Contributions to the Developer Guide

Given below is one of the sections that I contributed to the Developer Guide. They showcase my ability to write technical documentation and the technical depth of my contributions to the project.

Adding ingredients to inventory and cart

The inventory and cart acts as storage for `Ingredient` classes. They are facilitated by `InventoryCommand` and `CartCommand` respectively, which extends the `Command` abstract class. Since `CartAddCommand` and `InventoryAddCommand` both serve the same purpose in different contexts of `Cart` and `Inventory` respectively, they will be mentioned together in tandem.
The format of the commands are as follows:

- For cart: `cart add ingredient i/INGREDIENT_NAME q/INGREDIENT_QUANTITY`
- For inventory: `inventory add ingredient i/INGREDIENT_NAME q/INGREDIENT_QUANTITY`

Implementation

Below is a step-by-step sequence of what happens when the command `cart add ingredient i/INGREDIENT_NAME q/INGREDIENT_QUANTITY` is added.

1. The user adds a ingredient to the cart by entering the command `cart add ingredient i/INGREDIENT_NAME q/INGREDIENT_QUANTITY` in the command line input.
2. `CartAddCommandParser` parsers the input to check and verify that the input provided for `i/INGREDIENT_NAME` amd `q/INGREDIENT_QUANTITY` are correct. Otherwise a `ParseException` will be thrown.
3. The fields are then passed to `CartAddIngredientCommand` as an `Ingredient` object and is returned to `LogicManager`.
4. `LogicManager` calls `CartAddIngredientCommand#execute()` and checks if the `Ingredient` object given has the same `INGREDIENT_NAME` and `INGREDIENT_QUANTITY` unit. If that `Ingredient` exists, it will simply add on to the quantity of that ingredient. Otherwise, a new instance of that `Ingredient` will be added to the Cart.
5. If `CommandException` is not thrown, `Model#addCartIngredient` will be executed, with the given `Ingredient` as the parameter
6. `Model#addCartIngredient` then executes, adding the `Ingredient` to the local cart storage and updates with `Model#updateFilteredCartIngredientList()`.
7. A `CommandResult` with the successful text message is returned to `LogicManager` and will be displayed to the user via the GUI to feedback to the user that the `Ingredient` has been successfully added.

The above implementation is the same for `Inventory` with the command `inventory add ingredient i/INGREDIENT_NAME q/INGREDIENT_QUANTITY`

Implementation reasoning

This command was implemented to allow the user know to add an ingredient to the cart or inventory respectively. An ingredient only has two main components - its name and quantity. We allow the user to use their own measurement up to their own preferences and do not force any fixed unit of measurement. Although similar, `Cart` and `Ingredients` differ in certain functions from a user's point of view. For a user to immediately sort where they wish to sort the ingredient they are adding, `Cart` and `Inventory` is the first parameter they would use for the command.

Sequence diagram

The following sequence diagram shows how the function of adding ingredients to cart work (full command omitted for brevity):

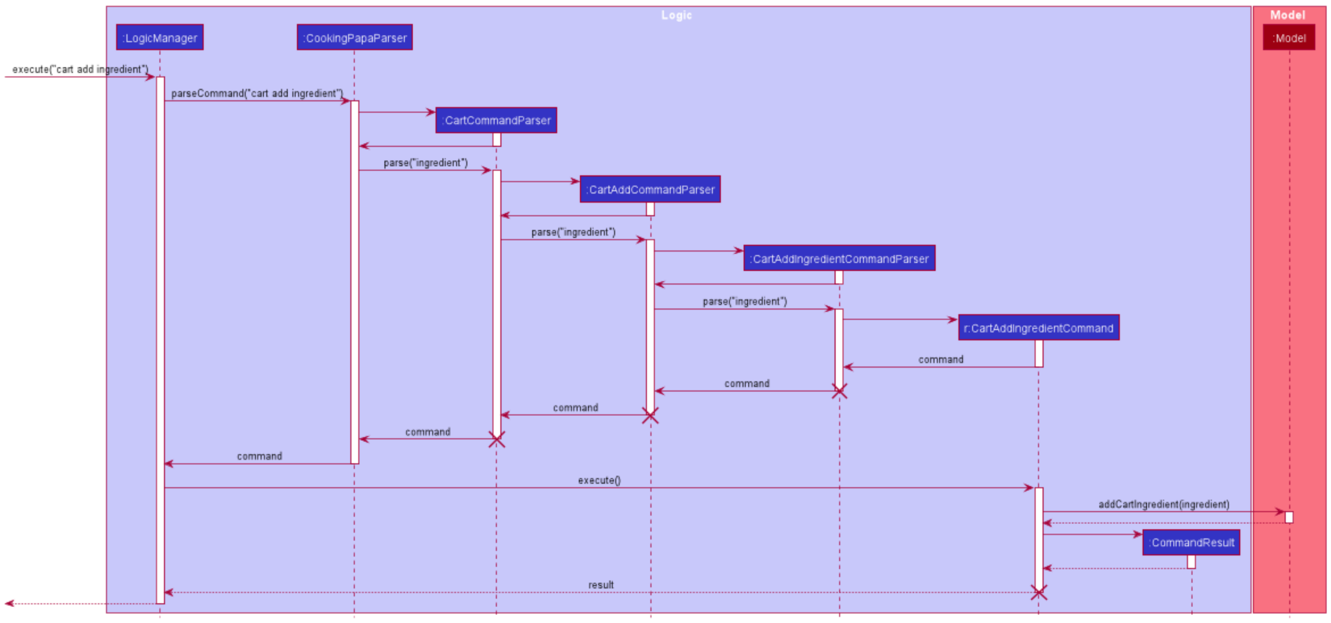
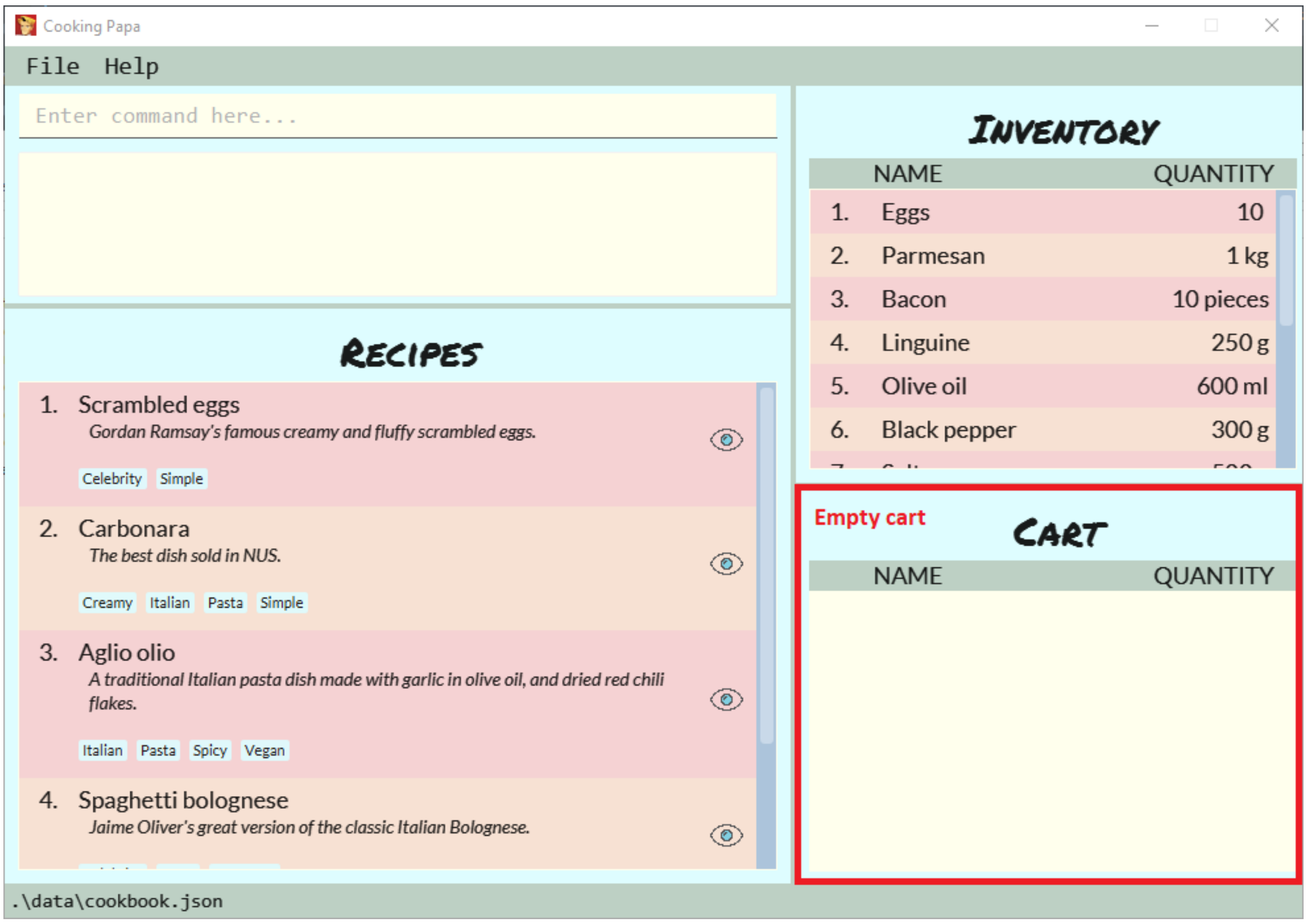


Figure 1. Sequence diagram for CartAddIngredientCommand

How the feature works

Step 1: The below diagram shows an initially empty cart



INVENTORY

	NAME	QUANTITY
1.	Eggs	10
2.	Parmesan	1 kg
3.	Bacon	10 pieces
4.	Linguine	250 g
5.	Olive oil	600 ml
6.	Black pepper	300 g

RECIPES

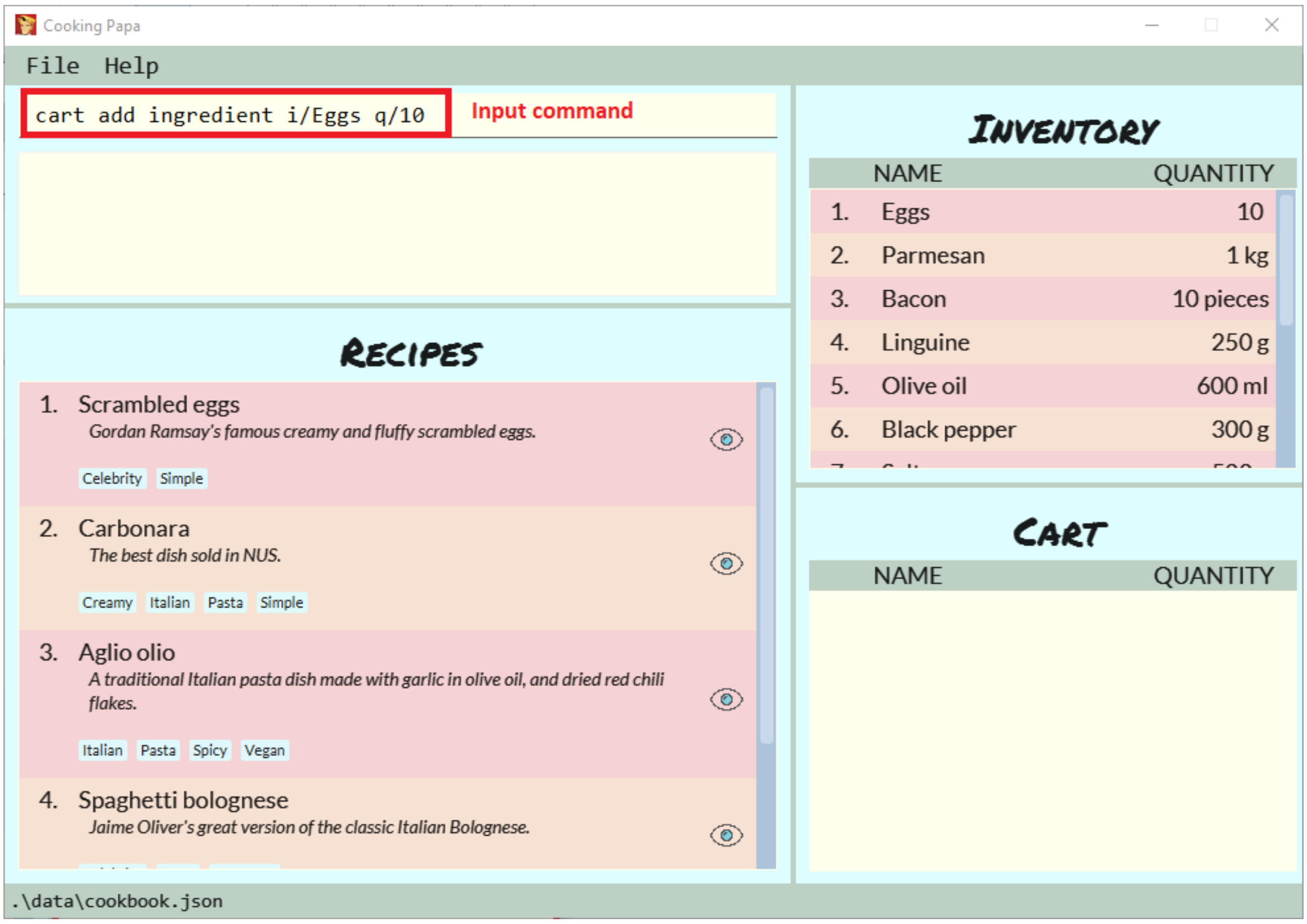
1. Scrambled eggs
   Gordan Ramsay's famous creamy and fluffy scrambled eggs.
2. Carbonara
   The best dish sold in NUS.
3. Aglio olio
   A traditional Italian pasta dish made with garlic in olive oil, and dried red chili flakes.
4. Spaghetti bolognese
   Jaime Oliver's great version of the classic Italian Bolognese.

CART

NAME	QUANTITY

```
.\data\cookbook.json
```

Step 2: Input the command `cart add ingredient i/INGREDIENT_NAME q/INGREDIENT_QUANTITY`. An example command is as follows: `cart add ingredient i/Eggs q/10`.



```
cart add ingredient i/Eggs q/10
```

INVENTORY

	NAME	QUANTITY
1.	Eggs	10
2.	Parmesan	1 kg
3.	Bacon	10 pieces
4.	Linguine	250 g
5.	Olive oil	600 ml
6.	Black pepper	300 g

RECIPES

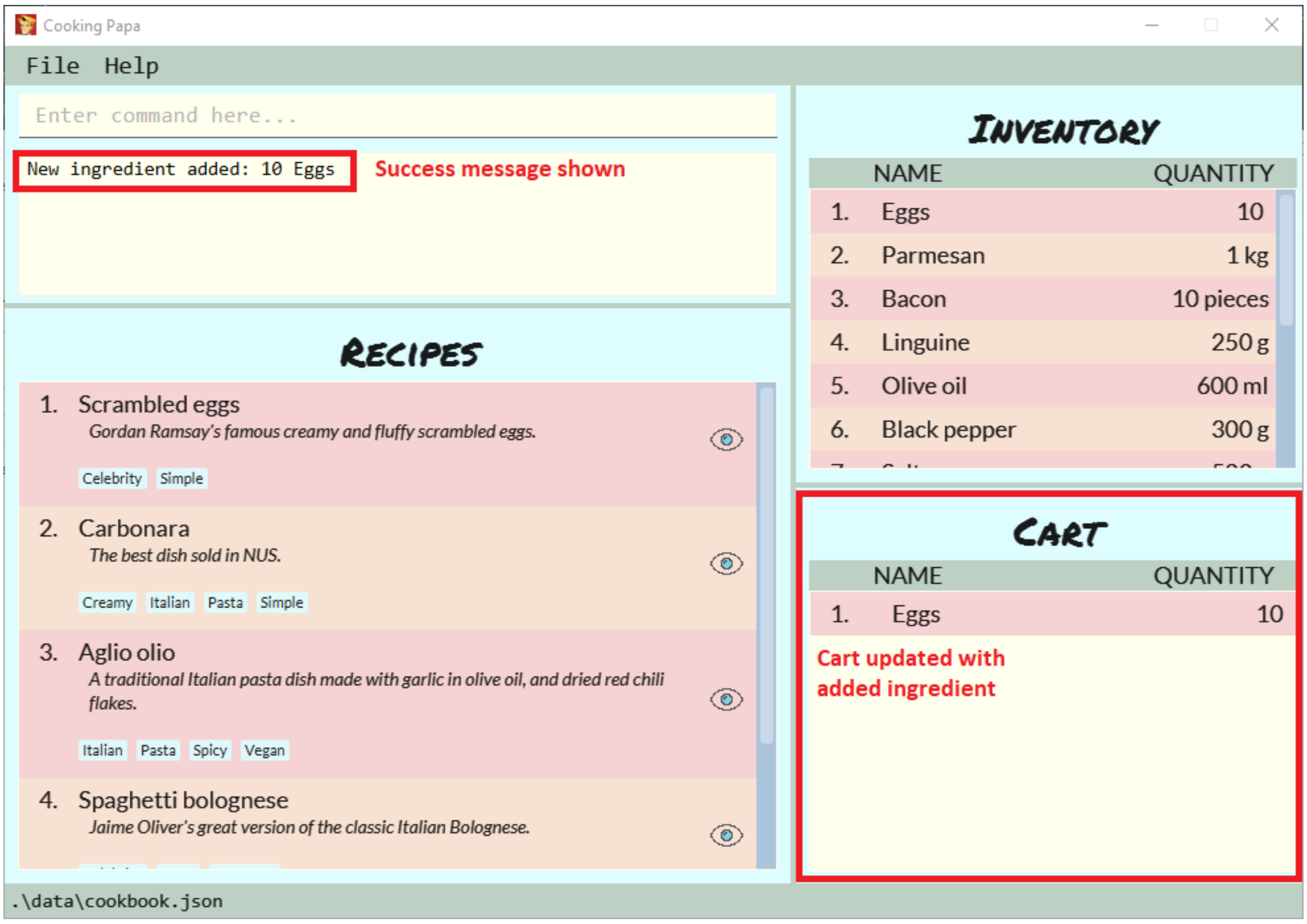
1. Scrambled eggs
   Gordan Ramsay's famous creamy and fluffy scrambled eggs.
2. Carbonara
   The best dish sold in NUS.
3. Aglio olio
   A traditional Italian pasta dish made with garlic in olive oil, and dried red chili flakes.
4. Spaghetti bolognese
   Jaime Oliver's great version of the classic Italian Bolognese.

CART

NAME	QUANTITY

```
.\data\cookbook.json
```

Step 3: Hit [Enter]. The cart should be updated as follows:



```
New ingredient added: 10 Eggs
```

NAME	QUANTITY
1. Eggs	10
2. Parmesan	1 kg
3. Bacon	10 pieces
4. Linguine	250 g
5. Olive oil	600 ml
6. Black pepper	300 g

1. Scrambled eggs
   Gordan Ramsay's famous creamy and fluffy scrambled eggs.
   Celebrity Simple
2. Carbonara
   The best dish sold in NUS.
   Creamy Italian Pasta Simple
3. Aglio olio
   A traditional Italian pasta dish made with garlic in olive oil, and dried red chili flakes.
   Italian Pasta Spicy Vegan
4. Spaghetti bolognese
   Jaime Oliver's great version of the classic Italian Bolognese.

NAME	QUANTITY
1. Eggs	10

The above implementation is the same for `Inventory`

Design Considerations

Aspect: The need for many parsers for this command

Table 1. Design considerations for the need for many parsers for this command

	Design A (Current choice): Create parsers for every individual action	**Design B: Create parsers for each specific action**
Description	The command will go through the parsers in the following order: `CookingPapaParser` → `CartCommandParser` → `CartAddCommandParser` → `CartAddIngredientParser` before finally returning `CartAddIngredientCommand`. We eventually went with this as we wanted the add functionality to be expanded, namely to be able to add all the ingredients of cookbook recipes into the cart.	`CartAddCommand` will not be created facilitate `CartAddIngredientCommand` and `CartAddRecipeIngredientCommand`.
Pros	More organised and can do more with `cart add` as the prefix.	Many parser classes to make and keep track of.

	Design A (Current choice): Create parsers for every individual action	**Design B: Create parsers for each specific action**
Cons	The classes can be more specific and atomic in their actions.	Might lead to disorganisation during troubleshooting with so many classes to keep track.

Moving ingredients from cart to inventory

The user may use this command after their shopping trip. With this one command, all ingredients will be shifted from the cart to the inventory.

Implementation

This command is facilitated by `CartMoveCommand`, which extends the `Command` class. The format of the command is as follows: `cart move`.

Below is a step by step sequence of what happens when the user executes this command.

1. The user enters the command `cart move` in to the command line input.
2. `CartMoveCommandParser` then ensures that the user does not enter any other commands after `cart clear`.
3. `CartMoveCommandParser` then returns a `CartMoveCommand` and returns it to `LogicManager`
4. `LogicManager` calls `CartMoveCommand#execute()`. If there are other commands after `cart clear`, a `CommandException` will be thrown.
5. If `CommandException` is not thrown, `Model#cartMoveIngredients()` will be executed.
6. `Model#cartMoveIngredients()` will move every ingredient from the `cart` and add it into the `inventory`
7. A `CommandResult` with the success message text will be returned to `LogicManager`, which will then be passed to `MainWindow` and will then feedback to the user.

Implementation reasoning

This command is implemented to ease the process of having the user adding every single ingredient to their inventory after they have bought ingredients from their cart and eventually deleting the cart after that tedious process. These gives a convenience to users that frequently use our application and we forsee that such an action will be used very often by these users. As this command only performs an atomic action, no extra arguments are needed to further supplement the use of this command.

Sequence Diagram

The following sequence diagram shows how this function works (full command omitted for brevity):

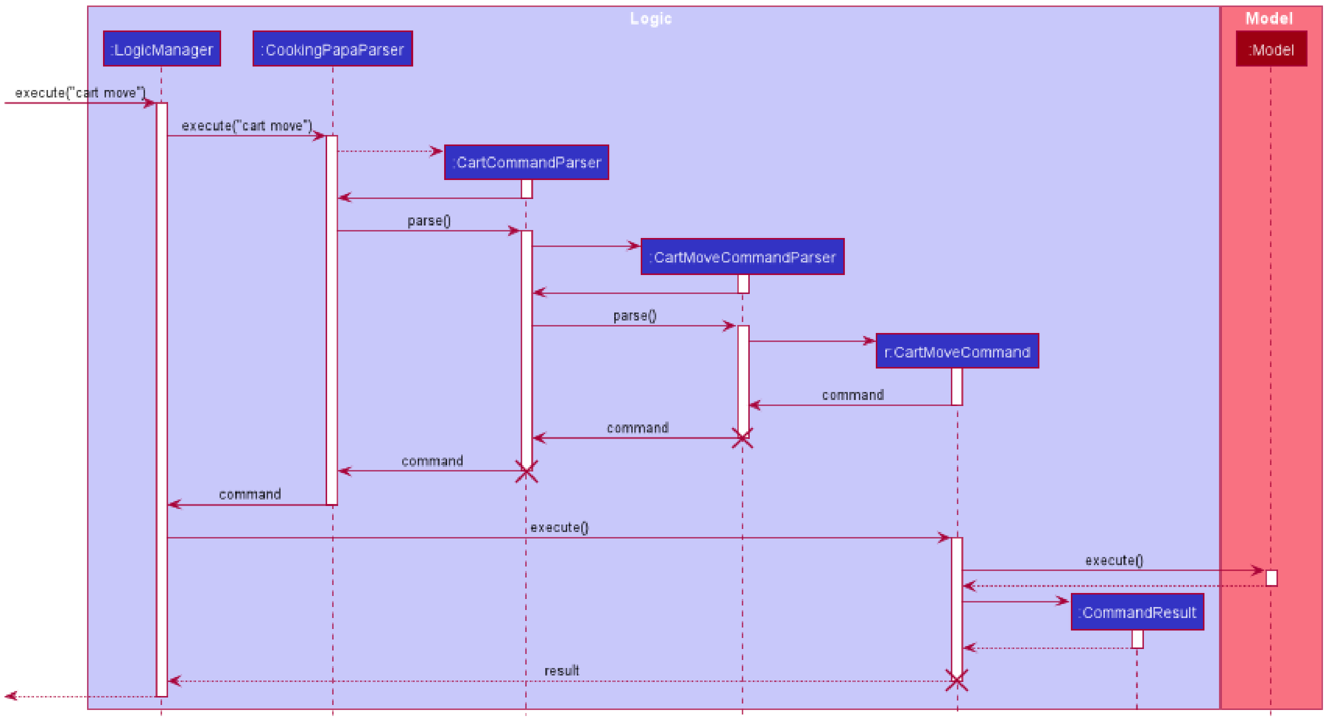
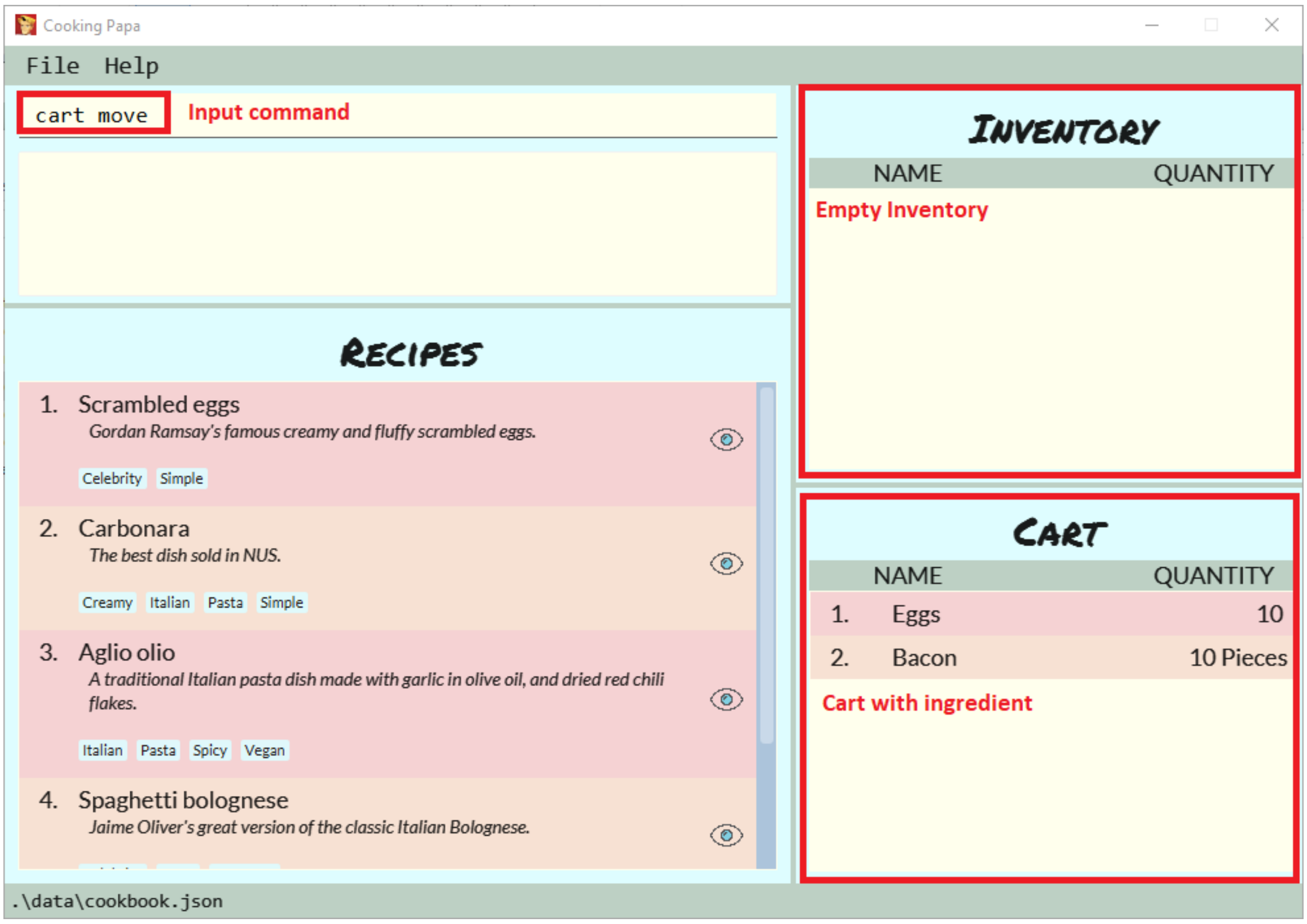


Figure 2. Sequence Diagram for CartMoveCommand

How this feature works

Step 1: This feature is intended when you have ingredients in the cart. As an example, the diagram below shows an empty inventory, along with a cart with an ingredient.



Recipes:

1. Scrambled eggs
   Gordan Ramsay's famous creamy and fluffy scrambled eggs.
   Celebrity Simple
2. Carbonara
   The best dish sold in NUS.
   Creamy Italian Pasta Simple
3. Aglio olio
   A traditional Italian pasta dish made with garlic in olive oil, and dried red chili flakes.
   Italian Pasta Spicy Vegan
4. Spaghetti bolognese
   Jaime Oliver's great version of the classic Italian Bolognese.

Inventory:

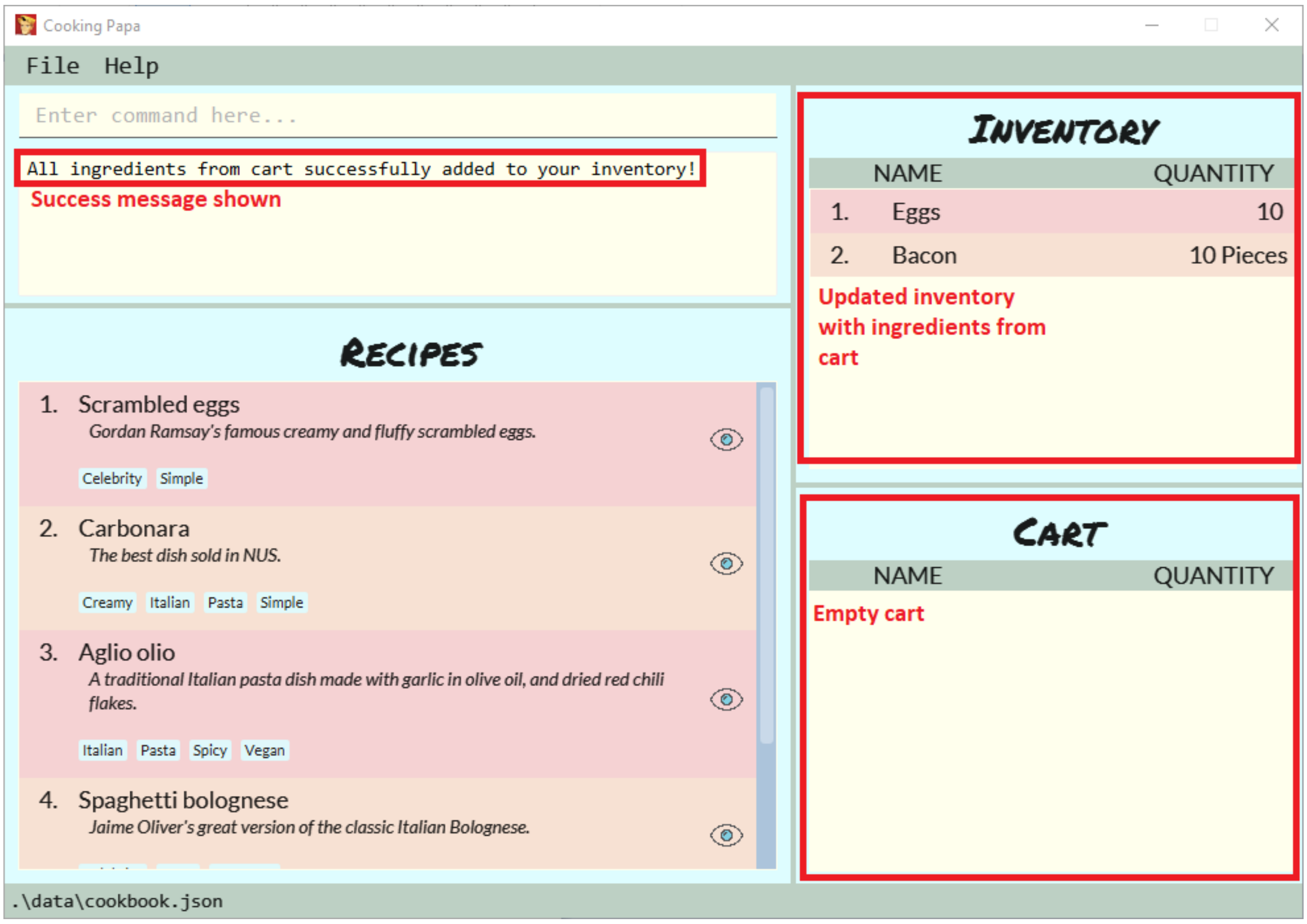
NAME	QUANTITY

Cart:

	NAME	QUANTITY
1.	Eggs	10
2.	Bacon	10 Pieces

Step 2: Press [Enter] The ingredients from cart will all be shifted to inventory as shown in the

diagram below



All ingredients from cart successfully added to your inventory!

Success message shown

NAME		QUANTITY
1.	Eggs	10
2.	Bacon	10 Pieces

Updated inventory with ingredients from cart

1. Scrambled eggs
   Gordan Ramsay's famous creamy and fluffy scrambled eggs.
   Celebrity Simple
2. Carbonara
   The best dish sold in NUS.
   Creamy Italian Pasta Simple
3. Aglio olio
   A traditional Italian pasta dish made with garlic in olive oil, and dried red chili flakes.
   Italian Pasta Spicy Vegan
4. Spaghetti bolognese
   Jaime Oliver's great version of the classic Italian Bolognese.

NAME	QUANTITY

Empty cart

.\data\cookbook.json

Design considerations

Aspect: Allowing users to move some or all ingredients from cart to inventory

Table 2. Design considerations for allowing users to move some or all ingredients from cart to inventory

	Design A (Current choice): Move all ingredients	**Design B: Allow users to move individually or exclude some ingredients when moving**
Descriptio n	There was a consideration to allow the user to move the ingredients by individual ingredients. Eventually the options was not given as we know that typical users will want to move all the ingredients except for individual ingredients.	The use cases of such an action happening is very little and the user can simply manually remove the few ingredients they do not wish to add to the inventory after using the `cart move` command. The user can also manually add back the ingredients to the cart after it is cleared if they wish to.
Pros	Straightforward to implement	Lesser implementations, more time to focus on other parts of the project
Cons	Lesser functionality to users that really want to only move certain ingredients	Poorer user experience for users that do not want to move all ingredients from the cart to inventory on a regular basis,