Quality Assurance Plan

CMPT 276

Group 5

App: Mealify

Members: John Zheng, Justin Yu,

Juey Lew, Vincent Yu, Feng Wu

Table of Contents

Unit Testing Tools and Methods 3

Unit/system testing internal deadlines 3

User acceptance testing of version 3 3

Integration testing 3

Measuring size and complexity 5

Miscellaneous 5

1. Unit testing tools and method

Component testing and integration testing were used to test the usability of our app. Each member uploads their version of the app onto Github, and then writes out a testing plan on Jira for the QA analyst to execute. If a defect is found after the test, a ticket is sent to the member that created that version for the member to fix the corner cases. This step is then repeated until there are no errors.

2. Unit/system testing internal deadlines

For iteration 2, each member had an important role in completing the sprints on time. John was tasked with integrating the database from USDA to CNF. Vincent was tasked with tracking the user nutrient information. Feng was tasked with beginning to implement the friends system. Justin was tasked with the calendar and bar graph functions. Juey was tasked with upgrading the user interface through high-quality assets. Since each member had a important task in a span of 2 weeks, we set out sprints for each member on Jira to ensure we have time to do testing. Everyone's sprints were to completed on July 16th. For the remaining day and half before the deadline, we ensured the quality of our app by doing system testing and improving our UI.

3. User acceptance testing of version 3

Since most of all the application's core functionality has been implemented within the first two iterations, it would be ideal to perform a user acceptance test July 25, 4:30 pm, which is a week before the due date of version 3 of the project. Here we will ask users to do the following:

- input the amount of food intake for multiple days and update us on whether the application properly and accurately tracked the total food intake on each nutrient (carbohydrates, fat, protein, iron, folate, vitamin D, magnesium)
- after inputting the food intake for a meal, check to see if the the meal's nutrients, and date/time of consumption is accurately recorded in the calendar
- test adding/removing friends and checking the leaderboards function
- to notify us in any confusion the application has created in regards to the flow from one page to another (if the user interface makes sense to the user)

4. Integration testing

Integration Test 1. For our first version of the project, we have finished implementing the feature that would allow users to fill in a form to input the food intake for a certain meal whether it be for breakfast, lunch and dinner. The result are then stored in our database that way we can query the nutrient intake and use the Charts API to help the user visualize intake over a period of time. To perform an integration test between the form, database and Charts API the Quality Assurance member and other developers have inputted nutrient data over multiple days and check if the charts plot the correct values. We also checked the database if the results are the same values as it was originally input in the form. This testing of this feature and fixing bugs that arise took place between June 29 - July 1 which left us one week to implement this feature.

Integration Test 2. The second version of the project included the ability to track meals via the Google Calendars API. The user's breakfast, lunch and dinner nutrient intake as well as its meals itself will be included in the calendar in the corresponding date of consumption. To test this feature, whenever the user inputs a meal into a form, the meal and its ingredients will be placed in the calendar in the correct date of consumption. We completed this test on July 16th.

Integration Test 3. Mealify would allow new users to register an account to the application. In the registration form, the user would input username, e-mail, and other health/fitness statistics. This data would be saved in the Firebase authentication database. The testing procedure would include registering a fake user with data in the registration form and asserting if the correct values are inputted into the database. The passwords for users would have to be encrypted for user's protection. This feature has been tested on June 26th.

Integration Test 4. The add friends feature implemented in iteration 2 will be tested by accessing through the database to ensure both friends have the other in their friend's list. Removing a friend is tested by checking in Firebase if the friend is removed from the friends list. This feature has been tested on July 16th.

Integration Test 5. The Food Asset Map will be implemented in iteration 3 and completed by July 30th. The feature would allow users to pick a certain diet restriction and the map would mark restaurants that serve meals that would cater to those restrictions. A way to test this is to make sure that for each restaurant that was marked there exists at least one meal in the menu that satisfies the user's diet restriction. Also, the markers must be placed in the right restaurant's coordinates.

The difference between the integration tests above and unit testing is that these integrations requires multiple components and external APIs. Unit testing would only cover tests for each individual component rather than multiple components and their interactions. In order to prevent “big bang” integration tests, we perform integration tests for each feature implemented then proceed to add integration tests between multiple features for each iteration.

5. Measuring size and complexity

As more features are implemented into our project, it is inevitable that the size and complexity of the source code will increase. We currently have around 30 swift files and multiple cocoa pods installed on a pod file. The size of all our files is about 830 MB. To ensure that our app is organized, we create folders to organize each member’s code.

6. Miscellaneous

For each feature implemented, the developers would create their own ‘git branch’. Before merging the branch to master, as the quality assurance analyst, I would have to perform integration testing and to see if the features implemented does what is specified in accordance to the requirements documentation. Any issues will be logged in a separate document that way the developers that is responsible for implementing the feature can change the source code in order to satisfy the actual requirements. When the feature implemented satisfies all the requirements specified, it is the quality assurance analyst’s responsibility to merge the branch back to master and log the feature as completed.