Recipe Vault - Final Report

UMGC CMSC 495 Section 7382

6 May 2025

Gaurab Shrestha

Josh Kinnes

Gracie Saxon

Karina Ortega

Table of contents

Introduction

In today's digital-first world, the way people store and share personal recipes is evolving. Traditional recipe cards and handwritten notes are being replaced by streamlined, shareable digital formats. The Recipe Vault application was developed in response to this need, offering a user-friendly platform for entering, storing, and exporting recipes into elegant, themed PDF documents. Whether preserving cherished family meals or compiling recipes for a themed cookbook, users benefit from the structure and customization offered by the app.

The initial concept for Recipe Vault was ambitious: the team hoped to incorporate editable PDF functionality and even integrate AI to suggest or enhance recipes. However, as development progressed, it became clear that prioritizing simplicity and user experience would yield a more polished and reliable product within the project's timeframe. This decision aligned with user feedback and usability testing, leading to a streamlined application that focuses on intuitive design, quick data entry, and visually appealing exports.

Throughout the process, the team remained attentive to the client's original requirements while also incorporating refinements based on feedback from friends and family. This evolution led to thoughtful features such as seasonal PDF templates, real-time field validation, and character limits—all implemented with the goal of making the application accessible to users of all technical backgrounds.

From a technical standpoint, Recipe Vault combines Java, JavaFX, and Apache PDFBox to offer a cohesive and reliable user experience. The project's modular design, attention to interface usability, and thoughtful scope decisions reflect strong problem formulation and practical trade-offs. This report details the development lifecycle of Recipe Vault, highlighting key design choices, collaborative teamwork, and testing strategies that shaped the final product.

Process Overview

The development of the Recipe Vault application was carried out using a collaborative team-based approach guided by project planning, iterative feedback, and task delegation. The team utilized Java as the core programming language, with JavaFX for crafting the user interface and Apache PDFBox to generate PDF documents from recipe data. Development tools included Visual Studio Code, Git for version control, and Java SDK for compiling and testing the application. The design phase involved sketching mockups, creating reusable components, and preparing form-fillable PDF templates. As the project progressed, the team followed a timeline with clear milestones, enabling each member to focus on specific roles while maintaining open communication and code integration.

The success of the project relied heavily on the coordinated efforts of all members:

- **Josh (Team Lead):**
  - Oversaw the entire development process and served as the main point of coordination.
  - Managed the project timeline and ensured that all team members stayed on track with their deliverables.
  - Provided code review and feedback, guiding technical decisions and feature integration.
  - Created the batch file used to launch the application, improving ease of use for end users and testing to overlook the correct functionality of the application.
  - Compiled documentation and ensured all deliverables met project requirements.
- **Karina:**
  - Contributed to the development of user interface elements, aesthetic layout and application testing.
  - Participated in the creation of visually appealing and functional PDF templates.
  - Assisted in writing and formatting documentation for both the application's functionality and user guidance.
  - Ensured visual consistency and theme alignment across the application's components.
- **Gracie:**
  - Led the implementation of the UI components using JavaFX, organizing the layout and refining visual elements.
  - Enhanced the user experience by applying design improvements and responsive layout techniques.
  - Took the lead in technical writing, including documenting the application's architecture and usage instructions.
  - Worked closely with team members to refine the user interface based on feedback and testing.
- **Gaurab:**
  - Implemented the core functionality of the application, including logic for data entry, list manipulation, and form resets.
  - Developed the integration with Apache PDFBox, enabling recipes to be saved into themed, structured PDF files.
  - Created and tested form-fillable PDF templates, ensuring compatibility with the data model.
  - Designed and executed test scenarios to verify proper functionality across various user inputs and edge cases.

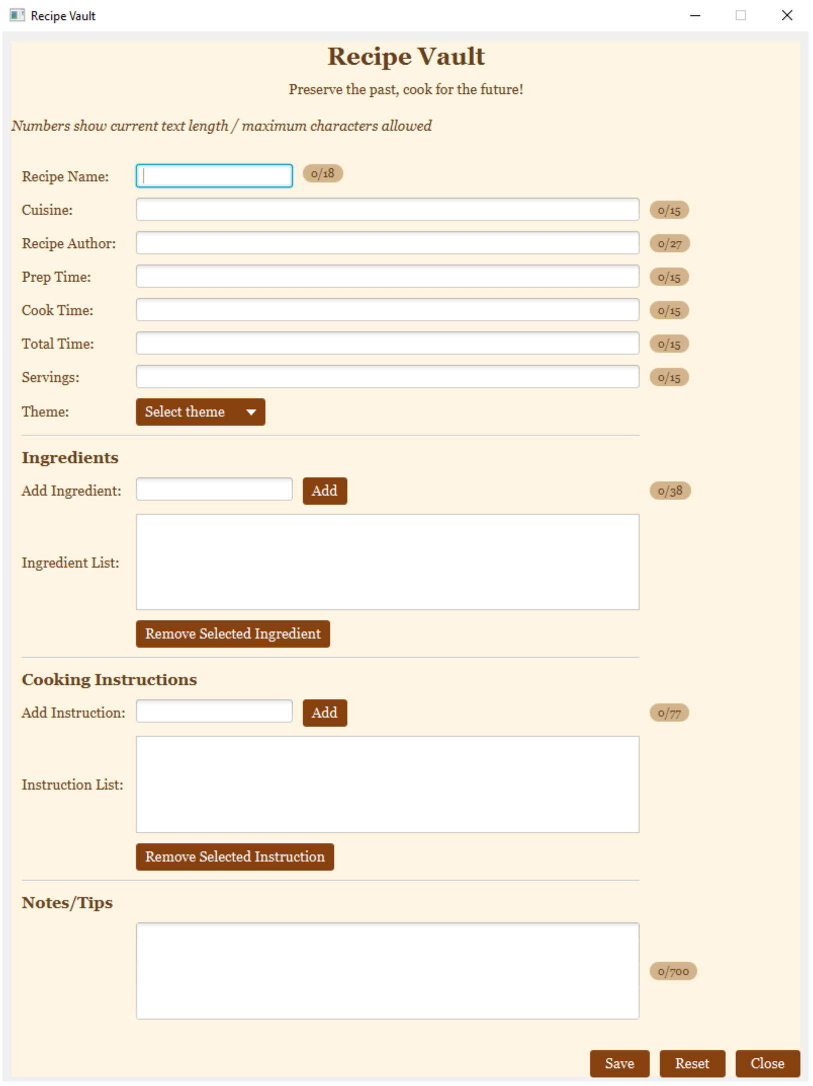
This collaborative effort allowed the team to transform a conceptual idea into a functional, polished application by combining technical skill sets with strong communication and task management.

Requirements Specification

The Recipe Vault application includes a comprehensive set of functional and non-functional requirements to ensure both usability and reliability for end-users. These requirements were defined through a combination of user expectations, practical usability needs, and technical constraints. User input, including polling from friends and family members, was sought early in the design phase to guide decisions around features and interaction flow, helping tailor the application to real-world expectations.

- **Functional Requirements:**
    - Users must be able to input recipe metadata, including name, category, author, prep time, cook time, total time, servings, and select a seasonal theme.
    - The application must allow users to add up to 15 ingredients and 25 instruction steps.
    - Users must be able to remove ingredients and instructions from their respective lists.
    - The application must enforce character limits on text fields and show dynamic counters that provide visual feedback.
    - The ‘Save’ function must validate all required fields before generating and saving a filled PDF file based on the selected template.
    - Reset and Close buttons must clear the form and close the application, respectively.
    - Users must be informed via pop-up dialogs when errors occur (e.g., exceeding limits, missing required fields).
- **Non-Functional Requirements:**
    - The system must be implemented in Java 22 with a JavaFX-based user interface.
    - The interface must be intuitive and visually appealing, styled with consistent fonts and spacing.
    - The application must be responsive, display-centered, and maximized vertically on launch.
    - The application must not require internet connectivity or an external database, ensuring full offline usability.
    - PDF templates and resources must be accessed reliably across different user environments using classpath-based resource loading.
    - Error messages and field validations must be clear and user-friendly, minimizing user confusion and support needs.

These requirements ensure the Recipe Vault is not only functionally complete but also easy to use, robust, and usable across machines. By incorporating character limits, input validation, and clear user feedback mechanisms, the application reduces user error and enhances the overall experience. The use of embedded PDF templates and local resource access ensures cross-platform compatibility and avoids reliance on external systems or internet connectivity. Furthermore, the consistent structure of the interface, thoughtful layout design, and real-time data validation all contribute to a streamlined and intuitive workflow, making the application suitable for users with varying levels of technical experience.



Recipe Vault

Preserve the past, cook for the future!

Numbers show current text length / maximum characters allowed

Recipe Name:	0/18
Cuisine:	0/15
Recipe Author:	0/27
Prep Time:	0/15
Cook Time:	0/15
Total Time:	0/15
Servings:	0/15
Theme:	Select theme
Add Ingredient:	0/38
Add Instruction:	0/77
Notes/Tips	0/700

Figure 1 - Recipe Vault Graphical User Interface

Project Design

The architecture of the Recipe Vault application follows a modular layered design with clear separations between the presentation layer, the application logic, and the export layer that is responsible for the PDF generation. This approach was chosen to maximize maintainability and allow future enhancements without affecting the application's core functionality. The graphical user interface (GUI) was built using JavaFX, which provides a robust framework for building applications with responsive and customizable layouts. The application is structured around a single Main class that houses the GUI, logically segmented into methods that perform input validation, event handling, and user interaction. The RecipeWriterPDF class is responsible for loading pre-formatted seasonal PDF templates and writing user-submitted data into the form fields using the Apache PDFBox library. This decoupling allows for easy modification of the GUI without interfering with the data export logic and vice versa.

User input is collected through form elements such as TextFields, TextAreas, and ListViews, with real-time character counters and input validation that guides the user and prevents invalid data entry. Ingredient and instruction lists are limited to 15 and 25 entries, respectively, with visible feedback if those limits are exceeded. On save, the application validates all required fields and then loads the appropriate seasonal template based on the selected theme. The PDF is filled using field names mapped directly from the template and then saved to the user’s selected location via a FileChooser.

The use of JavaFX ensured native support across platforms but required the inclusion of JavaFX modules and proper VM arguments, which increased setup complexity. The use of Apache PDFBox was a strategic tradeoff as it increased the application size and memory usage, but gave precise control over form field population in PDF templates. This was essential for the completion of this project. Local storage was chosen over creating a database for recipe storage to simplify resource demands and avoid the need for internal storage. This decision resulted in the inability for users to edit recipes at a later time; however, due to the project scope and increased security overhead, the trade-off was accepted. While the decision to forgo internal storage limits some functionality, it significantly reduced development complexity and aligned with the primary goal of producing a clean, dependable, user-driven recipe exporter.

The architecture of the Recipe Vault project ensures that each module is focused on ease of use, clarity, and reusability. To ensure the design would resonate with end users, a poll was done of friends and family for input on layout preferences, usability, and theme selection, incorporating their feedback before finalizing the user interface and overall structure. By carefully weighing performance, maintainability, and resource constraints, an application was created that is both functional and visually polished. The final implementation reflects a strong balance between practical design decisions and user-centric development.

Several UML Diagrams were created to facilitate robust project design. Three of the sequence diagrams created are provided in figures 2, 3, and 4.

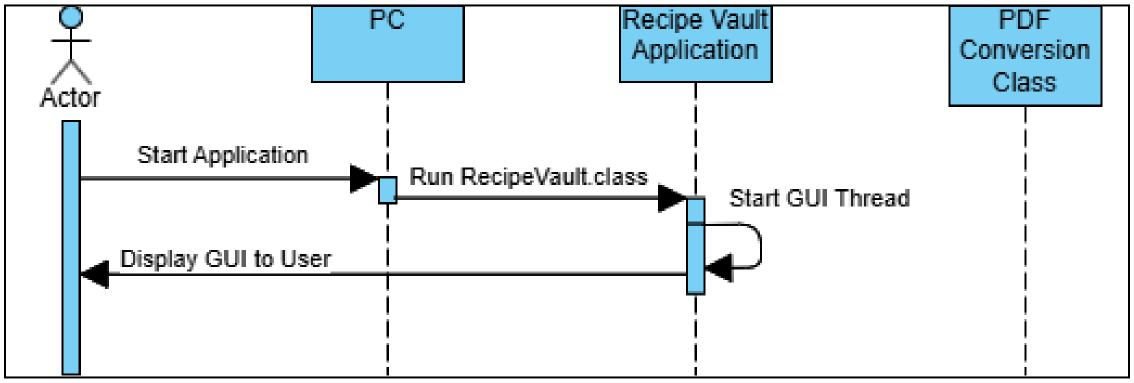


Figure 2 - Application Start

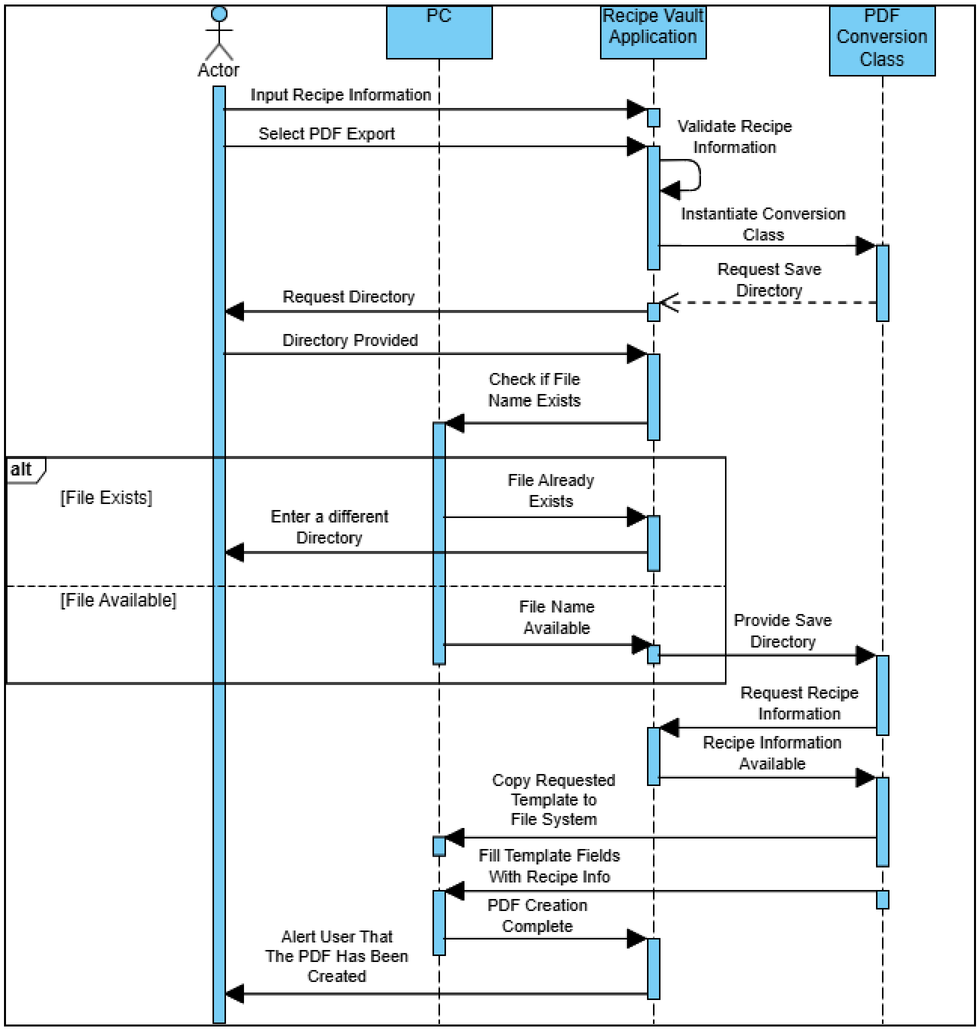

Actor
PC
Recipe Vault Application
PDF Conversion Class
Input Recipe Information
Select PDF Export
Validate Recipe Information
Instantiate Conversion Class
Request Save Directory
Request Directory
Directory Provided
Check if File Name Exists
alt
[File Exists]
File Already Exists
Enter a different Directory
[File Available]
File Name Available
Provide Save Directory
Request Recipe Information
Recipe Information Available
Copy Requested Template to File System
Fill Template Fields With Recipe Info
PDF Creation Complete
Alert User That The PDF Has Been Created

Figure 3 - PDF Generation

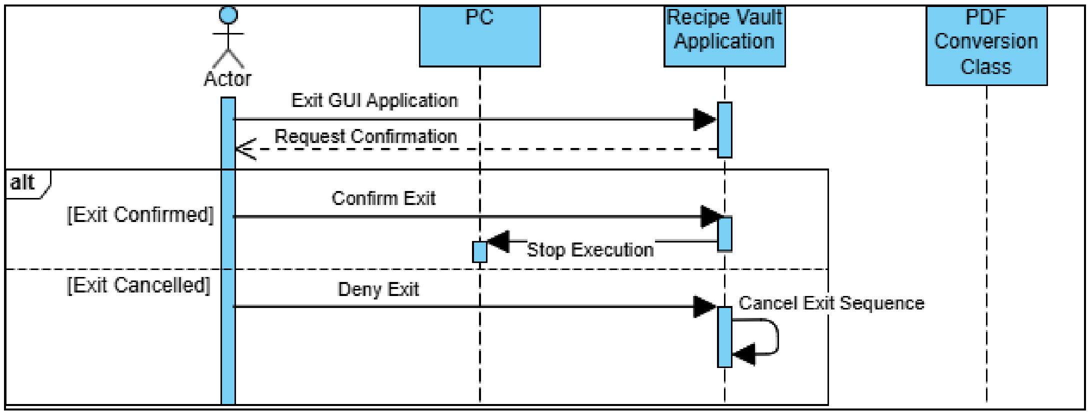


Figure 4 - Application Exit

Project Evaluation

The project was evaluated utilizing a systematic test plan that ensured each feature of the application was operating as intended. In order to facilitate effective testing, the test plan was broken into several sections following each of the major functions of the application. By breaking down the testing, the team was able to create a more robust and in-depth test procedure. Each of the test sections that were conducted and the aspects of the program that they tested are listed below. The full test plan can be found in Appendix E:

- Installation Verification
  - Ensure the program properly runs on the given system.
- GUI Responsiveness
  - GUI buttons implement intended functionality
  - Character counts are tracked and enforced
- Data Manipulation
  - Program tracks whether required fields are filled
  - Correct number of steps/ingredients can be added
- File System Access
  - Saving prompts file system dialogue
  - File system can be successfully accessed
- PDF Conversion
  - The correct PDF template is selected from the available templates
  - Data from the application GUI is correctly filled into the PDF
- File Saving
  - Ensure the program can successfully fill and save the PDF file.

By testing the program systematically, the Recipe Vault team was able to identify problems within the program and correct them before the Phase 2 Source release. While not a formal run of the test plan, having the line items and procedures in place allowed development-level testing to proceed at a much faster rate than would have otherwise been possible. When the final test plan was formally executed, the program successfully executed all test cases and was released with the Phase 2 delivery.

Design and Alternate Designs

1. Overall Architecture:

The Recipe Vault application adopts a Model-View-Controller (MVC) architectural pattern, fostering a clear separation of concerns between data management, user interface presentation, and application logic.

- **Model (Implicit):** The Main class serves as the central repository for the application's state. Instance variables within Main, such as recipeNameField, recipeCategoryField, ingredientsList (an ObservableList<String>), instructionsList (an ObservableList<String>), and notesArea, collectively represent the data model for the currently active recipe being created or edited by the user. These in-memory structures directly reflect the information manipulated through the user interface.
- **View:** The visual aspect of the application is entirely constructed using the JavaFX Scene Graph within the start(Stage primaryStage) method of the Main class and its associated helper methods (createHeader(), createContentGrid(), createButtons()). Each UI component, including Label, TextField, ListView, Button, and ComboBox, forms a node within this hierarchical structure. Layout panes like BorderPane, GridPane, HBox, and VBox are employed to organize these components into a user-friendly and visually coherent interface. Styling using CSS-like syntax (-fx-) is applied directly within the code to define the application's visual theme (cream background, brown accents, Georgia font).
- **Controller:** User interactions within the view trigger specific events. These events are handled by event handlers attached to the interactive UI components. Examples include:
    - **Data Input:** Typing within TextFields automatically updates their corresponding text properties. This change is often bound to other UI elements, such as the character count Labels, providing immediate visual feedback driven by changes in the underlying model.
    - **List Manipulation:** Clicking the "Add" buttons for ingredients and instructions triggers the addIngredient() and addInstruction() methods within the Main class. These methods act as controllers, modifying the ingredientsList and instructionsList (the model). Due to the observable nature of these lists, the associated ListViews (the view) are automatically refreshed to reflect these changes. Similarly, the "Remove Selected" buttons invoke removeIngredient() and removeInstruction(), again acting as controllers to update the model and subsequently the view.
    - **Form Management:** The "Reset" button calls the resetForm() method, which clears all input fields and the observable lists, effectively resetting the application's model and the displayed view.

    - **Data Persistence:** The "Save" button initiates the crucial data persistence process by calling the saveRecipeToPDF() method within the Main class. This method acts as an orchestrator, retrieving the recipe data from the current model and delegating the task of generating the PDF file to the RecipePDFWriter class.
    - **Application Lifecycle:** The "Close" button triggers the stage.close() method, directly controlling the application's termination.

The RecipePDFWriter class functions as a specialized service or controller dedicated to the specific task of converting the in-memory recipe data into a persistent PDF format. It operates based on user-selected themes and utilizes the Apache PDFBox library to interact with pre-designed PDF templates.

2. Implementation Details:

The Recipe Vault application is implemented using Java and the JavaFX framework for the user interface, with the Apache PDFBox library integrated for PDF generation.

- **JavaFX Foundation:** The Main class extends javafx.application.Application, serving as the entry point for the JavaFX application. The start(Stage primaryStage) method is the core of the UI initialization, responsible for creating the main application window (Stage), setting its properties (title, initial dimensions), and constructing the entire scene graph that defines the user interface.
- **Modular UI Construction:** The UI is built in a modular fashion using helper methods (createHeader(), createContentGrid(), createButtons()) within the Main class. These methods encapsulate the instantiation, configuration, and layout of individual JavaFX UI components. Layout panes are strategically used to arrange these components into a user-friendly and responsive design. Visual styling is applied directly to the components using inline CSS-like syntax.
- **Reactive Data Handling:** JavaFX's powerful property binding mechanism is employed to create dynamic relationships between UI elements and the underlying data model. For instance, the textProperty() of TextFields is bound to the textProperty() of the character count Labels, ensuring real-time updates as the user types. The ObservableLists for ingredients and instructions are fundamental for managing dynamic collections of data that are directly reflected in the ListViews. Any changes to these observable lists automatically trigger updates in the corresponding UI elements. Event listeners (setOnAction, setOnKeyPressed) are attached to interactive UI components to handle user-initiated actions.
- **User Input Management:** Input from TextFields and the TextArea is directly accessed through their respective text properties. The TextFormatter class is utilized to enforce character limits on text-based input fields, ensuring data integrity and preventing overly long entries. The setOnKeyPressed event handler added to the ingredient and instruction

input fields allows users to add new items by simply pressing the Enter key, streamlining the input process.

- **The Role of RecipePDFWriter (Apache PDFBox Integration):** The RecipePDFWriter class encapsulates the logic for saving recipe data to a PDF file using pre-designed templates and the Apache PDFBox library.
    - The saveRecipeToPDF() method within RecipePDFWriter receives all the recipe details as parameters.
    - Based on the user-selected theme, it determines the appropriate PDF template file name (e.g., "Summer_Template.pdf") and attempts to load it as an InputStream from the application's resources using RecipePDFWriter.class.getResourceAsStream().
    - It then utilizes the Apache PDFBox library's core classes (PDDocument, PDAcroForm, PDField) to:
        - Load the PDF template from the InputStream.
        - Access the interactive form (PDAcroForm) within the loaded PDF document.
        - Retrieve specific form fields within the template by their defined names (e.g., "Recipe", "Author", "Ingredient1", "Direction1", "Notes").
        - Set the values of these retrieved form fields using the recipe data passed to the method. For ingredients and instructions, it iterates through the respective lists and populates sequentially named form fields (e.g., "Ingredient1", "Ingredient2", and "Direction1", "Direction2").
    - To allow the user to choose the save location and filename, a FileChooser is presented.
    - Finally, the modified PDDocument (now containing the filled recipe data) is saved to the user-specified file.
- **Resource Handling:** The application employs getResourceAsStream() to correctly access PDF template files that are bundled within the application's resources. This ensures that the application can locate and utilize these templates regardless of how it is deployed (e.g., as a standalone JAR file). The use of try-with-resources blocks when handling PDDocument and InputStream objects guarantees that these resources are properly closed after use, preventing potential resource leaks.
- **User Feedback via Alert Dialogs:** The showMessageDialog(), showErrorDialog(), and showInfoDialog() methods within the Main class provide a consistent mechanism for displaying feedback to the user in the form of standard JavaFX Alert dialogs. These dialogs are used to communicate validation errors, informational messages (e.g., successful save), and other relevant application states.

3. How it Works (End-to-End User Workflow):

1. **Application Launch:** The user initiates the Recipe Vault application, leading to the execution of the main() method in the Main class, which in turn launches the JavaFX runtime and displays the primary application window defined in the start() method.
2. **Recipe Data Input:** The user interacts with the various UI controls to input recipe details. This includes typing text into fields for metadata, ingredients, instructions, and notes, as well as selecting a theme from the ComboBox. Real-time feedback, such as character counters and updates to the ingredient and instruction ListViews, keeps the user informed of their input.
3. **Initiating Save:** Once the user has entered all the desired recipe information, they click the "Save" button.
4. **Data Validation:** Upon clicking "Save," the validateFields() method in the Main class is executed. This method checks for completeness in required fields and adherence to list size limits for ingredients and instructions. If any validation rules are violated, an error dialog is displayed to the user, and the saving process is interrupted.
5. **PDF Generation Orchestration:** If the data validation is successful, the saveRecipeToPDF() method within the Main class is called. This method then delegates the actual PDF generation to the RecipePDFWriter class, passing the collected recipe data as arguments.
6. **Template Loading and Data Merging:** The RecipePDFWriter's saveRecipeToPDF() method determines the appropriate PDF template based on the selected theme and loads it from the application's resources. It then accesses the interactive form fields within the template using Apache PDFBox and populates these fields with the provided recipe data.
7. **Save File Selection:** A FileChooser dialog is presented to the user, allowing them to specify the desired save location and filename for the generated PDF recipe file.
8. **PDF Saving:** Once the user confirms the save location, the RecipePDFWriter saves the filled PDF document to the chosen file using Apache PDFBox.
9. **User Confirmation:** A success message is displayed to the user via an information dialog, confirming that the recipe has been saved successfully.
10. **Optional Actions:** The user can choose to "Reset" the form to clear all entered data and begin entering a new recipe or "Close" the application.

In summary, the Recipe Vault application provides a user-friendly JavaFX interface for recipe input. Upon the user's request to save, it leverages the RecipePDFWriter class and the Apache PDFBox library to seamlessly merge the entered recipe data with pre-designed, theme-specific PDF templates, resulting in a well-structured and visually appealing digital record of their culinary creations. The clear separation of UI logic (in Main) and PDF generation logic (in RecipePDFWriter) promotes modularity and maintainability within the project.

4. Key Modification after testing:

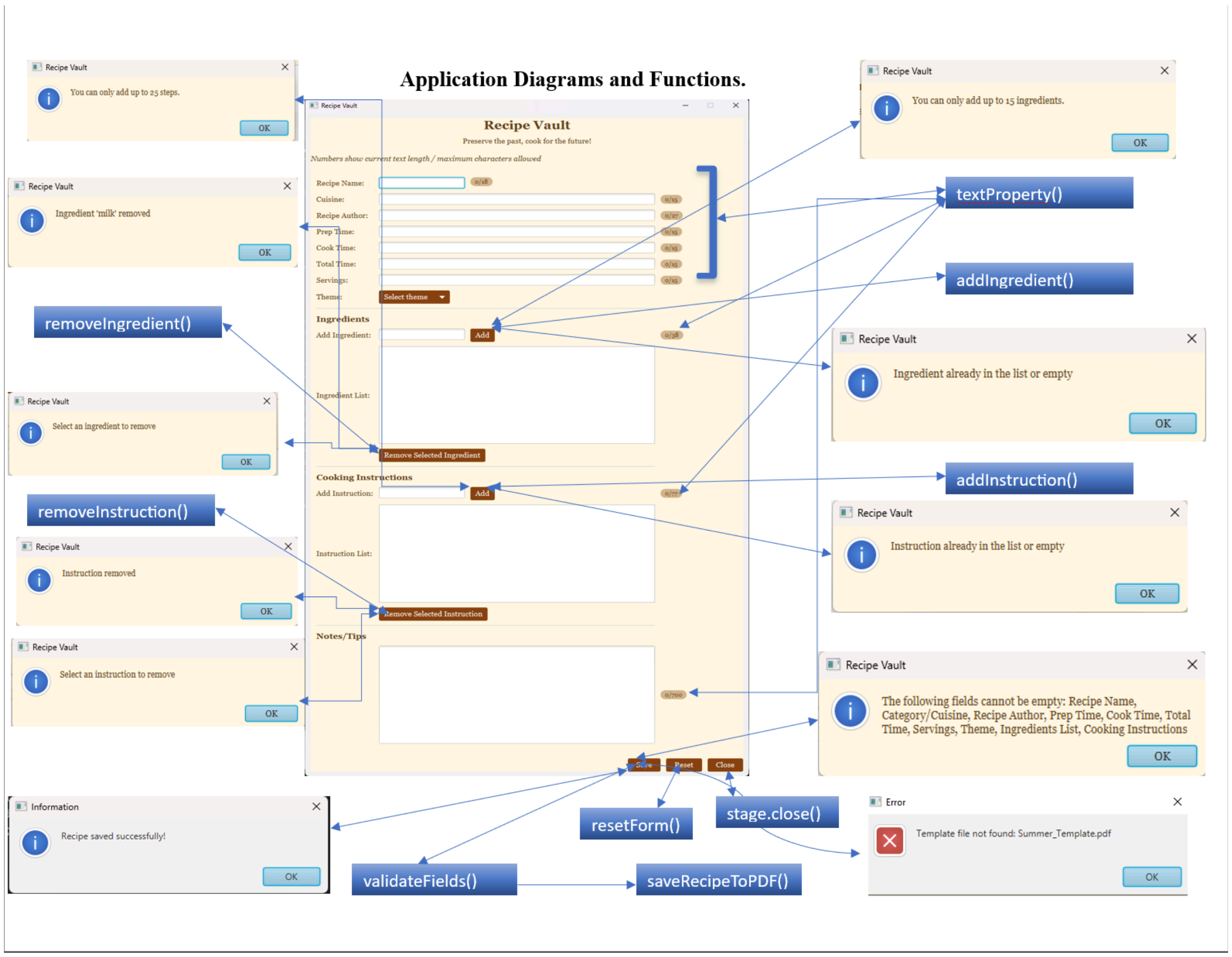
- **Character Limits for PDF Template Compatibility:**
    - **Problem Identified:** During testing, it was observed that user-entered text in various input fields could exceed the space allocated for those fields within the PDF templates, leading to text overflow and a poorly formatted output.
    - **Implementation:** To address this, character limits were implemented for each relevant input field (recipeNameField, recipeCategoryField, prepTimeField, cookTimeField, totalTimeField, servingsField, recipeAuthorField, ingredientField, instructionField, and notesArea) within the Main class. This was achieved using JavaFX's TextFormatter. For each TextField and TextArea, a TextFormatter was created with a filter that rejects any changes that would exceed the predetermined character limit for that specific field, ensuring the text fits appropriately within the corresponding PDF template form field.
    - **Impact:** This change guarantees that the data passed to the RecipePDFWriter adheres to the layout constraints of the PDF templates, resulting in well-formatted and readable PDF recipe documents.
- **GUI Enhancement and Polishing:**
    - **Problem Identified:** Initial versions of the GUI lacked a cohesive and visually appealing design, potentially impacting user engagement and satisfaction.
    - **Implementation:** Significant effort was invested in refining the application's visual appearance. This involved:
        - Defining and applying a consistent color palette throughout the application, utilizing a cream background (CREAM_BACKGROUND) and various shades of brown (BROWN_BUTTON, BROWN_LIGHT, BROWN_DARK) for text, buttons, and accents. These color constants were defined in the Main class for easy management.
        - Styling individual UI components using inline CSS-like syntax (-fx-) to apply these colors consistently. This included setting background colors for panes, text colors for labels and input fields, and button styles.
        - Specific styling was applied to elements like the ComboBox for theme selection to ensure it integrated visually with the overall color scheme.
    - **Impact:** These visual enhancements resulted in a more polished and aesthetically pleasing user interface, improving the overall user experience and making the application more inviting.

- **Real-time Character Counters:**
  - **Problem Identified:** Users lacked clear feedback on the character limits imposed on the input fields, potentially leading to frustration when their input was truncated or rejected upon attempting to save.
  - **Implementation:** To provide users with real-time information about character limits, a Label was added next to each relevant input field to act as a character counter (recipeNameCount, recipeCategoryCount, etc.). JavaFX's property binding was utilized to dynamically update the text of these counter labels. The textProperty() of each TextField and TextArea was bound to the textProperty() of its corresponding counter label using Bindings.createStringBinding(). This binding calculates the current length of the input text and displays it alongside the maximum allowed characters (e.g., "12/18"). Helper methods like setupCharacterCounter() were introduced to streamline the creation and styling of these counter labels.
  - **Impact:** The addition of real-time character counters provides users with immediate and clear feedback on the input constraints, allowing them to manage their text entries effectively and avoid potential issues during PDF generation. This significantly improves the usability and intuitiveness of the application.

5. Alternate Project Concept and Rationale for Forgoing It

During the initial planning phase, the team explored several potential project ideas before finalizing the Recipe Vault application. One of the alternate concepts considered was a more advanced recipe management system that functioned like a searchable database. This version would have allowed users not only to store and view their recipes but also to edit existing entries, search recipes by ingredients or categories, and receive intelligent suggestions to enhance their cooking. We even discussed the possibility of integrating AI-driven features, such as recommending complementary ingredients or optimizing recipes based on dietary preferences.

While the idea was innovative and had the potential to greatly improve user experience, we ultimately decided to discard it due to the limited time frame and the complexity of implementation. Incorporating database functionality, AI algorithms, and dynamic search features would have significantly expanded the project’s scope beyond what was manageable within our timeline. Given our team size, skill sets, and the course objectives, we prioritized building a solid, well-designed application that reliably performs its core functions such as storing and generating professional recipe PDFs over adding more advanced, yet time-consuming, features.



Application Diagrams and Functions.

Recipe Vault — You can only add up to 25 steps. — OK

Recipe Vault — Ingredient 'milk' removed — OK

Recipe Vault — Select an ingredient to remove — OK

Recipe Vault — Instruction removed — OK

Recipe Vault — Select an instruction to remove — OK

Information — Recipe saved successfully! — OK

Recipe Vault — You can only add up to 15 ingredients. — OK

Recipe Vault — Ingredient already in the list or empty — OK

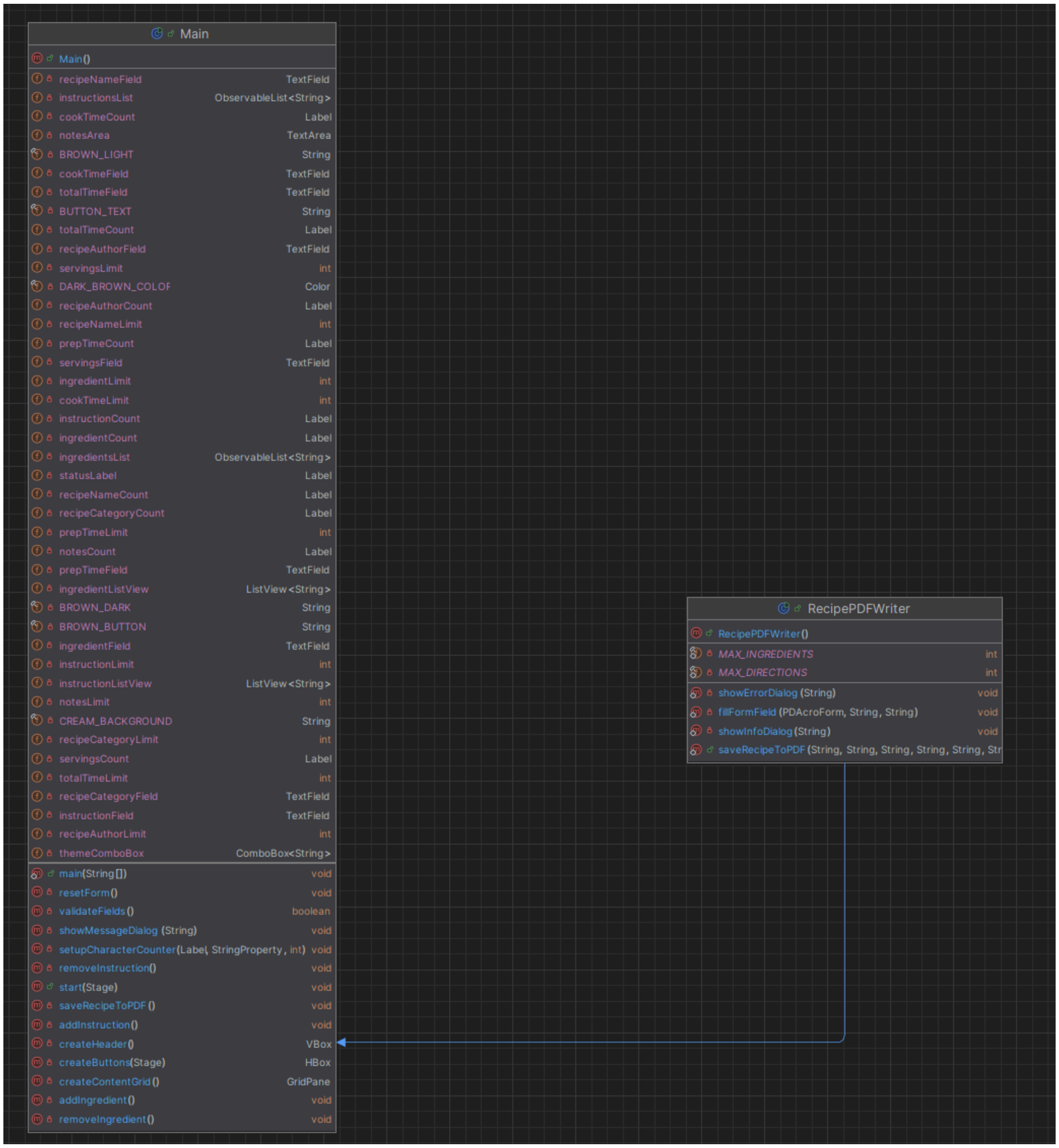
Recipe Vault — Instruction already in the list or empty — OK

Recipe Vault — The following fields cannot be empty: Recipe Name, Category/Cuisine, Recipe Author, Prep Time, Cook Time, Total Time, Servings, Theme, Ingredients List, Cooking Instructions — OK

Error — Template file not found: Summer_Template.pdf — OK

Figure 5 - Application diagram and functions.

UML Diagram



```
Main
Main()
recipeNameField              TextField
instructionsList             ObservableList<String>
cookTimeCount                Label
notesArea                    TextArea
BROWN_LIGHT                  String
cookTimeField                TextField
totalTimeField               TextField
BUTTON_TEXT                  String
totalTimeCount               Label
recipeAuthorField            TextField
servingsLimit                int
DARK_BROWN_COLOF             Color
recipeAuthorCount            Label
recipeNameLimit              int
prepTimeCount                Label
servingsField                TextField
ingredientLimit              int
cookTimeLimit                int
instructionCount             Label
ingredientCount              Label
ingredientsList              ObservableList<String>
statusLabel                  Label
recipeNameCount              Label
recipeCategoryCount          Label
prepTimeLimit                int
notesCount                   Label
prepTimeField                TextField
ingredientListView           ListView<String>
BROWN_DARK                   String
BROWN_BUTTON                 String
ingredientField              TextField
instructionLimit             int
instructionListView          ListView<String>
notesLimit                   int
CREAM_BACKGROUND             String
recipeCategoryLimit          int
servingsCount                Label
totalTimeLimit               int
recipeCategoryField          TextField
instructionField             TextField
recipeAuthorLimit            int
themeComboBox                ComboBox<String>
main(String[])               void
resetForm()                  void
validateFields()             boolean
showMessageDialog (String)   void
setupCharacterCounter(Label, StringProperty, int)  void
removeInstruction()          void
start(Stage)                 void
saveRecipeToPDF ()           void
addInstruction()             void
createHeader()               VBox
createButtons(Stage)         HBox
createContentGrid ()         GridPane
addIngredient()              void
removeIngredient()           void
```

```
RecipePDFWriter
RecipePDFWriter()
MAX_INGREDIENTS                                  int
MAX_DIRECTIONS                                   int
showErrorDialog (String)                         void
fillFormField (PDAcroForm, String, String)       void
showInfoDialog (String)                          void
saveRecipeToPDF (String, String, String, String, String, Str
```

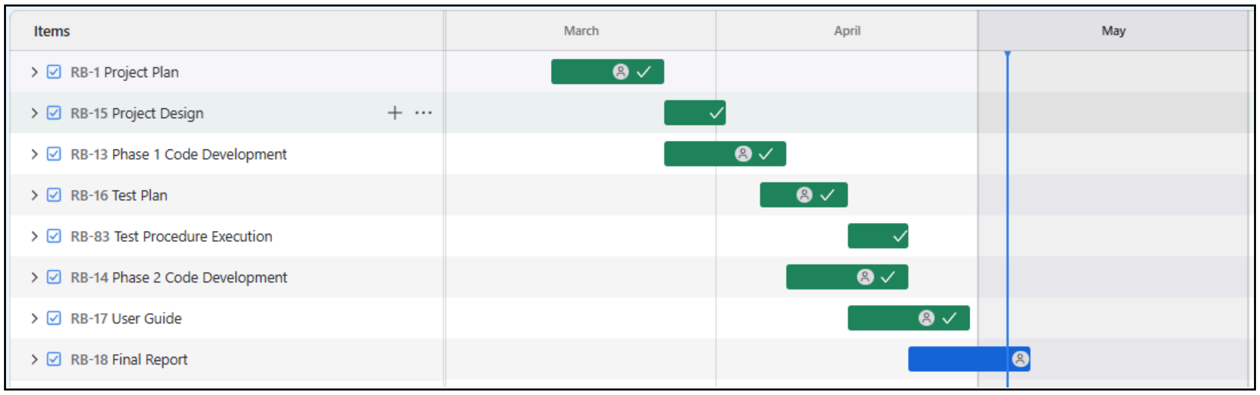
Figure 6 - RecipeVault UML diagram.

Development History

The development of the project followed a structured track leading from deliverable to deliverable. Each item had specific start and end dates and was split into subtasks with their own deadlines. The timeline followed closely to the assignment due dates for each of the required items, and was maintained utilizing a Jira page for both item tracking and assignment. Most project items followed a general timeline of; a Thursday meeting to discuss the required sections and establish a notes document, a Saturday meeting to go over the notes and assign each section to a group member, and a Monday or Tuesday meeting to review the final document before submission. Several of the items had start dates before the week of the assignment. During this time, members individually reviewed and prepared for the initial assignment kickoff meeting. Each of the schedule items are listed below:

- Project Plan
    - 13 March - 25 March
- Project Design
    - 26 March - 1 April
- Phase 1 Code Development
    - 26 March - 8 April
- Test Plan
    - 6 April - 15 April
- Test Procedure Execution
    - 16 April - 22 April
- Phase 2 Code Development
    - 9 April - 22 April
- User Guide
    - 13 April - 29 April
- Final Report
    - 23 April - 6 May

Figure 7 shows the schedule timeline with the overlapping areas designated during the planning process. Each of the sections prior to the current item was completed on time and contained all required information.



Items
RB-1 Project Plan
RB-15 Project Design
RB-13 Phase 1 Code Development
RB-16 Test Plan
RB-83 Test Procedure Execution
RB-14 Phase 2 Code Development
RB-17 User Guide
RB-18 Final Report

Figure 7 - Recipe Vault Project Schedule

Discussion

Throughout the development of the Recipe Vault project, one of the most valuable lessons learned was the importance of modular design and iterative testing. The separation of the UI, application logic, and PDF export layers allowed for easier debugging, clearer collaboration, and more manageable code updates. Additionally, the understanding of how small UI enhancements, such as the maximum character counter in the Recipe Vault, can make a significant impact on overall usability is very important. Equally important was the experience with using external libraries like Apache PDFBox, which showed how to integrate and manage third-party tools within a project effectively. A major strength of the Recipe Vault's design lies in its simplicity and clarity. The application is intuitive for all users and contains clearly labeled fields, input validation, and a straightforward PDF export process. The decision to avoid internal data storage reduced system complexity and resource demands, making it easier to deploy and use across various environments. Additionally, the visual feedback mechanisms, such as pop-up dialogs, help users avoid errors and complete the process efficiently.

The application also has a few limitations, such as the lack of support for recipe editing, which limits its long-term utility for users who may want to manage a recipe collection over time. Additionally, the static nature of the PDF templates means that any layout changes require manual editing of the source PDFs, rather than dynamically adjusting to user content. Lastly, the lack of automated testing in our development process meant more time was spent on manual UI testing, which can become inefficient as the codebase grows. For future improvement, the omitted internal recipe storage could be implemented to allow users to edit and organize their recipes with the application. The incorporation of automated tests for key logic and validation routines would improve development speed and reduce the risk of regressions. Finally, the expansion of theme customization or the provision of dynamic template previews could enhance the visual flexibility and personalization of the final exported recipes.

Teamwork also played a pivotal role in the successful development of the Recipe Vault application. By dividing responsibilities based on individual strength, such as interface design, backend logic, and PDF integration, each team member was able to focus on their area of expertise while maintaining clear communication and collaboration throughout the process. Regular check-ins and code reviews ensured that all components integrated smoothly and adhered to the project's overall design principles. This collaborative approach not only accelerated development but also led to creative problem-solving and a higher-quality final product. Ultimately, the shared vision and collective effort of the team transformed a simple idea into a fully functional and user-friendly application.

Conclusion

The completion of the Recipe Vault application represents the culmination of a well-coordinated team effort that blended technical skills, thoughtful design, and user-centered development. The team effectively translated the project's initial concept into a polished and functional system, meeting all original goals while responding to usability feedback along the way. Each member contributed meaningfully to different phases—from UI design and PDF generation to testing and documentation—resulting in a cohesive and maintainable final product.

One of the most valuable lessons learned was the importance of balancing ambition with usability. While the team originally considered incorporating editable PDFs and AI-driven features, these ideas were ultimately set aside in favor of a simpler, more intuitive design. This allowed the team to focus on core functionality, minimize complexity, and ensure a seamless user experience. The decision to simplify, rather than overextend, proved critical to the project's success.

The Recipe Vault project demonstrated excellence in multiple dimensions, including its thorough documentation, robust testing process, and strong team dynamic. By adopting a modular structure and integrating tools like Apache PDFBox, the team created a flexible system with room for future enhancements. The final product is accessible, aesthetically pleasing, and effectively fulfills its purpose of transforming personal recipes into professional, themed PDFs.

Though the current version lacks long-term storage or editing capabilities, it provides a solid foundation for future growth. Potential enhancements include internal recipe management, dynamic PDF templates, image attachment, and AI-based recipe suggestions. Ultimately, the team's shared vision and disciplined execution transformed a creative idea into a reliable, user-friendly application that blends the art of cooking with the power of technology.

References

Apache PDFBox. (2023). *Apache PDFBox – A Java PDF library*. The Apache Software Foundation. https://pdfbox.apache.org/

iText Software. (n.d.). *iText PDF library documentation*. https://itextpdf.com/en/resources/documentation

OpenJFX. (n.d.). *JavaFX documentation*. https://openjfx.io/

Oracle. (n.d.). *Java SE documentation*. Oracle. https://docs.oracle.com/en/java/

Stack Overflow. (n.d.). *How to use Apache PDFBox to fill a PDF form*. https://stackoverflow.com/questions/25004633/how-to-use-apache-pdfbox-to-fill-a-pdf-form

Appendix

All appendices for the project can be found at the following Google Drive link:
https://drive.google.com/drive/folders/1-iHYnJt_kpYe_00oEF979tqYXvlZm9hs?usp=drive_link

- Appendix A is the Project Plan
- Appendix B is the Project Design Document
- Appendix C is the Phase 1 delivery documentation
- Appendix D is the Test Plan Documentation
- Appendix E is the Test Case Spreadsheet
- Appendix F is the Phase 2 delivery documentation
- Appendix G is the test plan executed for the Phase 2 delivery
- Appendix H is the Recipe Vault User Guide
- Appendix I is the Javadoc created for the Recipe Vault Java files
- Appendix J is a diagram of the Recipe Vault functions
- Appendix K is the Recipe Vault UML class diagram