COMP 3111H Group 13 Milestone 3

Github URL

https://github.com/ignavier/hkust_cs3111h

QR code

User Guide

When users first add the Dietbot as a friend, the Dietbot will ask the user for their physical information. Although this section is very long, but it is very important, as we need to know the user's physical information in order to give the best recommendations. But do not worry, as you still can change your information in the future.

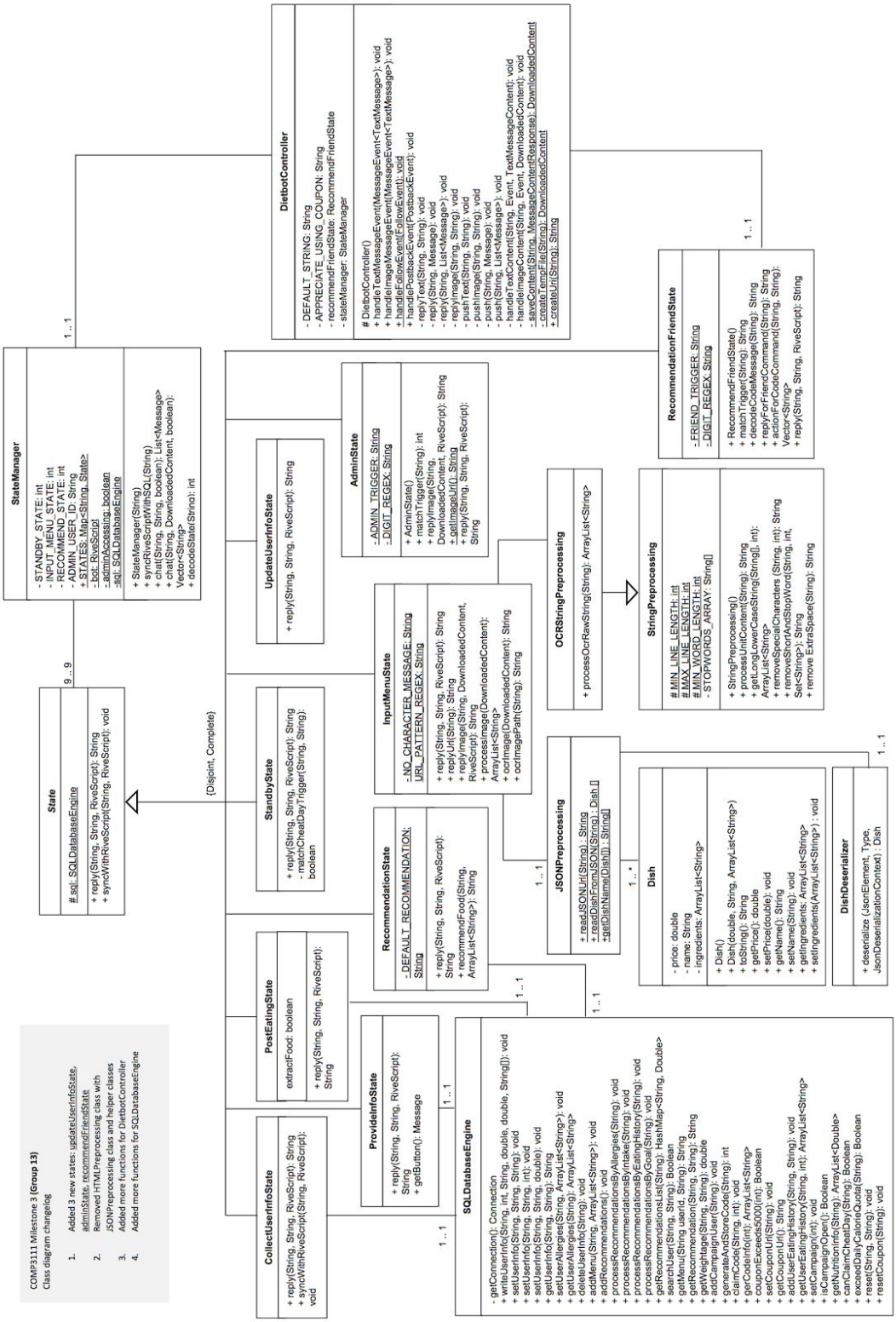
Afterwards, our Dietbot provides the following functions:

- The user can update their personal information.
- The user can ask the Dietbot for a food recommendation or suggestion. Then, the Dietbot will ask the user to input a menu, either by text, image, or JSON.
- The user can record what they ate.
- The user can check the nutrition information of a specific food, or check their nutrition history.

We have tried our best to make the conversation with our Dietbot feel as natural as possible, so just type what you think makes sense. For example, if you want to record what you ate, you can try "I want to record what I ate", "I just had my dinner", or simply "Record".

Other than that, just follow the instructions given by our Dietbot, and everything should work smoothly. Wish you a happy and comfortable experience using our Dietbot!

2a. UML class diagram



(Typo: It should be RecommendFriendState, not RecommendationFriendState)

2b. Design pattern

Design pattern(s) implemented: **Finite State Machine + Mediator**
Architectural pattern(s) used: **MVC + Pipe and Filter**

Finite State Machine: The reason why we choose the Finite State Machine design pattern is that it is scalable. Initially, we wanted to use Boolean flags to indicate the current state, but we realized that there are too many states, and handling the transition is very complicated.

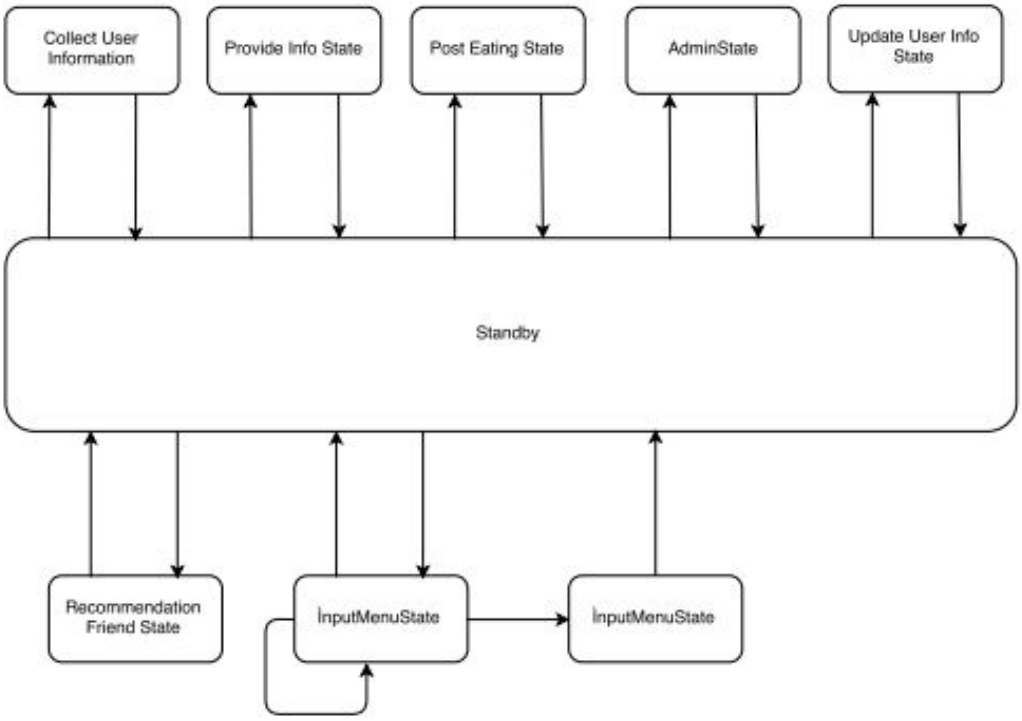
MVC: In our chatbot operation, our SQLDatabaseEngine class is the Model which manages the data. Our Chatbot (Rivescript) is the View which format the reply to users, and our StateManager class is the controller. Our StateManager is in charge of transition between the different states, and it controls access to our database and the replies that our Chatbot provides to the user.

Mediator: Our implementation of the StateManager is also very easy to maintain because the various states are not aware of each others' existence; the transitions are entirely handled by the StateManager. Since our states are independent of each other, we can remove states or add new states very easily. In other words, our StateManager is the mediator of all the different states.

Pipe and Filter: For processing recommendations on what to eat, we used the Pipe and Filter design pattern. When the user inputs a menu (either in text, JSON or image format), we will first match it with our food database and store its information into our recommendations table. Then, we have four layers of processing:

1. Process recommendations by user's allergies.
2. Process recommendations by user's daily recommended intake.
3. Process recommendations by user's eating history.
4. Process recommendations by user's dieting goals.

After the recommendations are completely processed, they are returned to the user as a meal recommendation.


Collect User Information
Provide Info State
Post Eating State
AdminState
Update User Info State
Standby
Recommendation Friend State
InputMenuState
InputMenuState

6. Project management tools

We used Trello and Github Issues as our project management tools.

All functions we have used with Github Issues

We highlighted some bugs or unexpected behavior of deployment in the GitHub issues. For example, when the Rivescript does not respond as expected, or when our Database function does not return the correct result.

All functions we have used with Trello

For Trello, the biggest problem we had was piling up too much card into one segment. The problem was that it was hard to find the updated comments from the board, and we spent too much time finding the issued on GitHub. We solved the problem by making new checklists and most importanty, we reordered the checklists every 2 hours to ensure all necessary information is easily found.

Aside from that, not all members were working in person in the early stage and they forgot to check the Trello app. We find this to be an issue, so we used the send email function because all of us check UST email very often. The response rate increased in the most critical period, which is the last 3 days.

Also, we used the burndown chart to estimate the time and effort we needed. We find that our estimated completion time overshot by 2 hours, so we noticed the issue and stayed up overnight to finish the project on time.

Screenshot for Github Issues

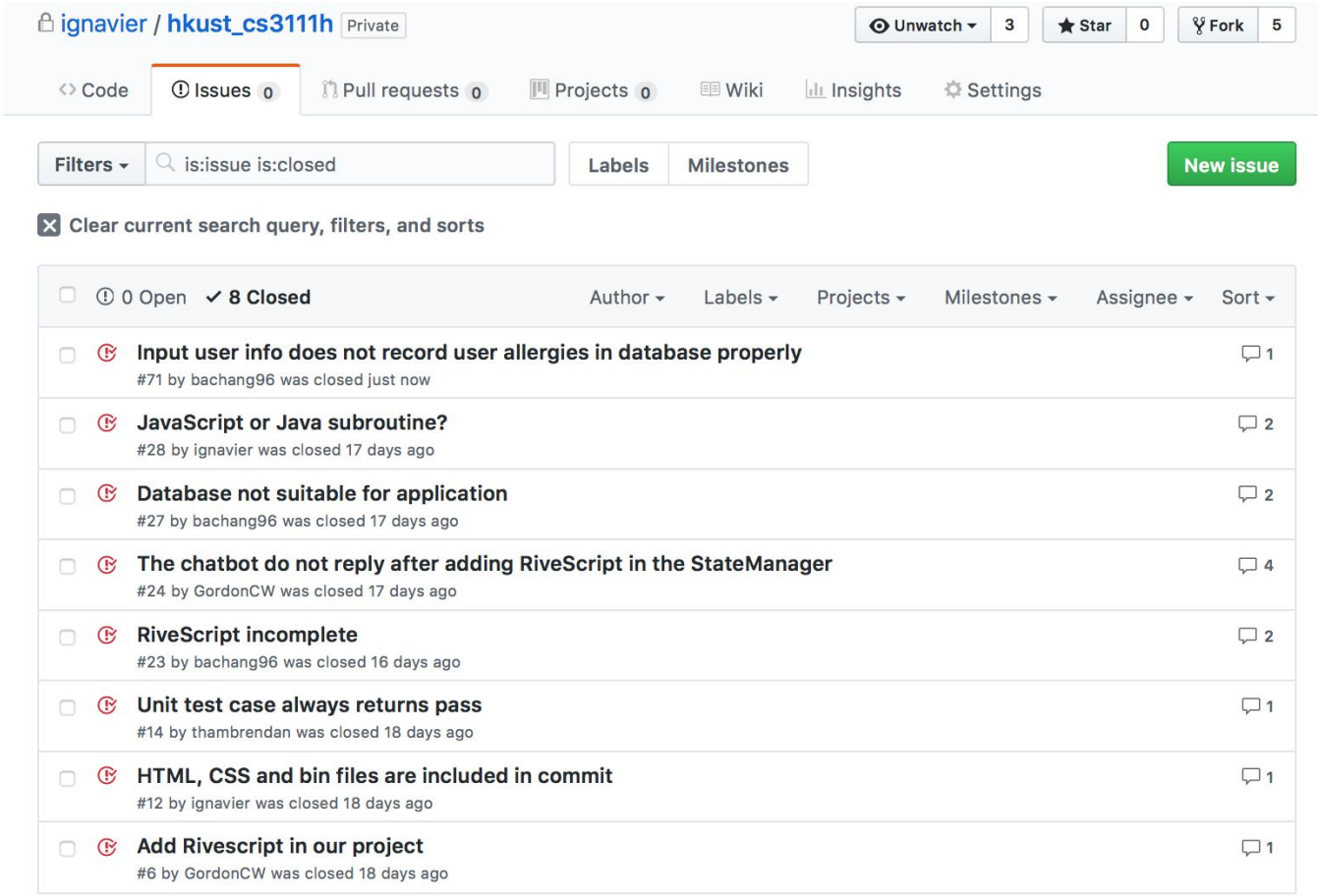


Diagram 1: Github issue as of completion of milestone 3

Screenshot for Trello

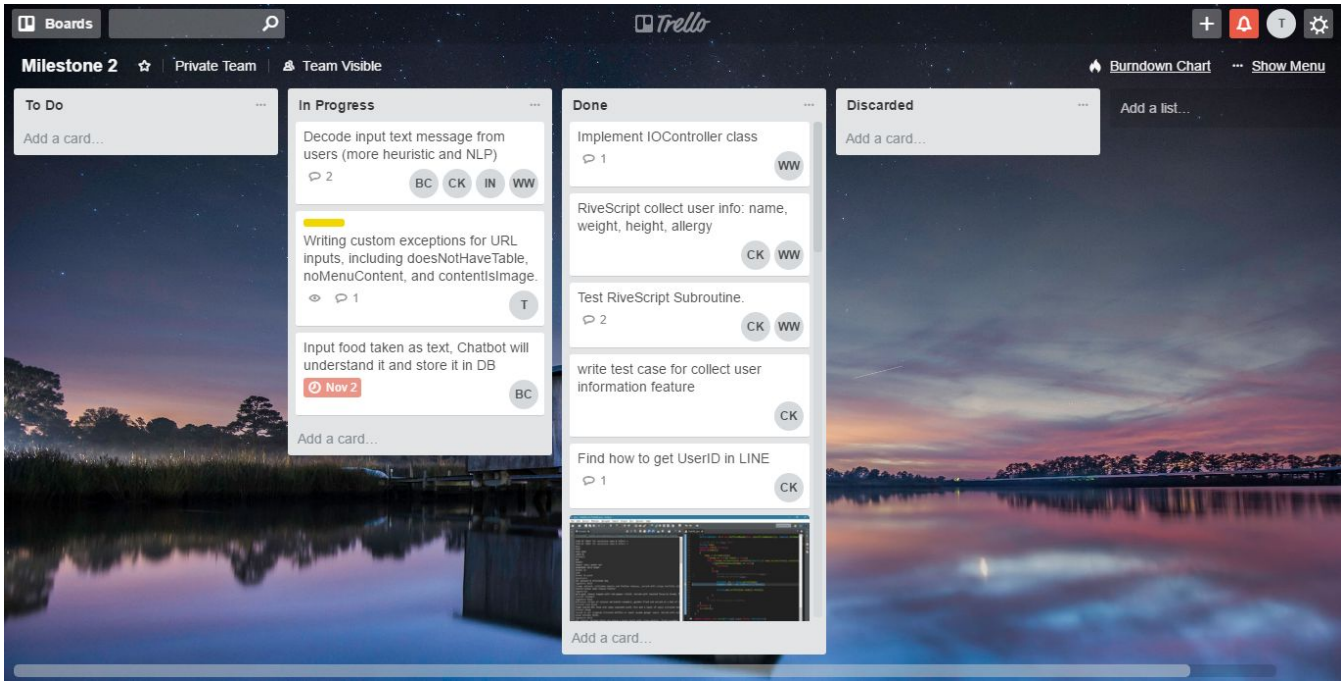


Diagram 2: Trello board as of completion of milestone 2

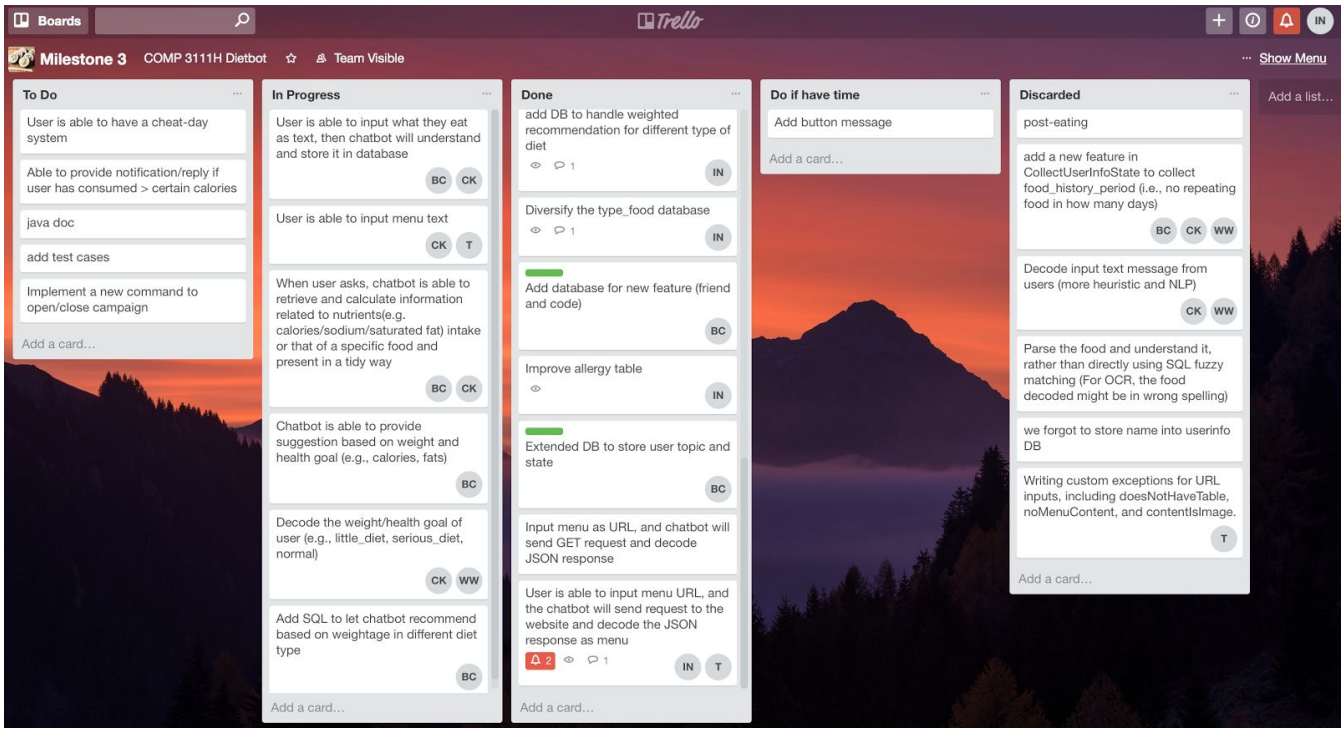


Diagram 3: Trello board during development of milestone 3

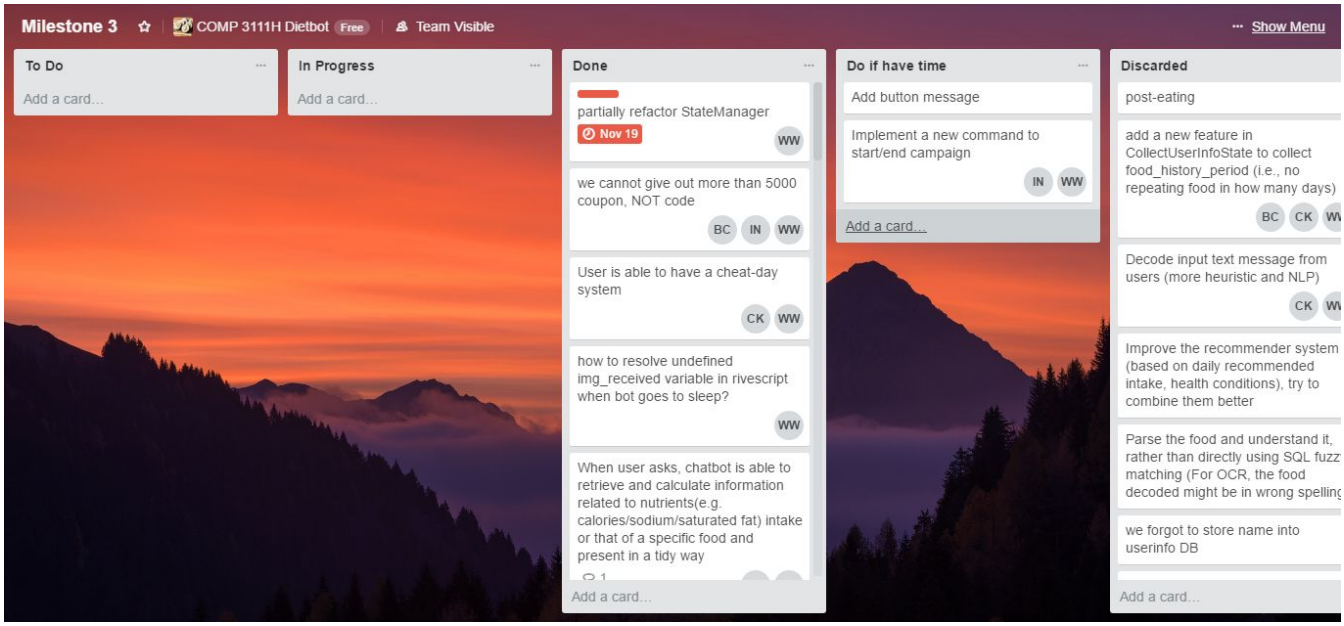


Diagram 4: Trello Board upon completion of all tasks

4. Test cases

Description on how we do testing

After we wrote a function, we wrote the corresponding test case to ensure that the functions works as expected. This way, if a new function makes an old test fail, we would know. Also, we would know if certain functions do not work after integration.

How many test cases we wrote

44

Test Coverage Report

https://github.com/ignavier/hkust_cs3111h/blob/master/jacocoHtml/index.html

Appendix

(appendix here)

Disclaimer

To return back to standby state, input “talk later” and the chatbot will exit its current state. Currently, the chatbot will output its current state after every message for to avoid confusion because we have yet to the functionalities implement all states.

7. Feature Table

ID	Features	Completed?	Sample Test Cases	Expected Result	Checked?
1	User is able to input **menu text**, and the chatbot will **recognize** it	Y	User type “input”. And input some food separated by comma	The chatbot will reply user “Thanks, I'm looking at your text menu now! I'll try to give you some recommendations.”.	
2	User is able to input **URL** that leads to JSON content, and the chatbot will **recommend the most suitable dish** it	Y	Recognize the URL and i) create the associated *Dish* objects and ii) recommend the *name* of the most suitable dish URL: “http://www.json-generator.com/api/json/get/cjTeRHAnfS?indent=2”	I would recommend you to eat <food_name> where food_name = { Chilli Chicken on Rice, Sweet and Sour Pork served with Rice, Spicy Bean curd with Minced Pork served with Rice}	
3	User is able to input **menu in JPEG format**, and the chatbot will **recognize** it	Y	Recognize the menu and preprocess it (remove special characters and digits)  	splcy bean curd wllh mlnced pork served wllh rice sweet sour fork sewed thn rce chlh chlcken che fried instance needle luncheon meat [There might be some incorrect words]	
			Recognize the menu and preprocess it (remove special characters and digits)	shortbread puddle cookie cookie macaroon biscotti	

				ginger choc teabread organic bliss cookie loralyn bar muffin croissant organic bliss teabread glutenfree teabread cinnamon roll scone bear claw boulder cookiett brownie almond chocolate croissant ham cheese croissant [There might be some incorrect words]	
4	User is able to **input what they eat as text**, then chatbot will **understand and store** it in database	Y	User reports that they've eaten "chicken soup, spaghetti bolognese" and "apples, chocolate cake"	Two new rows are added to the eating_history table in our database. The first row contains "chicken soup, spaghetti", while the second row contains "apples, chocolate cake".	
5	When user **first** talks to the chatbot, it is able to **collect user's physical and medical information**, then **store** it in a database	Y	Function testCollectUserInformation in DietbotTester	The chatBot will ask the user to answer a series of questions like "What is your name, What is your age, Are you allergic with milk…". Most questions are followed immediately with a prompt for confirmation from user to ensure that there is no typo or wrong information inserted into the database.	
6	User is able to **reset and update their user**	Y	Function testCollectUserInformation in DietbotTester	This chatbot will be triggered to ask a series of questions when the user type something like "i want to update my user information".	

	information stored, **delete relevant records** in database			The chatBot will ask the user a series of questions just like when the user first engage with the chatbot. The format is exactly same as that of (5).	
7	Chatbot is able to provide suggestion based on **weight and health goal**	Y	Create two users, one who is on a little diet, another who is on a serious diet. Both of them input the same menu to the Dietbot.	For the user who is on a little diet, the Dietbot would be less likely to suggest meats and grains, compared to a user who is on a normal diet. For the user who is on a serious diet, the Dietbot would be a lot less likely to suggest meats and grains, and a lot more llikely to suggest vegetables and fruits, compared to a user who is on a normal diet.	
8	Chatbot is able to provide food recommendation based on **daily recommended intake**	Y	Input “chicken potato soup” and “caramel apples” as the menu. The user is 19-year-old male.	“Chicken potato soup” is recognized as meat, while “caramel apples” is recognized as fruits. For 19-year-old males, the daily recommended servings of meat is 2.5, while that of fruits is 2, and this is reflected in our recommendation table.	
9	Chatbot is able to provide food recommendation based on various **health conditions (e.g., allergy)**	Y	Input “chicken potato soup” and “grilled salmon” as the menu. The user is allergic to seafood.	After processing the recommendations by user allergies, the “grilled salmon” is removed from the recommendations table.	
10	Chatbot is able to provide food recommendation based on **eating history**	Y	Over the past 3 days, the user has eaten apples twice, bananas once, and he has never eaten an orange.	Note that all of the food are fruits, so normally their weightage within the recommendations table is the same. But after processing the recommendations based on user history, the weightage of oranges is 2, whereas the weightage of bananas is 1, and the weightage of apples is 0.5. This reflects that our	

				system is a lot less likely to recommend food that the user has recently eaten.	
11	User is able to have a **cheat-day system**	Y	First, the user asks if he can claim a cheat day. Then, after claiming a cheat day, he asks if he can claim the cheat day again.	When the user first asks if he can claim a cheat day, the response is “yes”, because the user has not claimed a cheat day within the past week. However, after the user claims a cheat day, when he asks if he can claim a cheat day again, the response is “no”.	
12	When **user asks**, chatbot is able to **retrieve and calculate information related to nutrients**	Y	The user asks for the nutrition info of “fried chicken”	Based on our nutrition database, we will return that fried chicken has 71kcal of calories, 354mg of sodium and 0.79g of fat.	
			The user eats “fried chicken” and “chocolate cake” once, then he asks about his nutrition intake info. Then, the user eats “fried chicken” and “chocolate cake” again in the same day, and asks again.	When the user asks the first time, our Dietbot will sum the total amount of calories, sodium and fats that the user has eaten. The results are 1050kcal calories, 291.1mg of sodium, and 4.3g of fat. After the user eats the same foods again in the same day and asks again, the total amount of nutrition is doubled, which are 2100kcal calories, 582.2mg of sodium, and 8.6g of fat. If the user waits another day before asking again, and he does not eat anything in the meantime, then the results will be the same as the first.	
13	Chatbot is able to provide **notification** (as warning) if user has consumed **more than certain calories**	Y	The user eats “fried chicken” and “chocolate cake” 3 times. After each time, our Dietbot will check if the user has exceeded his daily calorie quota.	For a 23-year-old male, their daily calorie quota is about 2600kcal based on our database. The calories of fried chicken is about 500kcal, whereas the calories of chocolate cake is about 550kcal. So, after the user eats friend chicken and chocolate cake twice, nothing will happen. But after the user reports that they ate friend	

			The user is a 23-year-old male.	chicken and chocolate for the third time, the Dietbot will warn users that they have exceeded their daily calorie quota.	
14	The admin is able type **“admin:upload_coupon”**, then he can upload the coupon which will be later send when user claims it	Y	This can only be done using Wong Wen Yan (SID: 20318893)’s phone because of its admin status. At the time of submission, a coupon should have been uploaded already.	The URL image uploaded by Wong Wen Yan will be stored in our database. Once someone manage to claim a coupon, our chatbot will access the URL to send out the coupon to user who requested the code and user who claimed the code	
15	User is able to use **“friend:generate”** command to generate a unique six digit coupon code	Y	A user who have registered himself (finished going throught the collect user information stage) should be able to get a 6-digit code as long as the number of coupons claimed has not exceeded 5000.	Type **friend:generate in** LINE**.** A 6 digit code will be received.	
16	User is able to use **“code:<six_digit_code>”** to claim coupon coupon.	Y	These are the requirements for the code to be claimed. 1. The code is generated by another user before 2. The code is not claimed before 3. The user added	The expected results is as mentioned in “Sample Test Cases” column. Only if all the requirements are fulfilled, the user can claim the coupons, and our chatbot will access the URL to send out the coupon to user who requested the code and user who claimed the code.	