Test Plan

Test Philosophy

For our team's philosophy, we believe that it is impossible to evaluate 100% of all possible test cases. The reasoning for this is that the scope within testing will be based on just members within the group. In order to deliver the required features and functionality needed for a functioning application, we believe that the feedback of our customers and stakeholders play the most important part in addition to our decisions. Most of our tests are based on implementation of basic functionalities before adding quality of life features.

Our testing densities consist of testing each modular component before implementation to make sure that each component works properly. Less understood code would definitely go through more testing as it would help us understand the functionality of the code with some portions of code that does not require testing.

We believe that our current plan of testing is sufficient for this project because most of our projects consist of components that require simple communication between each other. Our criteria for sufficient testing mainly comes down to the following: a user enters a recipe name, a request is made in the back end, information is fetched and then portrayed for the user to see and select the one they like.

Team Workflow

Our team's test workflow consists of two test plans with multiple steps in between.

Front-End

One of our testing plans would be the front end. First, we determine what feature that is needed to implement for the front end. Next, we would search up the documentations needed to implement such components. After, we would test the functionality through several stresses for all possible situations to the fullest of our ability. Lastly, we would fix all errors that are related to the functionality before adding it onto the application.

Machine Learning Research

Our other testing plan would be the machine learning aspect of our project. We decided to implement a recommendation model to use for our application, specifically a content-based recommendation model. Most of the testing that is done here consists of research in learning everything needed in regards to the model. A lot of research consisted on what we wanted to do in regards to our model. We first had to figure out whether we wanted to do content-based recommendation or collaborative filtering. We split up the research on both of these types and came to the conclusion that content-based recommendation would be better for our product. It would allow for our users to get recommended recipes based off of their past searches. This is the easiest to get the most accurate results for our users. We will be using cosine similarity to find the correct scores for each recipe result. By determining the tf-idf we will be able to properly compare the different searches.

Something that is also related to our machine learning is our recipe finder. There are two parts to our recipe finder, the vocabulary list and the web parsing. To make breaking down the code easier, we created several programs to separate the words in a paragraph and put them into a text file. To test these two programs, a random recipe is searched for and thrown into the paragraph splitter. This test is to make sure that the word is being split correctly as well as the words are being added to an existing word that already exists inside the file. Multiple tests are done in this section to account for the possible characters that website creators might include inside their paragraph for example the small circle that people typically use to shorten the word "temperature". Since not everyone can work on a file at a time, we had to solve that by creating separate text files for everyone and then create another program to take everyones text files and combine them. This part of coding did not need to be tested as hard because it is only a

matter of changing the file path when working with multiple files. After separating the words of importance to those that are not as important, we create a program that will take that scoring system that also utilizes word parsing to create this program but it is not working yet.
To test our parser, we only needed to try several different HTML tags to see what will be pulled out and which section of returned text will have to be taken out. Since our recipe finder is not currently working, this is the farthest that we will go for now.

We determine code coverage by going through each of the components within the application and the machine learning models being tested. Each component is tested many times throughout the lifecycle of the sprint before official implementation.

Bugs often come from our own tests rather than the user. Many of the feedback that comes back from the user are in regards to suggestions on making the application better rather than bug reports. When these bugs do happen, they do not exactly hinder our program workflow as when bugs do happen, we immediately work towards remedying the issue before continuing.

Test Design

For the test design, we wanted the code path to be very short and very linear. We wanted the experience to be straightforward and intuitive. The user makes an account, they find a recipe they like, they cook it and choose to save it or search for another. Then we wanted the cycle of search and cook to repeat. The only use cases in the code path we have to check back for are input checks and whether or not the user wants to save a recipe.

For how much of our code is tested, anything that we have right now has been tested as soon as we get minimum functionality working. From there we test the use cases that depend on user input.

The test environment is within our local machine. We make sure that we always test before we deploy. We run a localhost version of the web application from which we can test all the functionality. The test code and data for our language processing are all up on our GitHub as well.

For version control, we use GitHub. Hosting is all done on Google Firebase, but GitHub allows us all to work on separate projects within our respective branches. We then merge to master and deploy from the master branch when the feature is complete. We currently have no bug trackers deployed.