## 1. Test Plan

The intended purpose of our app is to encourage users to change their eating habits and start cooking for themselves, in order to test the success of this goal we would ideally have external users test our system. However, our app does not simply provide multiple, independent functions, rather, our system combines several different complex functions that create the complete experience for our user. Instead of attempting to test an incomplete product, we wanted to give a functional iteration to a user, which we knew would not be available until the end of the development cycle. Therefore, we knew the majority of our testing efforts would be focused on development testing. Our core functionalities are the food log and recipe recommendations, so we needed to make sure that these worked well for the demo. Our approach during development was to check that after any addition to these functionalities, our overall system continued to work. Though we did not use test-driven development, we used an incremental approach to testing. We added new functionality, then tested several inputs and made sure that we received the expected outcomes.

## 2. Technology Used

Postman is a chrome application that allows the creation of REST api calls and test. We used postman to test our database by making various REST calls and comparing with what we expected the output to be. This software allowed us to make easy requests to our server, and helped ensure that the data we store, change, or request, is being processed correctly. This application provided an efficient way to ensure that the database was working correctly.

## 3. Descriptions of the files

All the tests are included in one file which is opened with Postman,

| File Name | Description |
|---|---|
| GetCookingTests.postman_collection.json | Contains all the test files for the REST api. Is opened with Postman to view individual tests. |
| LogMeal | Tests that when a meal is added to the server, the request sent back is a confirmation of success along with the user id sent to it and the meals added to the log. |
| LogMeal(2) | Second test of the same variety. Tests that when a meal is added to the server, the request sent back is a confirmation of success along with the user id sent to it and the meals added |

| | |
|---|---|
| | to the log. Uses a different request than the first test. |
| GetLog | Tests that when a request for the log is made, that the log is sent back for the user with the user id specified. |
| GetLog(2) | Second test of the same variety. Changes the user id to another. Also ensures that the information sent back is what is expected for that user. |
| GetRecipe | Retrieves the ingredients and directions for the recipe with the id specified. |
| GetRecipe(2) | Second test of the same variety. Uses a different recipe id than the one requested. |
| Search | Makes a REST request to the server with a specific query. Ensures that the response sent back is what is expected. |
| Search(2) | Second test of the same variety. Uses a different query than the previous test. |
| GetRecommendations | Tests that the recommendations for a user are working by sending a request with a specific userid. Ensures that the response is what is expected for that user. |
| GetRecommendations(2) | Second test of the same variety. Uses a different userid than the previous test. |

## 4. User Testing

We did not use explicit use user testing, since we did not give a working product to an external user. However, the members of the team that did not work on the UI were asked to look at different iterations of the interface to ensure that it performed as expected. They provided feedback on the interface as if they were a user, since they did not have a direct hand in designing it.