# Test Document

Mobile Cooking Companion

Team 6

Jeremy Ray, Thomas Roberts, Vitor Santos, Joshua Schoen, and Katlin Scott

**Version: (4)** **Date: (11/21/2021)**

## 1. Testing Strategy

### 1.1 Overall strategy

**What tests will we conduct?**
Unit Testing. Specifically for the database management system (DBMS), we will test its basic functions (Recipe info can be stored, pantry can be updated, tables can be searched). For the Recipe Unit, we will test the methods listed in the software design document (SDD): addRecipe(string), addStep(string), editStep(string), and deleteStep(). For the Ingredient Unit, we will also test the methods listed in the SDD: newIngredient(string), editIngredient(string), and deleteIngredient().

Functional Testing. There are four general units (The UI (User Interface), Pantry, Ingredient, and the DBMS). This testing will make sure each corresponding unit can function together. For example, we want to test that information provided by the user can be converted into code and the application's data can be transferred to the DBMS. This testing can be summed up as making sure there is good cohesion between units.

Integration and Systems Testing. This testing will make sure the complete system functions. After unit and functional testing are conducted, systems testing will be used to check that the application is able to run. These tests may include mock users trying various app features or using a separate program to test various application pathways.

Stress Testing. This testing will answer the questions: How many recipes can the user add? How many ingredients can a recipe store? Can the user overload the search function? This testing will also make sure objects stay within their parameters (Last names remain under 30 characters, no negative measurements, and no blank entries).

User-interface Testing. This testing will focus specifically on the user interface through white-box testing. This will include confirming that all buttons work, that pages can be accessed and returned to, and that data can be entered.

Lastly, it is worth mentioning some tests that we considered and a brief reason the test was not incorporated. Configuration testing – we only have 1 version of the application that responds to only 1 user at a time. Conformance testing – As we do not work under a legitimate company, there are no specific policies the application must follow. Acceptance testing – Our group has no clientele that need to verify the app's functionality. Performance testing – We have no goals to perform at a certain level. We only need for our app to function at the minimum level which can be tested through systems testing.

**Who will perform such activities?**
Developers (Thomas, Jeremy, Katlin) will conduct unit testing as they code. Tester (Joshua) will conduct functional, integration & systems, and stress testing. Designer (Vitor) will conduct user-interface testing.

## 1.2 Test Selection

For the unit testing, the developers will use a mix of black-box and white-box techniques. We will rely on intuition testing as the code is developed and test each individual method for functionality after it is written. We will continue this mix of black-box and white-box techniques for functional testing. For stress testing, we will use error guessing and error-prone analysis to attempt to find any faults and see if the code will properly handle invalid user-input.
As we move on to systems testing, we will use path analysis testing, relying on Lab21, an automated testing tool. As stated later in Section 1.5, 21Labs has a section for UI testing, called "Test Steps". In this part of the app, you can choose a specific path you want to test and add all the screens associated with the path at the "Add Screen" section.

21Labs also has an automated self-learning AI system that adapts its tests to optimize bug reporting in the APK application. You can report bugs with a single click, and the dashboard of 21Labs keeps track of tests ran, and the overall success rate of your project. For example, a poorly designed app will have a worse test success rate than a well-designed one.

Unit Testing will be used to begin with UI testing to verify that the interface buttons and inputs are accurately representing what is being input into the interface. After the Unit Testing step is done, Functional Testing will be used as a whole to verify that the input is being directed at the correct pieces of code in the back end and is not being misread. Most of the Test Cases for the UI testing will be generated through Intuition by the Designer Vitor and Team Lead Katlin, as they are the closest to a clientele for the application, and also through Code, as a part of Path Analysis.

## 1.3 Adequacy Criterion

In order to ensure that the quality of our test cases is up to our standards, each test case's structural coverage of the code will be measured so that we can be sure that all the code involved in the project is tested thoroughly. In terms of how the coverage will actually be measured, 21Labs will be extremely helpful for this as there are various features which automatically analyze and report the coverage of tests. Overall, we will use intuition to ensure what should be what as we strive to be beyond adequate for those that use our application.

## 1.4 Bug Tracking

For our program, we strive to make changes to allow users the best experience possible. In order to do that, we will use a bug and enhancement request form (made in MS Forms or Google Form) that users will be able to use via a link in the actual mobile application under the information button on the main screen. This form will allow us to catch bugs that are found by users that ultimately limit the usefulness of the application or that are just annoying for users to deal with. The same form will also allow users to recommend feature enhancements that the team will take into consideration and implement if we feel that we agree with the user.

A test form for this will be linked via URL below:
https://forms.office.com/r/6Nva3PiNjC

## 1.5 Technology and Tools

For this project, we are going to be using an application called 21Labs.
This application is free and all you have to do is upload an APK file into the visual drop box, and it is all setup for testing. Its functionality is unique as 21 learns the structure of your application to know how to better run tests. It is kind of like the microphone on Google that understands your voice better as you continue to speak into it. It has automated bug reporting, and there is no setup required to start using it. You can also choose which individual paths in the app to test, and 21Labs has plenty of emulators to choose from so you can do just that. It has very few cons besides that the free version has limited testing ability compared to the paid version. However, it has tons of pros that make it worth even getting the free version.

## 2. Test Cases and Test results

**Pre-conditions:** Have application installed on Android mobile device

### Unit Testing: DBMS Testing

| Test Case | Purpose | Steps for Test | Expected Result | Actual Result | Pass/Fail | Additional Info |
|---|---|---|---|---|---|---|
| Add info to Recipe | To test if Recipe data can be added to the table RECIPE | 1: Create fake Recipe info<br>2: Execute code to add info to DB | Said Recipe info to be visible as a table entry | | | |
| Add info to Ingredient | To test if recipe data can be added to the table INGREDIENT | 1: Create fake Ingredient info<br>2: Execute code to add info to DB | Said Ingredient info to be visible as a table entry | | | |
| Edit Recipe | To test if Recipe data can be modified in the table RECIPE | 1: Create fake Recipe info<br>2: Execute code to update info to DB | Said Recipe info to be updated | | | |
| Edit Ingredient | To test if recipe data can be modified to the table INGREDIENT | 1: Create fake Ingredient info<br>2: Execute code to update info to DB | Said Ingredient info to be updated | | | |
| Delete Recipe | To test if Recipe data can be deleted from the table RECIPE | 1: Create fake Recipe info and add to RECIPE<br>2: Execute code to delete info from DB | Said Recipe info to be deleted from RECIPE and corresponding Ingredients to be deleted from INGREDIENT | | | |
| Delete Ingredient | To test if recipe data can be deleted from the table | 1: Create fake Ingredient info | Said Ingredient info to be deleted from INGREDIENT | | | |

| | | | | | | |
|---|---|---|---|---|---|---|
| | INGREDIEN T | 2: Execute code to delete info from DB | | | | |

## Functional Testing: Recipe Testing

| Test Case | Purpose | Steps for Test | Expected Result | Actual Result | Pass/Fail | Additional Info |
|---|---|---|---|---|---|---|
| Test addRecipe() | To test the method addRecipe() | 1: Create a new Recipe in source code.<br>2: Execute addRecipe given said new recipe | A recipe to be sent to the DBMS | | | |
| Test addStep() | To test the method addStep() | 1: Create a new recipe in source code.<br>2: Execute addStep() given made up data | A step to be added to the recipe | | | |
| Test editStep() | To test the method editStep() | 1: Create a new recipe with existing steps in source code.<br>2: Execute editStep() given made up data | An existing step to be edited correctly | | | |
| Test deleteStep() | To test the method deleteStep() | 1: Create a new recipe with existing steps in source code.<br>2: Execute deleteStep() given made up data | An existing step to be deleted | | | |

## Functional Testing: Ingredient Testing

| Test Case | Purpose | Steps for Test | Expected Result | Actual Result | Pass/Fail | Additional Info |
|---|---|---|---|---|---|---|
| Test newIngedient() | To test the method newIngre dient() | 1: Create Ingredient info.<br>2: Execute newIngredien t() given made up data | Ingredient object to be created | | | |

| | | | | | | |
|---|---|---|---|---|---|---|
| Test editIngredient() | To test the method editIngredient() | 1: Create an Ingredient in the source code.<br>2: Execute editIngredient() given made up data | Said Ingredient to be modified | | | |
| Test deleteIngredient() | To test the method deleteIngredient() | 1: Create an Ingredient in the source code.<br>2: Execute deleteIngredient(). | Said ingredient to be deleted | | | |

## Integration and Systems Testing

| Test Case | Purpose | Steps for Test | Expected Result | Actual Result | Pass/Fail | Additional Info |
|---|---|---|---|---|---|---|
| Application runs: Adding a Recipe | Overall ability for the mobile application to run | 1: Open application<br>2: Select Recipes ribbon tab<br>3: Select the + icon in the top-right corner<br>4: Input information in the text fields<br>5: Press the Add button | Application runs smoothly as user utilizes the adding a recipe function. | | | This test is a general test concerning the overall capability of the application. |
| Application runs: Adding an Ingredient | Overall ability for the mobile application to run | 1: Open application<br>2: Select Pantry ribbon tab<br>3: Select the + icon in the top-right corner<br>4: Input information in the text fields<br>5: Press the Add button | Application runs smoothly as user utilizes the adding an ingredient function. | | | This test is a general test concerning the overall capability of the application. |
| Application runs: Searching for Available Recipes | Overall ability for the mobile application to run | 1: Open application<br>2: Press on magnifying glass icon | Application runs smoothly as user utilizes the searching | | | This test is a general test concerning the overall capability of |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | for available recipes function. | | | the application. |
| Application runs: [Extra Features] | Overall ability for the mobile application to run | 1: Open application 2: ... | Application runs smoothly as user utilizes the app's functions. | | | This test is a general test concerning the overall capability of the application. |

## User-Interface Testing

| Test Case | Purpose | Steps for Test | Expected Result | Actual Result | Pass/ Fail | Additional Info |
|---|---|---|---|---|---|---|
| User can press button | UI Functionality | 1: Press Button 2: Record if produced result or not | Produces result, calls function related to button | | | This test case will be used for every interactable aspect of the UI |
| Interface Interaction calls appropriate code | UI Functionality | 1: Press Button/Input Text 2: Check if produced result or not 3: Check if result is sent to appropriate code 4: Check if code can read input | Input is sent to correct code and the data is the same as the input | | | This test case will mainly be used to double check the input of Recipes/Ingredients into databases is correct. |
| User can see item in database | UI Functionality | 1: Press button to see recipe/pantry 2: Record if proper database item is shown | Appropriate item from database is shown in proper view window | | | This test case will check if the stored input is able to be shown to the user |

**Note**: All table entries for "Actual Result" and "Pass/Fail" are yet to be filled in due to not conducting the full testing as of the time of this document.