FoodieBot - Developer Guide

By: `F11-03`   Since: `Feb 2020`   Licence: `MIT`

1. Introduction

1.1. Software Overview

FoodieBot is a campus food locator application which can help students look for food they want, by recommending the canteens nearest to them. Foodiebot is one stop application to solve all meal decision on NUS.

1.2. Purpose

The purpose of this document is to describe the architecture and system design of the FoodieBot. This documentation is for anyone who wish to understand more about FoodieBot and how it works.

1.3. Audience

Our target users are students, staff and tourists, in general anyone who comes to or visits NUS. In particular, for the indecisive user, this application can give a random food suggestion tailored to each user based on their budget and/ or past food selections etcetera.

2. Setting up

Our application is being managed by a dependency management tool Gradle. The project is available at https://github.com/AY1920S2-CS2103T-F11-3/main. After cloning the repository, you can follow these steps: We assume that you are using IntelliJ.

1. Run gradle build.
2. Run gradle run.

3. Definitions

4. System Overview

4.1. Architecture

4.2. Algorithms

4.3. Detailed Class Descriptions

4.4. Design Considerations

Appendix A: Product Scope

Target user profile:

- has a need to keep track for food expenses
- are indecisive on what food to have in campus
- does not know which canteens are near them
- is comfortable with command-line inputs on desktop

Value proposition: get a food choice decided without having to work with GUI controls

Appendix B: User Stories

NOTE The user is not particularly limited to student and stuff, it can be anyone who comes to visit NUS and is introduced to use the app

Priorities: High (must have) - * * *, Medium (nice to have) - * *, Low (unlikely to have) - *

Priority	As a …	I want to …	So that I can…
* * *	new user	see usage instructions.	refer to instructions when I forget how to use the App.
* * *	user	find nearest canteens.	get to the canteen quickly.
* * *	user	see which stores are open.	remove entries that I no longer need.
* * *	user who is new to NUS (tourist, visitor or freshman).	get a clear directory to the canteen	make my way to the canteen with ease.
* * *	user in campus	randomize a food choice.	try something new every now and then.
* * *	user who is budget conscious	know which food items fall within my budget.	I would not overspend.
* * *	user	take down some personal notes about the store, for example which dish at the mixed veg store is good.	see which is my favourite food amongst the NUS canteens.
* *	user who has an idea of what s/he wants to have	search for food items.	see which canteens sell them.
* *	user	see which food items I have not tried.	try all food items in the canteen.
* *	user with disability	know if there is convenient access to the canteen.	try all food items in the canteen.
* *	student on budget	search through prices of food items in different canteens.	discover which are the cheapest food items.
* *	user who do not carry a lot of cash	see the type of payment methods available.	prepare myself beforehand.

Priority	As a …	I want to …	So that I can…
`* *`	user	track the frequency of the food I eat.	eat certain food in moderation and save money if i have been eating expensive food frequently.
`* *`	user	see some images of the food .	get some better understanding of the food aside from just the food description.
`*`	user who is health conscious	view the dietary options available for each canteen.	know which stall i can visit.

B.1. User Story for Version 2.0

Priority	As a …	I want to …	So that I can…
`v2.0`	user	place an order.	receive the food when I arrive.
`v2.0`	store owner	add new food items on the menu.	easily update the menu.
`v2.0`	store owner	set menu items to be on promotion.	attract more students to select the menu item.
`v2.0`	user	view the crowd condition.	avoid going to the canteen if it is too crowded.
`v2.0`	user	send invitation to a friend.	have meals together with friends.

Appendix C: Use Cases

(For all use cases below, the System is FoodieBot and the Actor is the user, unless specified otherwise)

Use case: UC1 - Remove randomiser suggestion

MSS

1. User requests to randomise
2. FoodieBot shows the past randomized suggestions

3. User request to remove suggestion
4. FoodieBot updates the food item not to be suggested in the future

   Use case ends.

Extensions

   3a. The given index is invalid.

      3a1. FoodieBot shows an error message.

      Use case resumes at step 2.

Use case: UC2 - Set Budget

MSS

1. (Optional) User requests to view budget
2. FoodieBot shows the current budget with list of expenses
3. User request to set budget
4. FoodieBot updates the budget for the specified period

   Use case ends.

Extensions

   3a. The given amount is invalid.

      3a1. FoodieBot shows an error message.

      Use case resumes at step 2.

   3b. The given period is invalid.

      3b1. FoodieBot shows an error message.

      Use case resumes at step 2.

Use case: UC3 - Review Food Item

MSS

1. User requests to view transactions
2. FoodieBot shows a list of transactions
3. User request to review the food item in the list
4. FoodieBot shows the edit screen for user to update
5. FoodieBot saves the user review

Use case ends.

Extensions

2a. The list is empty.

Use case ends.

3a. The given index is invalid.

- 3a1. FoodieBot shows an error message.

  Use case resumes at step 2.

5. The cancel command is supplied.

Use case resumes at step 2.

Use case: UC4 - Rate Food Item

MSS

1. User requests to view transactions
2. FoodieBot shows a list of transactions
3. User request to rate the food item in the list
4. FoodieBot updates the review for the food item on the list

   Use case ends.

Extensions

2a. The list is empty.

Use case ends.

3a. The given index is invalid.

- 3a1. FoodieBot shows an error message.

  Use case resumes at step 2.

3b. The given rating is invalid.

- 3b1. FoodieBot shows an error message.

  Use case resumes at step 2.

Appendix D: Non Functional Requirements

1. Should work on any mainstream OS as long as it has Java 11 or above installed.
2. Should be able to hold up to 1000 food items without a noticeable sluggishness in performance for typical usage.
3. A user with above average typing speed for regular English text (i.e. not code, not system admin commands) should be able to accomplish most of the tasks faster using commands than using the mouse.
4. Should be usable by users who have never used an e-directory
5. The application should have images for the food items, if the food items are shown to the user
6. The system should be backward compatible with data produced by earlier versions of the system

Appendix E: Glossary

Mainstream OS
Windows, Linux, Unix, OS-X

Appendix F: Product Survey

Product Name Pizza on iOS appstore

Author: Bryan Wu

Pros:

- Allow randomisation for food that requires choosing of ingredients

Cons:

- Allow choosing of ingredients for pizza only
- Does not recommend which stores sell the pizza