids clutter and enhances usability for users of all skill levels.
3. Offline Functionality:
    - Unlike web-based platforms, this application will function fully offline, ensuring availability in all environments, including kitchens with limited connectivity.

Project Justification and Relevance:

The need for this project became clear after observing the difficulties faced by home cooks and hobbyists in managing recipes effectively. Paper-based systems are prone to damage and disorganization, while online platforms often come with distractions or require internet access.

Our proposed Recipe Management System addresses these issues by offering a simple, efficient, and accessible way to store and retrieve recipes, without unnecessary complications.

Common Issues and Proposed Solutions:

1. Disorganized Recipe Storage:
    - Issue: Users maintain recipes across notebooks, scraps of paper, or bookmarks, making them hard to locate when needed.
    - Resolution: The application stores all recipes in one place, with options to search by title or ingredients, making retrieval fast and intuitive.
2. Overly Complex Applications:
    - Issue: Many online tools require account registration, sync features, and advertisements, which can distract from the cooking process.
    - Resolution: Our system is designed for simplicity, providing essential recipe entry and management functionality with a clean and distraction-free interface.

Broader Impact and Benefits:

1. Cost and Time Efficiency:
    - Reducing reliance on physical notebooks or premium digital platforms helps save time and money in the long run.
2. Improved Cooking and Learning Experience:
    - Easier access to well-organized recipes enhances cooking productivity and helps users refine their culinary skills.
3. Expandability:
    - The system’s architecture supports potential enhancements such as printing recipes, exporting files, or future integration into mobile platforms, ensuring long-term usefulness.

2.2 Design Requirements

From our vision and outlined design problem, we are putting an emphasis on the database interactivity and readability of our program, focusing on its core functions and how users will interact with them.

2.2.1 Functions

These core functions are as outlined:

1. Create/store a recipe:
    a. Ability to create or upload a recipe in-app.
    b. Ability to save the recipe for later use.
2. Viewing a recipe:
    a. Ability to retrieve a saved recipe when needed.
    b. Ability to retrieve said recipe on different devices, such as a desktop computer, phone, or tablet.
    c. Ability to view recipes while offline.
3. Organizing recipes:
    a. Ability to apply custom tags/filters to recipes.
    b. Ability to sort by custom tags.
    c. Ability to search from saved recipes.
    d. Ability to favourite recipes.
    e. Ability to remove recipes that are no longer needed.
4. Sharing recipes:
    a. Ability to back-up a collection of recipes.
    b. Ability to share a collection of recipes with another user.

2.2.2 Objectives

To accomplish our goals with this application, we have the following objectives in place:

1. User-friendly:
    - the system should offer a clean and intuitive interface.
2. Efficient:
    - the system should respond quickly and minimize unnecessary steps.
3. Reliable:
    - the system should handle data accurately and predictably.
4. Maintainable:
    - the system should follow clean design principles with modular components to support future development.
5. Secure:
    - the system should validate user input to prevent invalid or malicious entries and ensure data integrity.

2.2.3 Constraints

Offline capability:

- An online database is not necessary to store and retrieve data, but rather a local storage system that mimics the concepts for retrieval and management in large collections.
- Sharing recipes and backing up a collection of recipes must be done in a way that can be done through file sharing, such as use of a JSON file to store and read information between independent instances of the application. This must include an import/export feature to handle these files for users so they may accomplish this seamlessly.

Security:

- No information needs to be passed online, and removes the need for account verifications.
- File and input verification are important to the integrity and function of the application in all aspects, so we must ensure all inputs to the application are appropriate and malicious behaviour is avoided. This should be achieved by preventing the use of restricted characters and file types.

File types and inputs:

- Files may not necessarily be text-based, and may also include images.
- Tags and filters can and should be able to be used based on text criteria, but must also be applied should the media a user intends to use includes images.

Tags and filters:

- They may not be based on text-skimming technology due to the variance in accepted file types.
- User may edit or change them independently.
- Favoriting recipes as a unique tag and sort option.

Removing recipes and data that are no longer wanted:

- Properly disposing of related data to remove unnecessary bloat and preserve privacy.

Economically:

- We do not wish to use any external services or paid APIs to accomplish the functions of our application.
- It is made to be free and open-source, to the benefit of our users.

Ethically:

- No user account or personal data should be collected, as part of the mindfulness of privacy and security, and all user inputs or outputs shall be handled equally without bias.

3 Solutions

3.1 Solution 1 (Java)

Solution 1 proposes building a basic Java desktop application for recipe management, where each recipe is stored as a separate plain text file (.txt) on the user's local computer. The application provides a minimal graphical user interface (GUI) for users to create, view, and delete recipes. All recipes are saved in a single folder, with each recipe occupying its own file.

How It Works

Recipe Creation

- The user clicks an “Add Recipe” button in the application.
- A form appears, allowing the user to enter the recipe title, ingredients, and instructions.
- When the user clicks “Save,” the application concatenates all the entered information into a single string and writes it to a new .txt file.
- The file is typically named after the recipe title (e.g., ChocolateCake.txt). If a file with the same name exists, a timestamp or number may be appended to avoid overwriting.

Recipe Viewing

- The application scans the designated recipes folder for all .txt files.
- It displays a list of these files (usually by filename) to the user.
- When a user selects a recipe, the application reads the contents of the corresponding .txt file and displays it in a simple, scrollable text area.

Recipe Deletion

- The user selects a recipe from the list and clicks a “Delete” button.
- The application deletes the corresponding .txt file from the folder.

User Interface

- The GUI is typically composed of:
- A list or table showing all recipe files.
- Buttons for “Add Recipe,” “View Recipe,” and “Delete Recipe.”
- A text area for displaying or editing recipe contents.

Example Recipe File Content

Recipe Title: Chocolate Cake
Ingredients:
- 2 cups flour
- 1 cup sugar

- 1/2 cup cocoa powder
- 2 eggs
Instructions:
1. Preheat oven to 350F.
2. Mix all ingredients.
3. Bake for 30 minutes.

Features and Limitations

Supported Features

- Basic CRUD: Create, Read, and Delete recipes.
- Offline Access: All data is local; no internet required.
- Simplicity: Easy to implement and understand.

Limitations

- No Structure: The data is just plain text, so there is no enforced format for ingredients, steps, or other fields.
- No Organization: No support for tags, categories, favorites, or sorting other than by filename.
- No Media Support: Cannot attach images or multimedia to recipes.
- No Search or Filter: Users cannot search for recipes by ingredient, tag, or keyword.
- No Import/Export: Recipes cannot be easily shared or backed up except by copying files manually.
- No Validation: There is no way to ensure users enter all necessary information or use a consistent format.

Testing and Validation Issues

- Difficult to Test Data Integrity: Since the file format is not enforced, automated tests cannot reliably check for missing or malformed fields.
- Limited Test Automation: Automated tests can only verify file creation, reading, and deletion—not the correctness of recipe content.
- No Advanced Features to Test: Search, filter, tagging, and image attachment cannot be tested because they do not exist.
- No Input Validation: The application cannot prevent invalid or incomplete recipes from being saved, making boundary value and equivalence class testing meaningless.

Why This Solution Is Not Selected

- Poor Testability: Most software testing techniques (e.g., boundary value, equivalence class, state transition, decision tables) cannot be applied to unstructured text files.
- Lack of Features: Does not meet the project requirements for organization, searchability, or extensibility.

• Not Scalable: As the number of recipes grows, managing them becomes cumbersome and error-prone.
• Difficult Maintenance: Any future improvements (like adding tags or images) would require a complete redesign.

In summary:
Solution 1a is a simple and easy-to-implement approach that works for the most basic use case of storing and viewing recipes. However, it lacks structure, organization, and extensibility, making it unsuitable for a robust, testable, and maintainable recipe management system.

3.2 Solution 2 (PHP/MySQL)

Our next consideration, since we already anticipated something web-based, was to use PHP and mySQL. Since we intend to store and retrieve recipes, this seemed like a natural choice, encouraging dynamic interaction. However, it lacked two key interests in our design; the ability to be standalone, and the ability to be easy to test.

We want our Recipe Management System to be able to be used on multiple devices, and be able to upload and download recipes of the user's own collection, so it may be used offline, and that is not feasible with something that exists solely on the web. Additionally, it would require the use of a domain, which, even if we hosted one ourselves, would have a cost associated with it. Lastly, there were also security risks to consider with it being online all the time. While the data we may be storing is unlikely to be sensitive, we should still be considering our users' privacy as a priority.

Finally, the ability to test with PHP is quite vague. Error messaging is lackluster, and the operation of commands between a database with mySQL would not prevent us from making mistakes. We consider this to be an important factor as well with this solution, as we do want to emphasize correctness and reliability in our project.

3.3 Final Solution

Our final solution (Solution 3) combines both ConsoleUI and JavaFX GUI interfaces, offering a flexible user experience. It retains the simplicity of JSON-based local storage, while also adding advanced features like recipe export/import (CSV/PDF) and search/filter functionality.

Why is Solution 3 better?

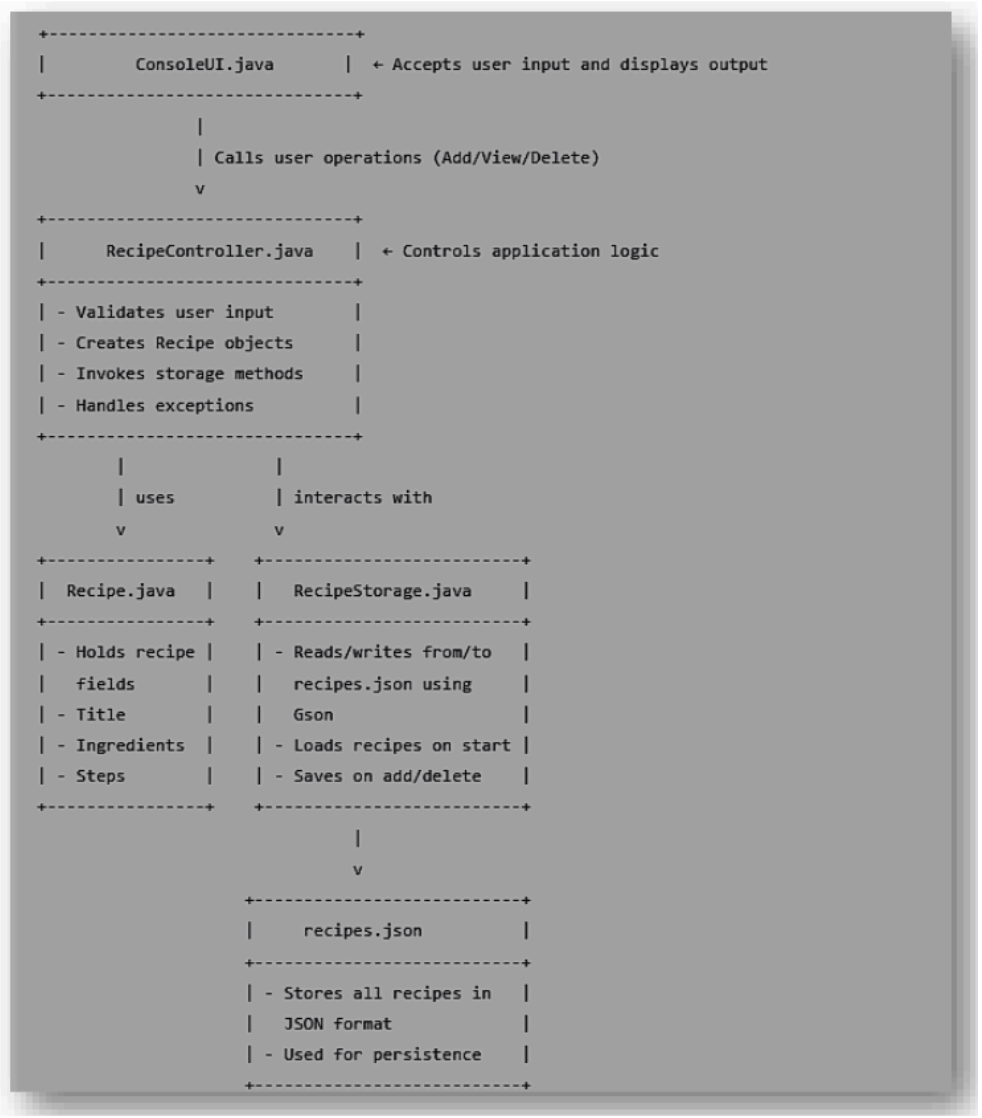
- Provides both Console & GUI options, meeting different user preferences.
- Improved test coverage using Boundary Value, Decision Table, State Transition, and Unit Testing.
- Fully offline and lightweight, requiring no external dependencies.
- Data security is higher due to local-only storage, with controlled export/import.

Comparison Table:

Feature/Criteria	Solution 1	Solution 2	Solution 3 (Final)
Interface	Console only	GUI only (complex)	Console + GUI (flexible)
Storage	Local JSON	Cloud DB (extra cost)	Local JSON (secure & simple)
Testing Coverage	Minimal	Moderate	Extensive (4 test suites)
Offline Support	Yes	No	Yes (full offline)
Export/Import	No	CSV only	CSV + PDF
Complexity & Usability	Low	High	Balanced & user-friendly

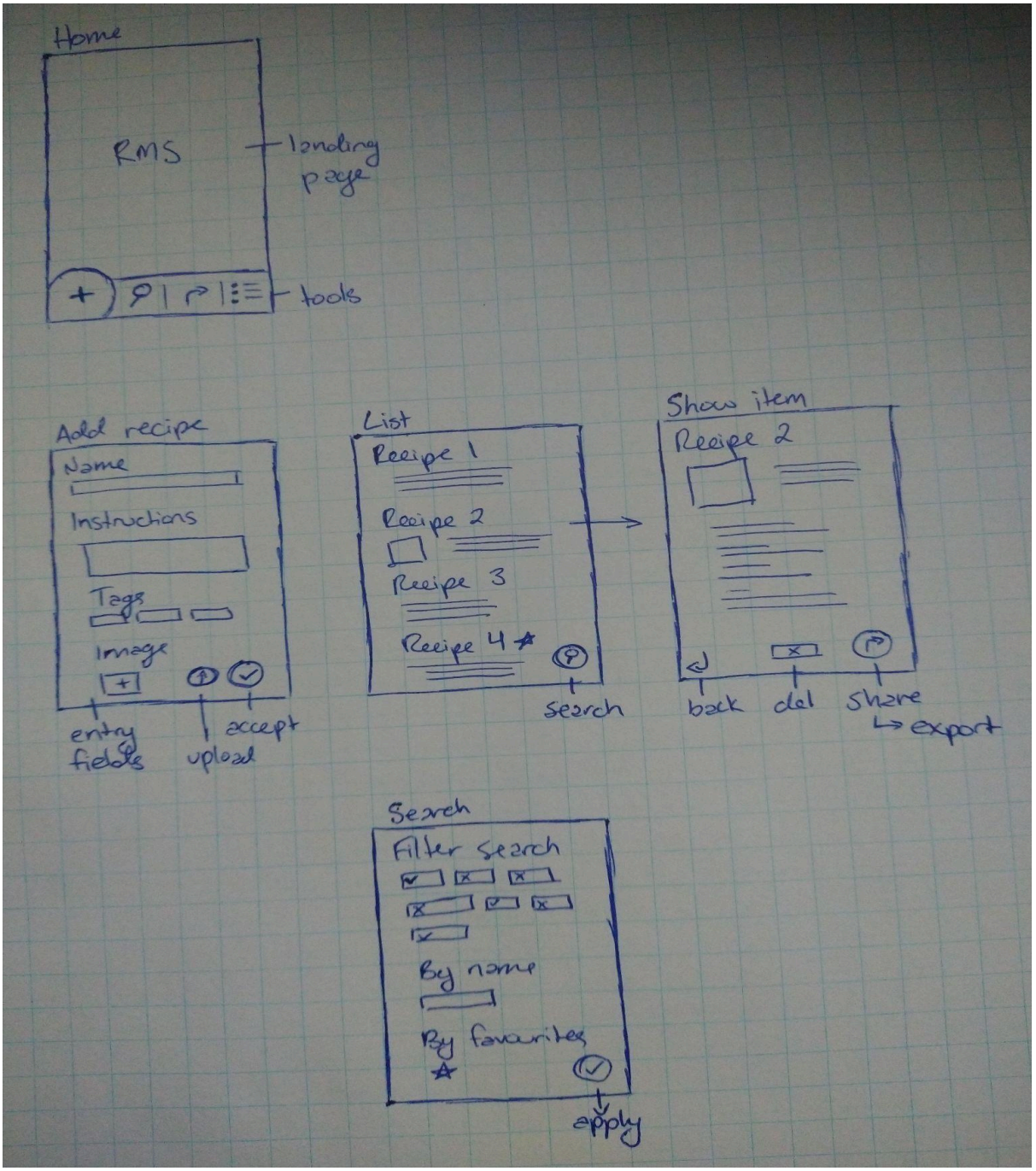
3.3.1 Components

• Recipe Model – Stores recipe data (title, ingredients, steps, categories, author).
• RecipeStorage (JSON) – Handles load/save, export/import of recipes.
• RecipeController – Acts as the logic layer between UI and storage.
• ConsoleUI & GUI (JavaFX) – Interfaces for user interaction (menu-driven & visual).
• Export Modules (CSV/PDF) – Allow users to share or back up recipes.
• Testing Methods:
• Recipe Model: Unit Testing (data validation).
• RecipeStorage: Boundary Value & Decision Table Testing (file operations & edge cases).
• RecipeController: State Transition Testing (app flow, add/delete).
• UI Components: Manual & Automated Unit Tests (menu navigation, button actions).



3.3.2 Features

As defined previously, the key features of our app will revolve around the ability to create/save a recipe, view a saved recipe, organize recipes by custom tags, and share a database with another user.



The user should find a landing page, from which they may select any of these core features to interact with the application. Viewing the list should show all stored recipes. There should be options to filter or search through these recipes, or to set these filters from the beginning, should that be more convenient.

Upon viewing a recipe, there should be an option to export the single recipe, or the entire database, so a user may share their recipes with other users. Similarly, when adding a recipe, a user should be able to upload them manually, or select an export file they were given to upload the information and begin using the new recipe(s) right away. A user should also be able to delete a recipe if they see fit, or be able to edit, add pictures, or change instructions. This information should all be saved and kept in the local database structure.

Some of these features should feel redundant, so they maintain consistency, and that is by design. Keeping the format simple is considered a feature here, as overcomplicated visions of these types of applications exist in plenty already.

3.3.3 Environmental, Societal, Safety, and Economic Considerations

Our considerations on the impact of our project were mostly baked into the idea of the project from the beginning. We wanted to create something that would respect our users, and in turn, respect the resources we are using to make and operate our project.

Environmentally, by keeping our software simple, we are trying to respect the resources that go into powering programs and the systems they run on. Our project does not need internet connection to run, requires very little in the way of prerequisite libraries and software, and does not need anything flashy or demanding to operate properly. This was a core concept to the simplicity of our design that we wanted to carry forward in all aspects, and while it definitely works in our favour to make our project more versatile and accessible, it certainly adds up and has advantages in environmental impact as well.

Societally, we want to see software that has less bloat. It has become very commonplace that similar applications would contain ads, take unnecessary space, and not necessarily even be as easy to operate in compensation. We want to create projects that do what they are tasked to do, without sacrificing performance or functionality for things that our users do not want. We see this approach as something we need to carry every day in the work we do.

Safety was an easy metric to address, where in our application the main concern is data security. Because this application works offline, and information can only be shared privately, there are no concerns to worry about with encryption level or data theft. We hope that this shows respect to our users' privacy as well as their autonomy as they use our application.

Finally, economically, we stand to make no profit from this project given the circumstances of its creation to begin with, and it cost us nothing to make except our time. We have made something to put out into the world for the betterment of the uses it provides, and nothing more.

Overall, we believe the considerations we made with this project exhibit the respect to other people who will make use of our project that we expect in applications we use ourselves. These are qualities we push onto ourselves in our work, and hope to positively reflect in our careers in the future.

3.3.4 Test Cases and results

We developed a suite of tests to validate each component:

- Boundary Value Testing: Checked limits (e.g., max ingredients, empty titles).
- Decision Table Testing: Verified combinations of valid/invalid inputs.
- State Transition Testing: Ensured proper navigation between menus and states.
- Unit Testing: Validated individual methods (add, delete, search, export).

Results: All tests passed successfully. The system behaved correctly under normal and edge cases, demonstrating stability and reliability.
Tests were executed using `mvn clean test` and all passed successfully.

3.3.5 Limitations

- No built-in cloud backup or sync (only local storage).
- GUI is basic and may need enhancements for large recipe collections.
- Currently supports only CSV & PDF exports (no direct Word/Excel export).
- No multi-user authentication system; optional login is basic.

4 Team Work

Weekly meeting records

4.1 Meeting 1

Time: May 14, 2025: 2pm to 5pm
Agenda: Problem Definition Discussion

Team Member	Previous Task	Completion State	Next Task
Ramsha Naeem	N/A	N/A	Detailed Problem Analysis
Kristina Langgard	N/A	N/A	Detailed Problem Analysis
Pratik Gadhiya	N/A	N/A	Detailed Problem Analysis

4.2 Meeting 2

Time: May 19, 2025: 5pm to 9pm
Agenda: Design Constraints and Requirements Discussion

Team Member	Previous Task	Completion State	Next Task
Ramsha Naeem	Detailed Problem Analysis	100%	Define Limitations and Gather Requirement
Kristina Langgard	Detailed Problem Analysis	100%	Define Limitations and Gather Needs
Pratik Gadhiya	Detailed Problem Analysis	100%	Specify Constraint

4.3 Meeting 3

Time: May 23, 2025: 5pm to 9pm
Agenda: Solution Discussion and Delegation of Tasks for Solution 1

Team Member	Previous Task	Completion State	Next Task

Ramsha Naeem	Define Limitations and Gather Requirement	100%	Develop Ideas and Code Implementation for Solution 1
Kristina Langgard	Define Limitations and Gather Needs	100%	Develop Ideas and Code Implementation for Solution 1
Pratik Gadhiya	Specify Constraint	100%	Identify Possible Approaches

4.4 Meeting 4

Time: May 28, 2025: 2pm to 5pm
Agenda: Solution 1 Progress Check and Solution 2 Task Distribution

Team Member	Previous Task	Completion State	Next Task
Ramsha Naeem	Develop Ideas and Code Implementation for Solution 1	100%	Examine Solution 1 Outcomes and Write Code for Solution 2
Kristina Langgard	Develop Ideas and Code Implementation for Solution 1	100%	Examine Solution 1 Outcomes and Write Code for Solution 2
Pratik Gadhiya	Identify Possible Approaches	100%	Review Solution 1 and Perform Markdown Syntax Check

4.5 Meeting 5

Time: May 28, 2025: 2pm to 5pm
Agenda: Review Progress on Solution 2

Team Member	Previous Task	Completion State	Next Task
Ramsha Naeem	Examine Solution 1 Outcomes and Write Code for Solution 2	80%	Finalize Solution 2
Kristina Langgard	Examine Solution 1 Outcomes and Write Code for Solution 2	75%	Finalize Solution 2

Pratik Gadhiya	Review Solution 1 and Perform Markdown Syntax Check	75%	Complete REPORT.md up to Week 7

4.6 Meeting 6

Time: June 1, 2025: 2pm to 5pm
Agenda: Discuss Final Solution Planning

Team Member	Previous Task	Completion State	Next Task
Ramsha Naeem	Finalize Solution 2	100%	Brainstorm features for final solution
Kristina Langgard	Finalize Solution 2	100%	Plan testing for final solution
Pratik Gadhiya	Complete REPORT.md up to Week 7	100%	Brainstorm features for final solution

4.7 Meeting 7

Time: June 14, 2025: 5pm to 8pm
Agenda: Assign Final Solution Tasks

Team Member	Previous Task	Completion State	Next Task
Ramsha Naeem	Brainstorm features for final solution	100%	Implement and test the final solution
Kristina Langgard	Plan testing for final solution	100%	Develop code for the final solution
Pratik Gadhiya	Brainstorm features for final solution	100%	Provide feedback on the final solution

4.8 Meeting 8

Time: June 15, 2025: 2pm to 6pm
Agenda: Review Final Solution Progress

Team Member	Previous Task	Completion State	Next Task

Ramsha Naeem	Implement and test the final solution	50%	Continue coding and testing the solution with Documentation
Kristina Langgard	Develop code for the final solution	50%	Continue developing the final solution
Pratik Gadhiya	Provide feedback on the final solution	60%	Continue feedback

4.9 Meeting 9

Time: June 12, 2025: 4pm to 9pm
Agenda: Final Progress Review

Team Member	Previous Task	Completion State	Next Task
Ramsha Naeem	Continue coding and testing the solution with Documentation	80%	Finalize coding, testing, and documentation
Kristina Langgard	Continue developing the final solution	80%	Finalize developement
Pratik Gadhiya	Continue feedback	70%	complete report

4.10 Meeting 10

Time: July 25, 2025: 2pm to 5pm
Agenda: Presentation Task Assignment

Team Member	Previous Task	Completion State	Next Task
Ramsha Naeem	Finalize coding, testing, and documentation	100%	Prepare slides and rehearse presentation
Kristina Langgard	Finalize developement	100%	Prepare for presentation
Pratik Gadhiya	complete report	100%	Finalize REPORT

4.11 Meeting 11

Time: July 28, 2025: 2pm to 5pm
Agenda: Presentation Recording

Team Member	Previous Task	Completion State	Next Task
Ramsha Naeem	Prepare slides and rehearse presentation	100%	Complete recording and all documentation
Kristina Langgard	Prepare for presentation	100%	Complete the recording
Pratik Gadhiya	Finalize REPORT	100%	Complete the recording

5 Project Management