Maurice Tan - Project Portfolio

PROJECT: CaloFit

Overview

My team of 4 software engineering student and I were tasked with enhancing a basic command line interface desktop addressbook application(AddressBook – Level 3) for our Software Engineering project. We chose to morph it into a calories tracker management system called CaloFit. This enhanced application provides health-conscious people or those who are aiming for a diet to set their calorie budget for the day or to simply input their meal for the day; a calorie budget tracker that manages the meal that they take; find dishes based on keywords; suggest a meal with the remaining calorie budget; notify the user if the user missed a meal; and get data about their calorie intake progress through a report.

CaloFit is a desktop application for tracking the calories that the user has taken from his or her meals over the course of using the application.
The user interacts with CaloFit using a Command Line Interface(CLI) that is represented by a box near the top of the application screen. This is where the user can type in their commands and press "Enter" on their keyboards to execute them.
It has a Graphical User Interface(GUI) created with JavaFX. The GUI is the main display that the user sees upon starting up CaloFit.

This is what our project looks like as shown in Figure 1 below:

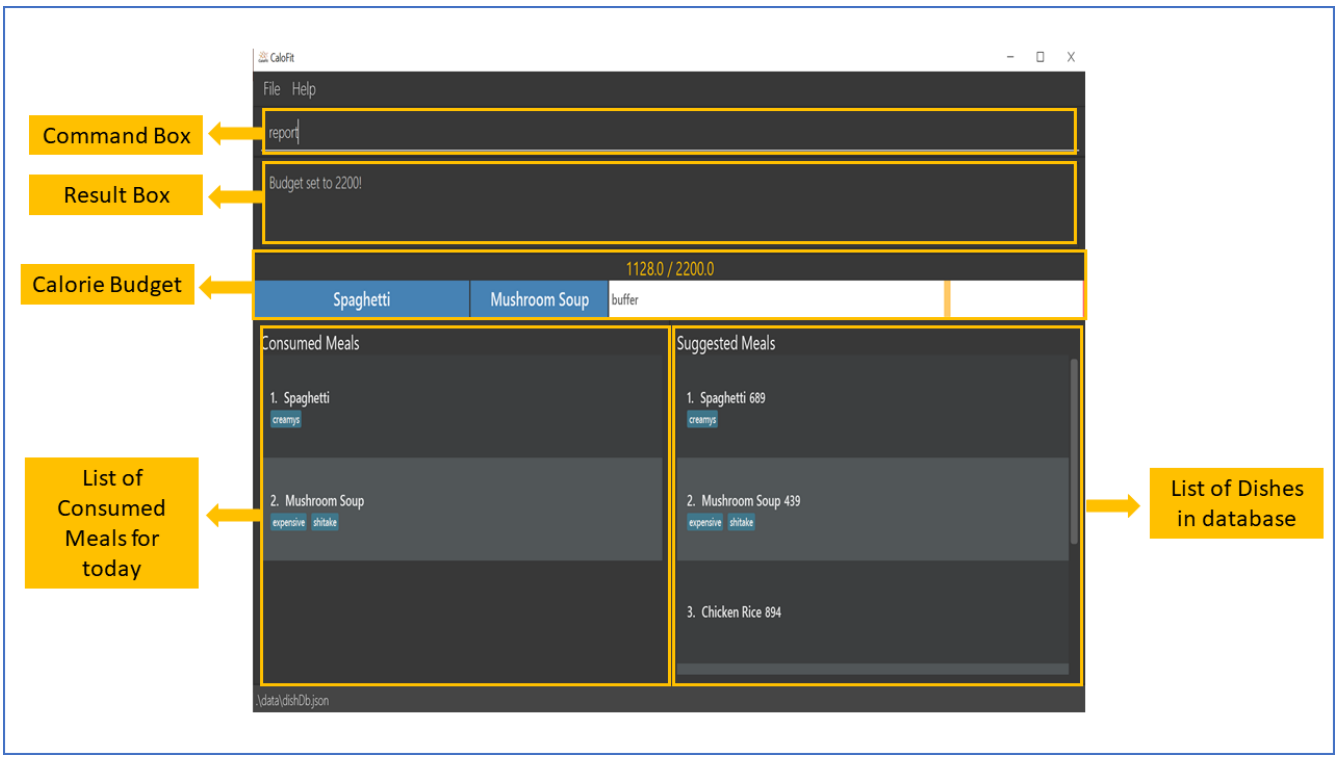


Figure 1. The graphical user interface for CaloFit.

My role was to design and write the codes for the `suggest` and `notification` features. The following sections illustrate these enhancements in more detail, as well as the relevant documentation I have added to the user and developer guides in relation to these enhancements.

Note the following symbols and formatting used in this document:

NOTE	This symbol indicates important information.

`suggest` A grey highlight (called a mark-up) indicates that this is a command that can be inputted into the command line and executed by the application.

Summary of contributions

This section shows a summary of my coding, documentation, and other helpful contributions to the team project.

- **Major enhancement 1**: added **the ability to suggest meals**
  - What it does: automatically provides the user with suggested meals based on the remaining calorie budget. Ability to toggled between finding meal and suggested meals.
  - Justification: This feature improves the product significantly because as a user most of the time we would not know what to eat, and a list of suggested meal will help us the user to decide on their meal quicker. Moreover, the suggested meals are all within the calorie budget, hence the user would not need to worry about having a meal that will be over their current calorie budget.
  - Highlights: This enhancement affects the future suggestion for a whole day meal plan or exercise plan that would be added in future. This feature provides the user with convenience of choosing a meal, and could potentially be the income generating factor for CaloFit.
- **Major enhancement 2**: added **the ability to notify user when a meal is missed**
  - What it does: the user will be prompted with a notification when a meal is missed.
  - Justification: On top of keeping within the calorie budget range, this feature also ensure that the user follows a spread-out and balance diet, of eating their meal at allocated timing. Furthermore, it also ensures that the user will not miss their meal throughout their busy day schedule.
  - Highlights: This enhancement affects the future implementation of notification being set by the user and users being allowed to decide on what they want to be notified about. This feature provides the user with a worry-free experience, as they need not worry about forgetting their meal in future.
- **Code contributed**: [Functional code]
- **Other contributions**:
  - Reported bugs and suggestions for other teams in the class (List of bugs reported)

Contributions to the User Guide

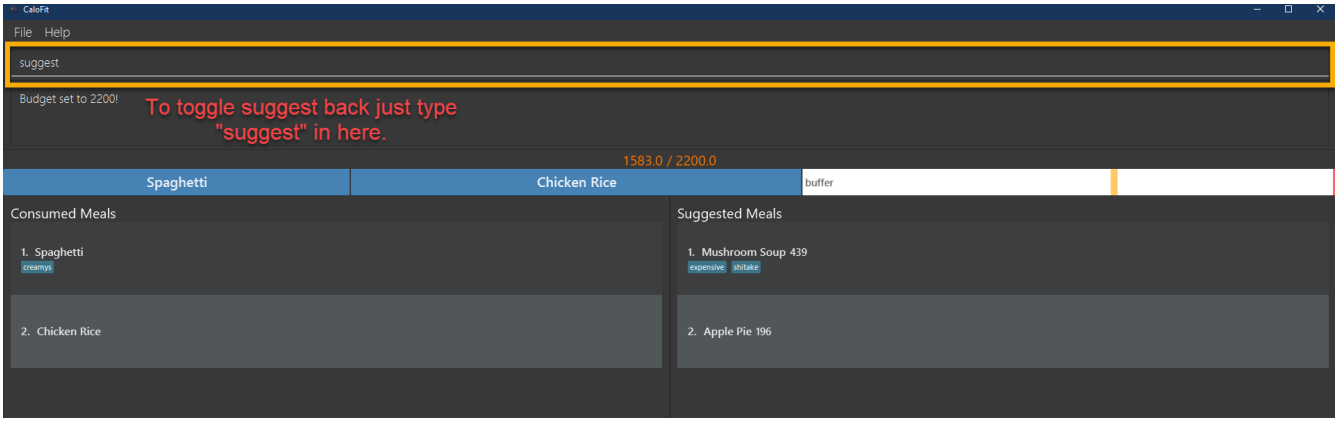
*Given below are sections I contributed to the User Guide as we had to update the original AddressBook-Level 3 User Guide with instructions for the enhancements that we had added. The following is an excerpt from our **CaloFit User Guide**, showing additions that I have made for the `suggest` and `notification` feature. They showcase my ability to write documentation targeting end-users.*

Suggesting Meal options: `suggest`

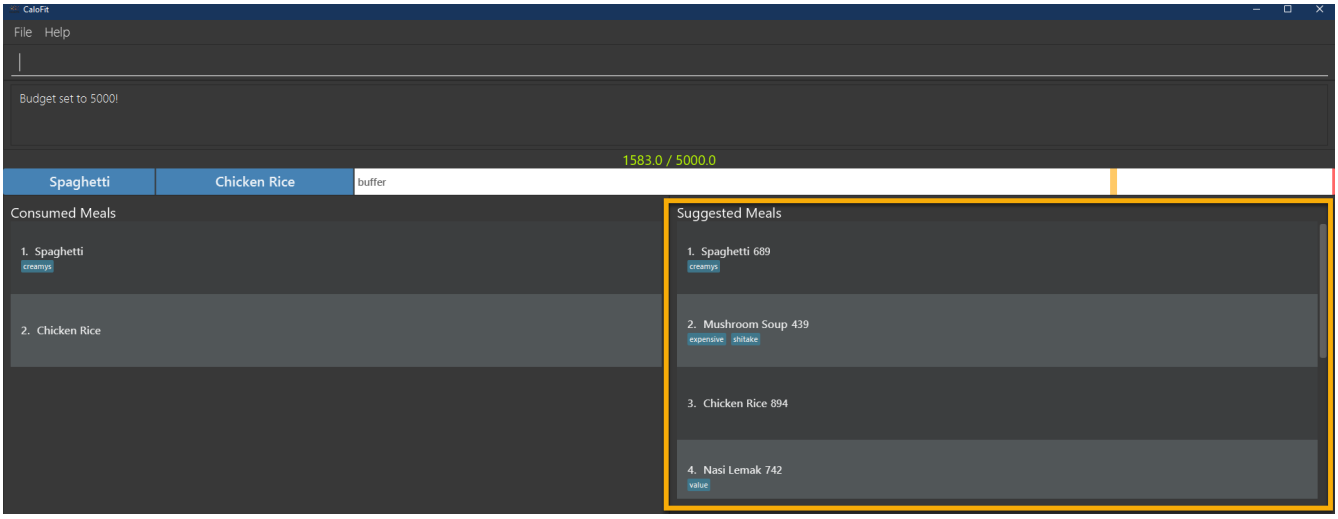
Suggest possible meals based on user calorie intake budget.
The available meals will be shown under the "Suggested Meals" section of the application.
The suggest feature is automatically toggled when the application starts, however if you use the find feature which replaces the suggest feature, then the suggest feature can be toggled back by typing "suggest" into the command box.

Example usage: You're tired from work and you don't want to think about what to eat for the day, you just want to have a meal to keep yourself full and stay within the calorie budget that you want set for yourself.

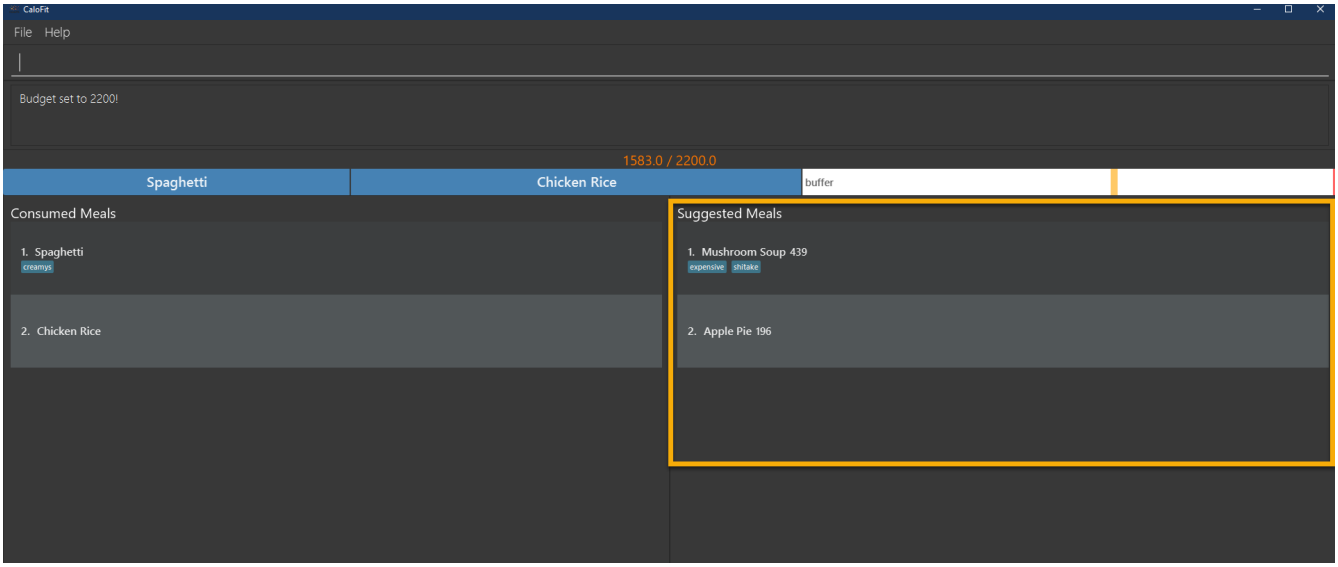
NOTE Type "suggest" into the command bar and press the 'enter' key, if you have previously used the find feature to look for a meal.



- The budget was set to a huge budget to show that there are actually a list of food inside as shown in the image below.



- The budget was then set to an average male budget of 2200, which then could be seen in the image that only those within the budget was shown.



Format: `suggest`

Notification that a meal is missed

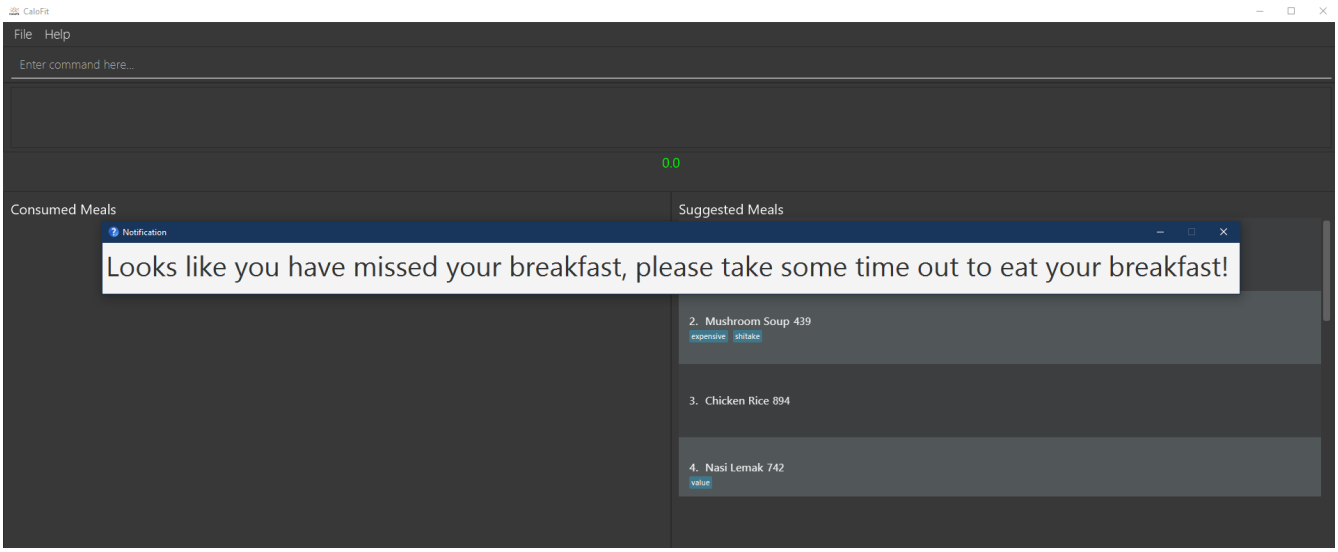
A notification will automatically be prompted when the application starts up or every 10 minutes if a meal is missed.
This feature cannot be disabled and will start once the application starts.

NOTE	The first meal will always be breakfast, hence no matter at which timing, if the first meal is not consumed, it will always notify the user as breakfast missed. If lunch have not been consumed after 8pm, notification will change and notify user that they have not consumed their dinner instead of lunch, as consuming two meals at once is not healthy.

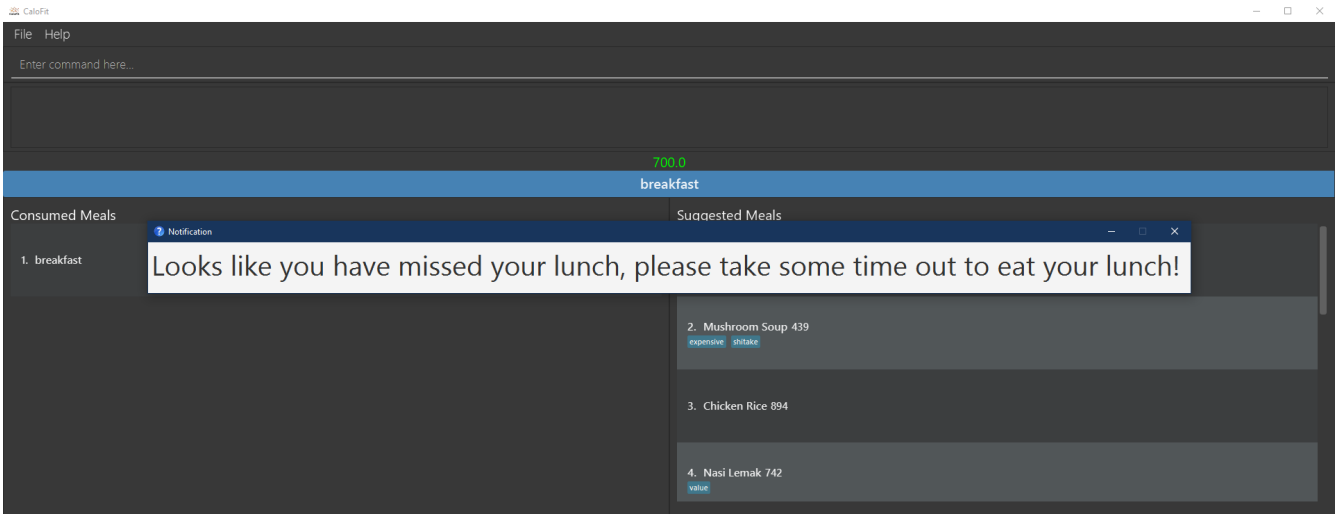
NOTE	Notification will be prompted: - After 10am for breakfast - After 2pm for lunch - After 8pm for dinner

NOTE	Meals that will be counted: - As breakfast - anytime - As lunch - after 11am - As dinner - after 4pm

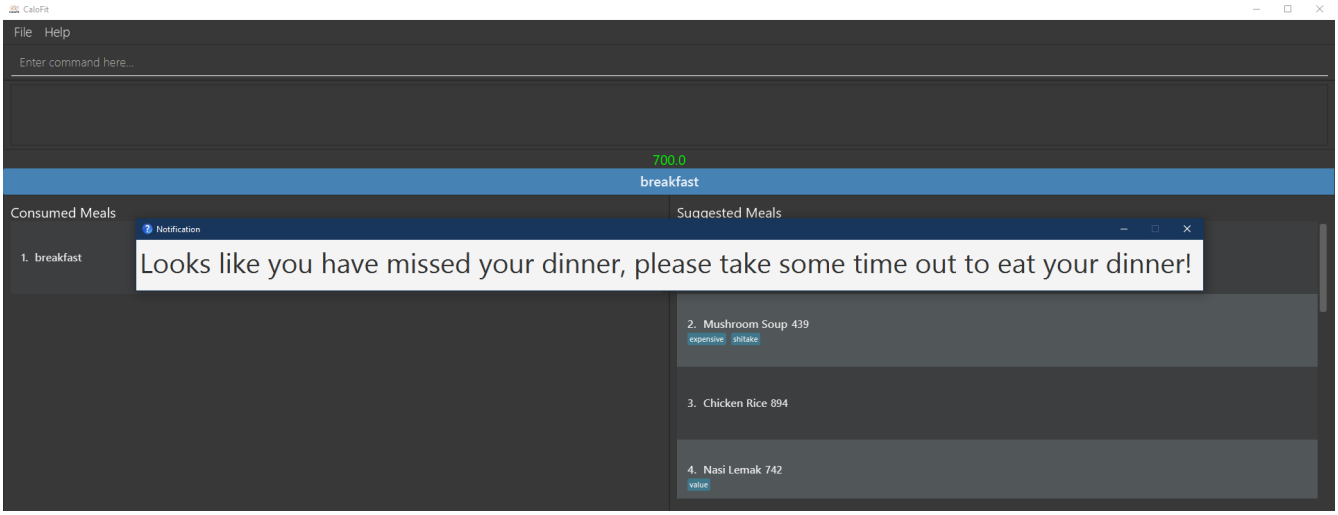
- If breakfast is missed this will be how it looks like



- If lunch is missed this will be how it looks like



- If dinner is missed this will be how it looks like



Contributions to the Developer Guide

Given below are sections I contributed to the Developer Guide for the `suggest` and `notification` feature. They showcase my ability to write technical documentation and the technical depth of my contributions to the project.

Suggest feature

The suggest feature displays the suggested automatically to the user accordingly to the remaining calorie budget.

Implementation

The suggest mechanism is toggled when the application starts up. It will always display the suggested meals for the user in the right pane. The feature can be can be toggled back by typing the "suggest" command. Through a `SuggestCommand` that extends from the abstract `Command` class. It interacts with the object that implements the `Model` interface by updating the observable list with dishes that are within the calories budget. The calories budget is obtained from `Model#getRemainingCalories()`.

Additionally, it implements the following operation:

- `Model#getRemainingCalories` — gets the current allowed calories budget.

This operation is exposed in the `Model` interface as `Model#getRemainingCalories()`.

Given below is an example usage scenario and how the suggest mechanism behaves at each step.

Step 1. The user starts up CaloFit for the first time. The `DishDatabase` will be loaded with the initial data by calling `MainApp#loadDishDatabase`.

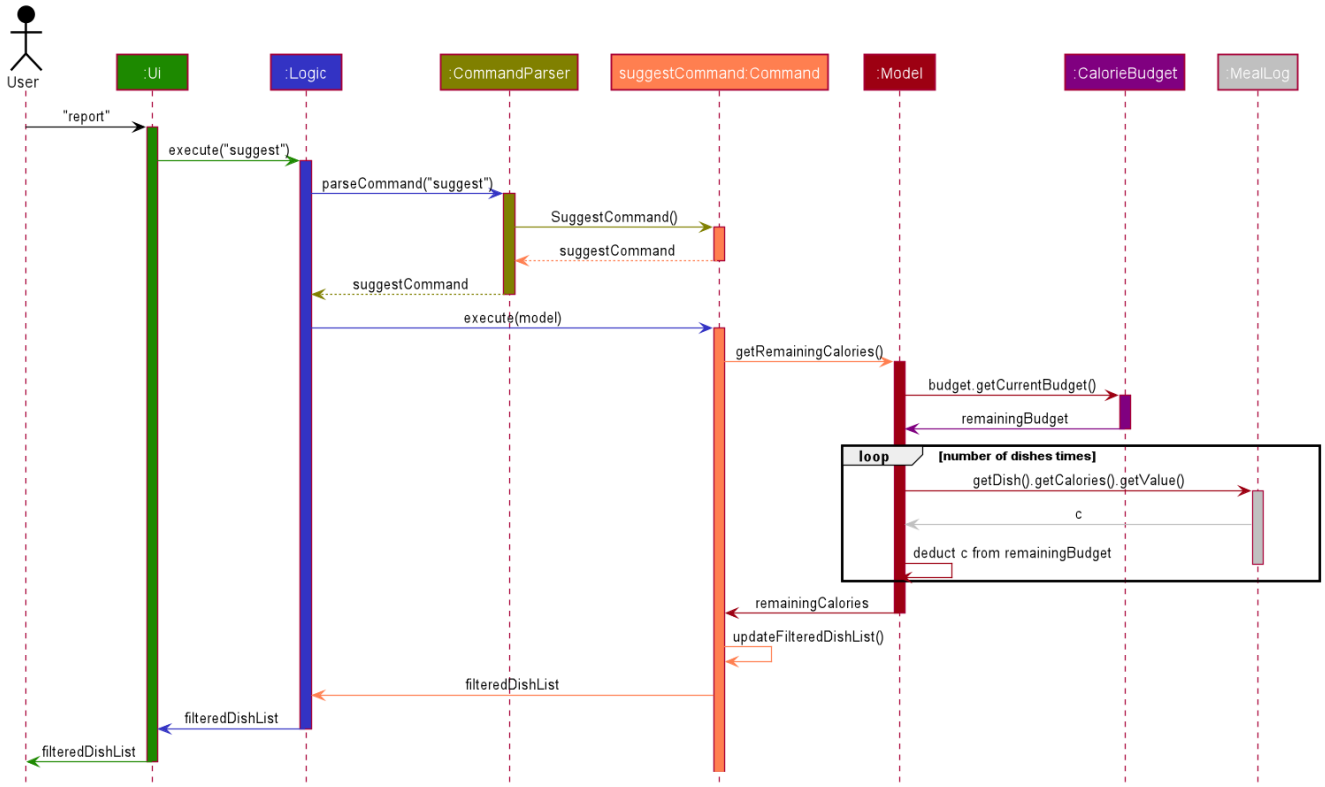
NOTE	If the DishDatabase is empty, or the daily calorie budget have exceeded the set amount, or there are no dishes that is within the amount, nothing will be display.

Step 2. The user enters "suggest" in the Command Line Input to invoke the `SuggestCommand` command

which will run `SuggestCommand#execute()`. `Model#getRemainingCalories()` will be executed to get the remaining calories, which will provide `#Model#updateFilteredDishList` with the calories budget to update the list accordingly.

Step 3. The success message will be returns, while printing the updated dish list on CaloFit right pane.

The following sequence diagram shows how the suggest operation works:



Design Considerations

Aspect: How suggest executes

- **Alternative 1 (current choice):** Update dish list with calorie budget
  - Pros: Neater code, easier to maintain and uses less memory.
  - Cons: Unable to get history of suggest list.
- **Alternative 2:** Create a new list for every `SuggestCommand`.
  - Pros: Easier to understand and customise if require data for method.
  - Cons: Unnecessary memory wastage for list created and not used.
- **Explanation of Choice:** Since we only need to show the user the current meal suggest, there are no usage for the past suggest result.

Aspect: Data structure to support the suggest command

- **Alternative 1 (current choice):** Stores the value in a dish list.
  - Pros: Commonly used, thus easier to understand and easier to deal with. It can also be easily updated.

    - Cons: Would constantly require `Model` with a responsibility that is not relevant to its current.
- **Alternative 2:** Wrap the values in a `Suggest` object
    - Pros: Neater and easier to maintain, since all suggest-related values are stored in the `Suggest` object.
    - Cons: Additional class to maintain, harder for newcomers to understand code with too many classes.

Notification feature

The notification feature prompts the user with new window if a meal had been missed.

Implementation

The notification feature is automatically activated when the application starts up. On start up, it is implemented through a `NotificationHelper` that gets information from `Model#getMealLog().getTodayMeals()` method to check if there are any meal consumed and if a meal had been consumed. The `NotificationHelper` class would the do a check on the timestamp of the latest meals by using various methods in the `Notification` class, more details are given below. If a meal had been missed, a notification will be prompted to the user to consume his meal, this prompt will constantly pop up every 10 minutes until a meal had been consumed. If a meal had been consumed within the period then the user can continue using the application without any prompt.

NOTE	Notification will be prompted: - After 10am for breakfast - After 2pm for lunch - After 8pm for dinner

NOTE	Meals that will be counted: - As breakfast - anytime - As lunch - after 11am - As dinner - after 4pm

Additionally, it implements the following operation:

- A "timer" object that is in the `UIManager` class will ensure that the notifications are executed in 10 minutes interval.
- `Notification#eatenBreakfast` — returns a boolean value to indicate whether the user has eaten their breakfast.
- `Notification#eatenLunch` — returns a boolean value to indicate whether the user has eaten their lunch.
- `Notification#eatenDinner` — returns a boolean value to indicate whether the user has eaten their dinner.

Given below is an example usage scenario and how the notification mechanism behaves at each step.

Step 1. The user starts up CaloFit for the first time. The `DishDatabase` will be loaded with the initial

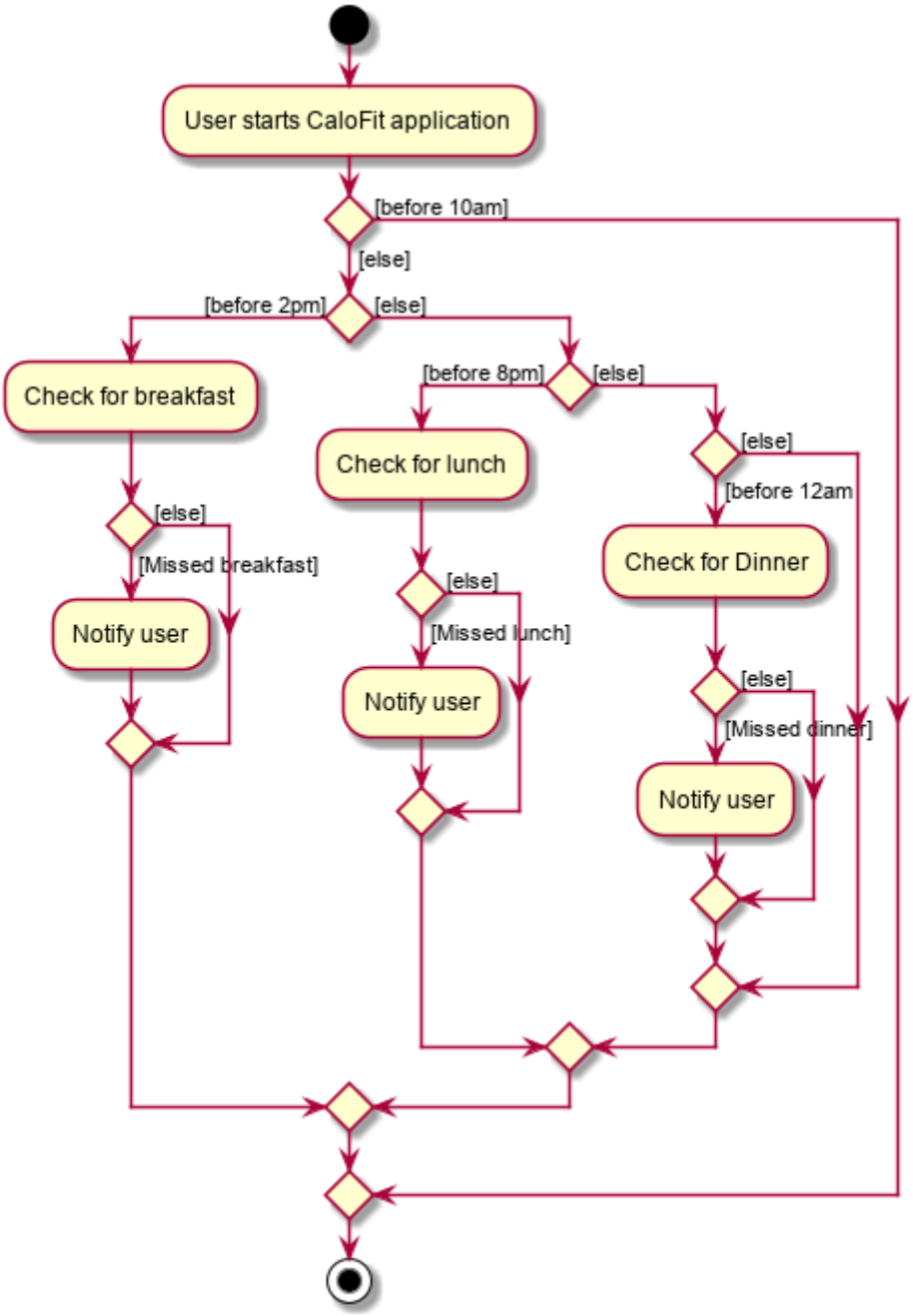
data by calling `MainApp#loadDishDatabase` with an empty MealLog.

NOTE	The application will immediately check once, when the application is launched, followed by every 10 minutes interval.

Step 2. The application will then execute `NotificationHelper` once to check if any meal have been missed. If there is none then no notification will be prompted. If a meal is missed it will prompt to the user, and notify them every 10 minutes.

Step 3. After 10 minutes, `UIManager` will then execute `NotificationHelper` and check if the user has keyed in any meals. This process will carry on for every 10 minutes utill the user keys in his meal.

The following sequence diagram shows how the notification operation works:



Design Considerations

Aspect: How notification executes

- **Alternative 1 (current choice):** Refocus the notification Window.
  - Pros: Ensure that less memory is used, so that buffer overflow is not possible. Furthermore user would not be annoyed by multiple tabs.
  - Cons: Higher chance of notification not showing up due to a single error.
- **Alternative 2:** Create a new Window for each notification
  - Pros: Less prone to mistake as previous notification will still stay until the user clears it.
  - Cons: Require much more memory as a new window is created, if the user was away for a period of time and the application was left opened, user would be required to clear quite a number of tabs.
- **Explanation of Choice:** Since this feature serves as an assistance to the user, we should not bring in more hassle and inconvenience to the user. Thus even with a higher risk of notification not showing up due to error. After 10 more minutes the notification will be prompted.