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?

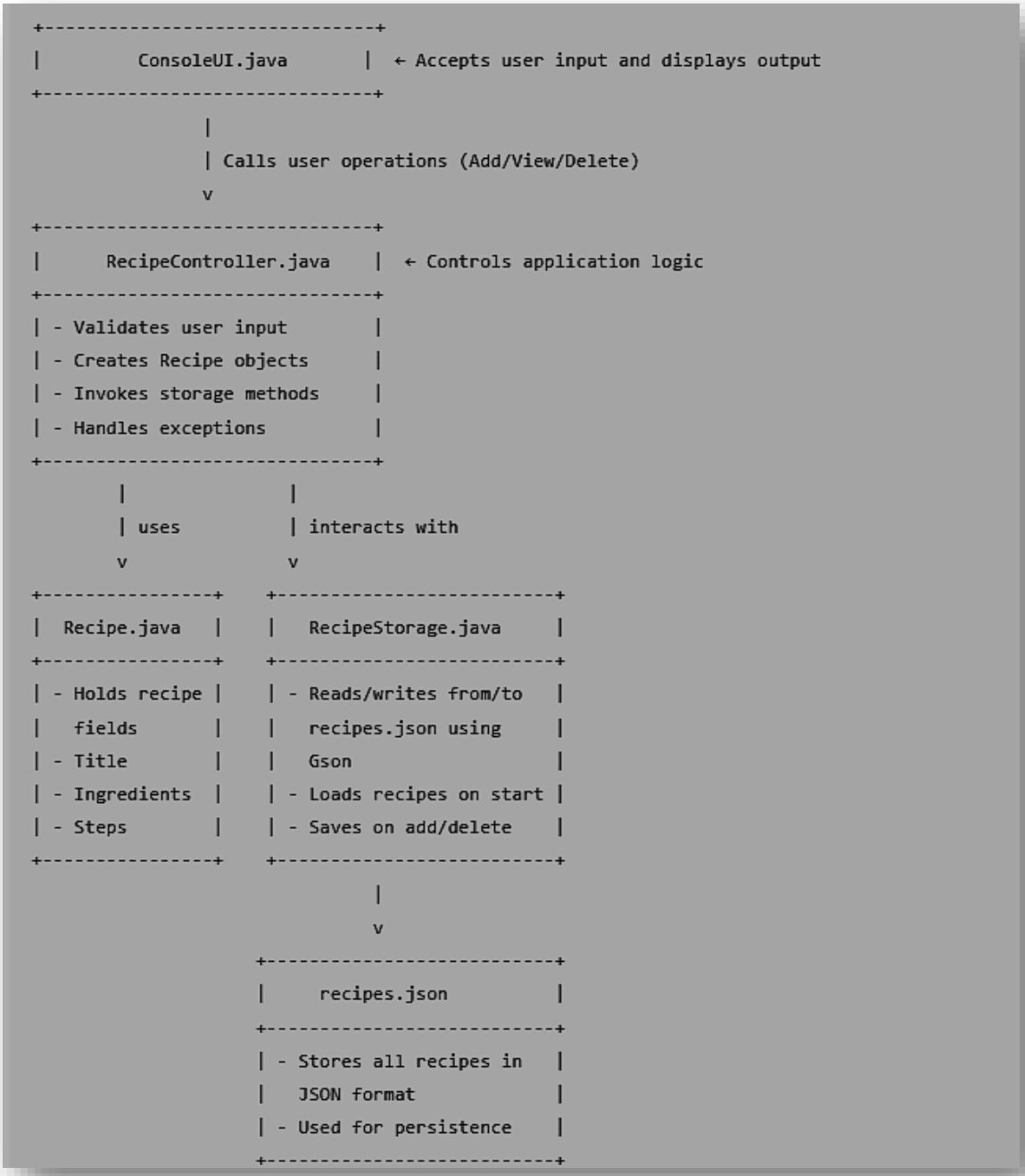
- Provides both Console & GUI options, meeting different user preferences.
- Improved test coverage using Boundary Value, Decision Table, State Transition, and Unit Testing.
- Fully offline and lightweight, requiring no external dependencies.
- Data security is higher due to local-only storage, with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

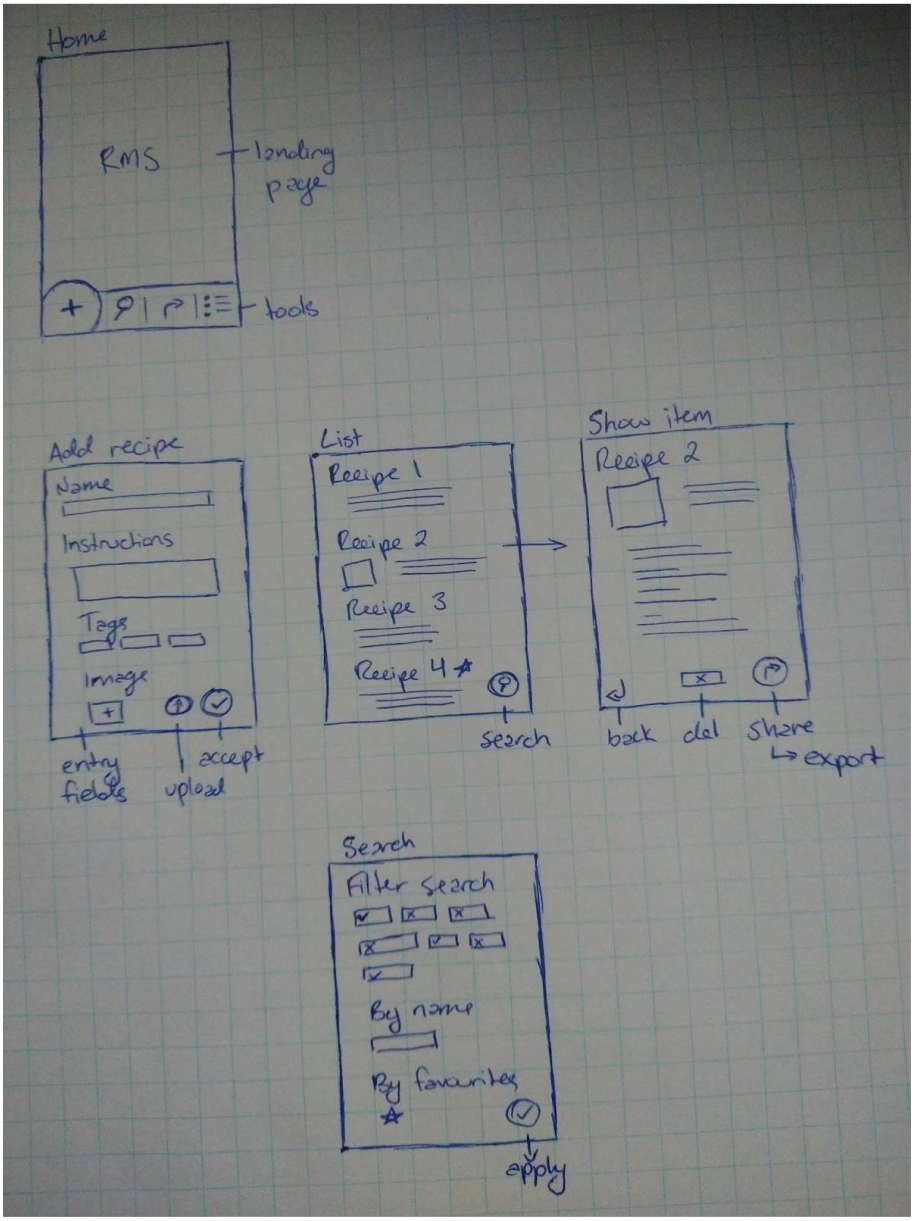
3.3.1 Components

- **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).


ConsoleUI.java
← Accepts user input and displays output
Calls user operations (Add/View/Delete)
RecipeController.java
← Controls application logic
- Validates user input
- Creates Recipe objects
- Invokes storage methods
- Handles exceptions
uses
interacts with
Recipe.java
- Holds recipe fields
- Title
- Ingredients
- Steps
RecipeStorage.java
- Reads/writes from/to recipes.json using Gson
- Loads recipes on start
- Saves on add/delete
recipes.json
- Stores all recipes in JSON format
- Used for persistence

Features:

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



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, allowing users to share their recipes with others. 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.

3.3.2 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.

3.3.3 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.

3.3.4 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.