Version 1.1

Fridge Filler Software Testing Plan

3/27/21

Team:
Shangyi Dai
Jonathan Derr
Travis Flake
Gage Gabaldon
Zhibang Qin

Project Sponsors:
Dr. Richard Rushforth
Sean Ryan

Mentor:
Sambashiva Kethireddy

Overview

This document will go over our testing plan for the application. Thus ensuring that the app is working as expected and what we are doing to minimize risks and potential problems.

Table of Contents

Introduction

Food banks have always had problems with getting food to families, and in recent times this demand is increasing even more. Even before the pandemic, Saint Mary's Food Bank and other food banks have had trouble making sure that food is not thrown away and wasted, which is caused in part by the uncertainty of what food will come through the food banks, meaning the distribution of food products is not uniform. For example, in the United States alone:

- Every year, 80 billion pounds of food are thrown away in the United States.
- Food banks across America receive different food products in varying quantities from donors throughout the country.
- Foodbank workers may be unfamiliar with products given to the food bank.

All of these factors create an inefficiency that is hard to solve with the changing supplies available to the food banks and with the lack of knowledge to best use the food.
This is where our sponsors, Richard Rushforth and Sean Ryan are looking to solve. Partnering with Saint Mary's Food Bank (SMFBA), They want to better feed the people the food bank is serving. The process of delivering and managing food for families all across the country is no easy task and one that is vital to continuing to help those in need.

This is where our group. Fridge Filler, comes in. Fridge Filler aims to create a mobile application that can serve as a go-between for the food banks and the people and families that use the food banks. That way, families can figure out what food to cook with the ingredients they have through simple, easy-to-follow recipes. The app also can recommend places to pick up missing ingredients and even a way to sync what you have at home with the app. The overall goal of the application is to help families effortlessly plan meals and to help ensure that the food at the distribution sites is used more effectively.

With this in mind, we are now in the testing stage of the project. Software testing is a process of making sure that the functions and components of the app are working as intended by testing the code that makes up the application. This is an important step of the software engineering process that can help reduce bugs going forward and to help the app stay fresh and bug-free.

The testing of the app is a multifaceted process. We will be conducting unit testing for all the page-per-page functionality in the app and for the API service. That way we can ensure at the smallest level that things are working as expected. Next, we will be conducting integration testing between the major components of the application to ensure that everything is communicating well and displaying correctly. The last step is to test the user experience when using the app to ensure that the app is working from a user perspective.

Starting with unit testing for all the function to function relationships and lower level tests. In the case of our app testing the pages in our mobile app and testing the backend helper functions in the backend portion. Splitting up the testing into two different parts for the front end and for the back end. This type of testing will all be about what's happening for that particular request or page elements.

Next comes integration testing which is for testing the connections between the components of our app. The big ones are the connection between the database, API, and mobile applications. As the connections between the API and the app are the lifeblood of the food filler project. As the integrating control what recipes will be shown, editing the recipes to show, adding users, and many others stuff that includes interacting between the different modules of the application.

Lastly, we want to do usability testing to test how the user uses the app and what portions are being used as intended. If the user wants to add to the pantry the app should add to the pantry and other core functionality.

This plan for testing focuses on unit tests, integration testing, and usability testing. This testing regime will ensure that every portion of the app will be covered from the individual pages and modules in the app to the communication between the API and the fridge filler app and lastly, the testing from an end-user perspective. To start with we will be going over the low-level testing, unit testing.

Unit Testing

What is Unit Testing?

In software development, unit testing is the testing of individual components or units of a program. The purpose of unit testing is to verify that “units” of code are working exactly as expected, in order to avoid high-cost defect fixing, as, without unit testing of much of the code, it will likely take more time for a developer to find problems that do not return errors when failing to perform correctly.

The Process

Unit testing in an Ionic-based application using the Angular Javascript framework is done using the Jasmine javascript testing module, which is a standalone testing framework that has a simple test writing syntax meant to verify the functionality of units tested. Tests developed using Jasmine are executed using a separate tool, the Karma test running, which we will be using to execute unit tests.

User Application Testing

General Testing:

For all pages of the application, we will be testing a few basic qualities to ensure the app will function properly.

- Page loads upon request: this request will verify that upon being redirected to a page, the application will change pages and attempt to gather elements to show the user. This will be verifying that the ionic app does not have any unresolved references in the PAGE_NAME.spec.ts files or the PAGE_NAME.ts files. For instance, if the login page spec.ts file was missing an import for the HttpClient node module, this test would fail, as it needs to import all modules referenced in the typescript files used.
- Page Elements render upon request: This test will verify that the elements on each tab of the app will show to users after the page has loaded, meaning that using the elements verified using the previous test, the app will now populate the current page with HTML elements correctly.

Login Page:

On the login page, we will need to verify that users will be able to log in, register, and sign in as guests

- Continue as Guest: we will unit test the continue as guest functionality by verifying that it properly bypasses the authentication system (it should return true for our auth service) and will set the stored user ID value to empty, to deny access to registered users only services. So if authentication passes AND the user ID is empty, this test will pass.
- Login: to test user login, we will be checking that a mock user will pass the authentication service when using a correct email address, will not pass if it is an invalid email address, as well as correct/incorrect password. We will also verify that the user ID is set to a nonzero number. As User ID should reset on logout, it will not pass if the user ID is stored as null, due to either bad login or failure to save user ID.
- Register: Verification of functionality of the API registering users correctly is best left to integration testing, so we will not test much for registration

other than the popup loading and closing upon submission of the user sign up, or canceling sign up.

Recipe Browser Page:
The recipe browsing page is largely reliant on the loading of recipes from the database, which is better verified in integration testing, but we will still test that several core elements are working using dummy recipes, namely, the Recipe modal loading and search features being properly interactable.

- Search Filter functionality: We will test to verify that the advanced search filtering modal properly loads when the openAdvancedSearch() function is called, so users can properly filter their recipe search results using the variety of options offered
- Recipe Modals: When you click on a recipe, the recipe will open in a new modal containing full recipe details. This unit test will verify that the on click function showRecipe() will provide the modal, and request the needed info.

User Pantry Page:
The user pantry page will be where users store their ingredients and the quantities that they own, intended for use with recipes. We will unit test addition, editing, and removal of ingredients from this page.

- Add Ingredient modal: When a user attempts to add ingredients, a modal will be provided. We will unit test to ensure that the modal contains all needed fields for the ingredient add query, and all of these are rendered for the user to see. We will also verify that upon closing the modal, the modal is unrendered, so the user can continue using the pantry page. This will be verified by using an onPageUnload() function provided in the page lifecycle functions that are provided by the ionic library.
- Edit Ingredients modal: Much like the add ingredients modal, we will want to verify the availability of the areas that users will be able to edit here, such as ingredient name, quantity, units, all rendering upon the edit modal being selected. This will be done in the same manner as the Add ingredient modal testing, using functions from the lifecycle of a page given to us by the ionic library.
- Delete Ingredient: The delete ingredient option will be verified by storing the ingredient ID upon deletion during the unit test, and checking after the delete was called to see if an ingredient of the same ID is currently present on the page. Should there still be an ingredient of the same ID, the test has failed to delete the entry. On success it should no longer exist.

My Recipes Page:
The my recipes page serves to allow users who have bookmarked recipes to view and print them. Functionality of viewing the recipes was tested in the recipe browser page, and the same function will be used here, so we will not test that the viewRecipe() function works again. Here we will unit test the pdf creation and download functionality

- Creation of PDF files: For recipe printing, basic recipe data is grabbed after the recipes are loaded onto the page using the recipe ID. this test will check the fields in the createPdf() function to verify that basic recipe elements i.e. recipeName, recipeSteps, etc are not null, as all recipes need these core elements. This test will pass if createPdf correctly generates this object and fills it out.
- Saving of PDF files: This function is intentionally being omitted from the unit testing portion of this app, as it does not return traditionally. It is best left to integration testing manually to verify that for each platform the PDF version is properly saved as a file upon users clicking the save pdf button, creating a file for them.

Navigation Page:

Unit testing of the navigation page is on hold temporarily as we shift the design of this page to focus more on integration of external maps, as Saint Mary's Food Bank does not wish to constantly upkeep this map of food distribution centers. Going forward we will still likely be capable of testing searchability functions to ensure they are properly finding elements within the map. This is not something we can verify testability of quite yet though, so this portion of the document will be revised once we know for sure (ideally the March 29th meeting will clear this up once and for all).

API Testing

General Testing:
Testing the API will be a bit different due to the fact that it's a persistent application hosted on the web. Thankfully, Firebase has provided some infrastructure for running unit tests on hosted functions in their project framework, so we just need to write the required tests and include it with the project files. Once deployed, these tests can be run anywhere as long as the user running the tests has the right credentials.

To test the API, internal functions will need to be tested with sample data to ensure that they work correctly and return the right data. The functions we will need to test are:

- sendQuery
- sendResult
- checkToken
- checkResult
- formatSortBy
- checkRequest
- formatLimit
- getRecipeData
- addIngredients

This will amount to about 18 tests, though it's worth considering that some of these are simple functions that exist only to avoid code repetition (such as formatLimit and formatSortBy), and will be easy to test.

We will be using Mocha.js as our testing framework, and as stated previously all tests will be included with the function code that is deployed to Firebase, allowing the tests to be run externally. Mocha tests are simple to write, are very human-readable, and provide effective, clear results. This will be of great benefit to us considering all the tests we will have to write.

Integration Testing

What is Integration Testing?

Integration Testing focuses on the interfaces between major modules and components and focuses on whether the interactions and data exchanges between modules take place correctly. In our case, the interfaces we will need to test are the integration between the API and the front end components (the mobile app and the admin portal), and the integration between the Front End and the external services we are using such as Firebase. The API acts as a bridge between the database and all the other components of the project, and while the Unit Testing will prove that the API itself works in a vacuum, additional tests will be necessary to make sure that the data is being received and used correctly on the Front End.

The Process

We will be using many of the same testing tools that we are using for Unit Testing, but rather than using sample data, we'll be trying to simulate real use of each component as much as possible. This means we'll be monitoring each service externally rather than internally, to ensure that all of the data being sent/received is correct. These tests will look very similar to the unit tests discussed in the previous section, but will have a much wider scope, and will use real data instead of purpose-designed sample data.

API Testing

To ensure the API is functioning correctly, we will need to test all request cases. This includes testing the four major request types:

- GET
- POST
- DELETE
- PUT

For each of the major resources:

- /user
- /recipe
- /pantry
- /box
- /userfav

At a minimum, this will result in at least 20 unit tests to write for API access, with some variation as some resources do not require the support of all methods. Then, we will need to test different search criteria for each request/resource combination that takes arguments in the URL (specifically, GET and DELETE requests), which will increase the number of required tests considerably. We will also need to write incorrect versions of each test, to ensure that they fail when given incorrect data.

Mobile Application Testing

When testing the mobile application we need to test the functionality between the API and the application is one of the major points to integration testing. As we want to ensure that the data we are given through the API is correct. That way when we display it to the user everything will be working.

API Integration

When testing the integration with the API we have a few core requests that we want to ensure are given us the right data. These requests are the pantry, user favorites, and recipe GET requests.

Pantry GET request:
We are testing that the data we are given from the API will contain the information we need to display to the user. The test will check if the json returned has the following fields for each item in the pantry.

- IngredientName
- ingredient Quantity
- ingredient Unit.

If the json given has the following information the request is valid and can be used to display the information to the user.

User Favorites GET request and Recipe Get request:
We are testing if the information received from the http client contains the information needed to display to the user. Since user favorites and recipes are the same information we can check if the json is valid and contains the following fields.

- recipeName
- cuisineType
- timeToMake
- numSteps
- Steps
- recipeRating
- numRatings
- ingredients.

Firebase Integration

When testing the firebase analytics we want to ensure that the events being logged are actually working and being sent to the server. The process is going to be different then the other test as to verify the test went correctly. The user must open up the firebase console and see that the event has been logged.

So when testing we make a test case that sends a test event to the firebase service. Then we check that the event has gone through in the firebase side.

Usability Testing

Unlike other testing mentioned throughout this document, usability testing will bring the end-user into the testing process. Rather than testing our code or module, the goal for usability testing is to get feedback on the user interface and user experience of our application. With the usability testing, we can use the testing result to make our application to be more user-friendly. The usability testing can also expose some bugs that unit or integration testing did not catch. For our project, usability testing is a significant testing phase, the target users for our application and website are not those with a lot of technical knowledge. Thus, our application is designed to create a comfortable and easy user experience for any target user.

For the usability testing with our application, we are planning to have at least two different testing phases. The first phase will be sessions with our teammates' choice which could range from ourselves to friends or family members. The second phase will be working with our clients, St. Mary's Food Bank, Dr. Richard Rushforth and Sean Ryan to get participants. For all of the participants in our usability testing, we will enlist the help of two different groups of people, these being students and people from St. Mary's Food Bank.

Our criteria for our participants is as follows:

- Access to an Android smartphone
  For testing our mobile application, participants will need to be able to download the mobile app.
- Access to a computer
  For testing our web application, participants will need to be able to access our web application by the browser.

To conduct our usability test and get the result of the test, we are planning to use Email or Zoom. With Email, we can accept almost any kind of type of result from participants. With Zoom, we can communicate with our participants directly, see how they are interacting with our application, and they can talk to us about their feelings through their interaction.

Mobile Application Testing

When we find our participants, we will send our participants a link to download our APK. Once they have our application on their phone, we will introduce them to what our application can do, the main functions of our application and any other necessary information they may need.

After the participants finish the testing, they can send their feedback through email or talk to us by using Zoom. When we get all the feedback from our participants, we will analyze the feedback and get the testing result.

Web Application Testing

When we find our participants, we will send our participants a link to our website where our web application is being hosted and allow them to have access to our web application. We will introduce them to what our application can do, the main function of our web page and other necessary information they may need.

After the participants finish the testing, they can send their feedback through email or Zoom if they want to talk to us. When we get all the feedback from our participants, we will analyze the feedback and get the testing result.

Admin Portal Testing

When we find our participants, we will send them a link to our website where our admin portal is being host. Also, we will introduce our participants to what the admin portal page can do, what the buttons can do on the page, how to edit data and other necessary information they may need.

After the participants finish the testing, they can send their feedback through email or Zoom if they want to talk to us. When we get all the feedback from our participants, we will combine the result with other testing and create a final result.

The goal of the timeline for our usability testing is to have the first testing completed before mid-April. With this, we will have the time to make changes to our application by using the feedback.

Conclusion

For the application, we will mainly test the function we designed for the app including the following: navigation to food bank and retailers, a searchable library for recipes, virtual pantry keeping track of what food user has and printable recipes for users and food bank employees. To make sure our app works, we have set up a user, recipe, pantry, food box and user fav database to store all the data we need for the application. To manage these databases, we have written an API that can handle four major HTTP requests: GET, POST, PUT and DELETE, which are what we need to perform unit and integration testing on. In the unit test, first, we tested on our mobile app to make sure our app works and all the backend data can be displayed without error. We will test login with a user account and log in as a guest. In these two different ways we have

different data sent to the back end and guests will have less function on other pages.
For the recipe page we will focus on testing the search function and recipe modals. For the pantry page we will test the management of ingredients. For the user recipe we will test the function for pdf. For the API function we will mainly test sendQuery, sendResult, checkToken, checkResult, formatSortBy, checkRequest, formatLimit, getRecipeData and addIngredients.
Then in the integration testing we will test the connection from database to the app through the API function. For the usability testing, we will have two different testing phases, one from our team and one from our clients for the mobile application and web application. They will get access to our app through the link of the web or link of the application. We will receive feedback from them through email and zoom.
With this we are confident that the app progress is going smoothly and will keep an optimistic mind for the future of the project.