Senior Design

ENG EC 463

Memo

To: Professor Pisano
From: Trevor Chan, Jeffrey Chen, Andres Garcia, Cole Wentzel
Team: 10
Date: 4/4/2024
Subject: FridgeBuddy - Final Testing Plan

1.0 Required Materials

Hardware:

1. Mobile phone (or emulator)
2. Tablet
3. Tablet housing

Software:

1. AWS Services
   a. IAM Users (allow code to access AWS services)
   b. Amplify (interface for connecting React Native to AWS services)
   c. DynamoDB (user and item information storage)
   d. Lambda (run Python scripts)
      i. Insert user item using 3rd-party UPC lookup API
      ii. Get recipe suggestions given ingredient lists
   e. AppSync (define how React Native interacts with DynamoDB and Lambda)
2. React Native (User interface)
3. Third-party APIs
   a. FoodCentral API
   b. Spoonacular Recipe API
   c. Clarifai API

2.0 Setup

The setup is divided into the two main components of the Fridge Buddy user interface: the Fridge Buddy mobile application and the physical refrigerator attachment running the same application. As a whole, the application is powered in the frontend by React Native, using Amazon Web Services (AWS) for its storage and computational backend.

The tests for the Fridge Buddy application are meant to demonstrate the main functionalities of the project that have been completed and the typical workflow of a user. This includes showcasing user authentication, CRUD operations on user items, adding items through barcode scanning and object recognition, expiration date prediction, and recipe suggestion features. The tests will be performed on a new user to demonstrate the user sign up and login process that includes email verification and is powered by AWS Cognito. From there, the tests will show possible usecases by the user (different methods of adding items, and using those items to generate recipes).

The user and item data are stored in the DynamoDB database backend. The React Native application can query and mutate the DynamoDB data because of a GraphQL schema that defines the DynamoDB entity data schema, as well as how the application can interact with DynamoDB and Lambda.

Barcode scanning and object recognition tests will also be performed on the FridgeBuddy application to demonstrate the alternate, automated methods for adding item information to the user's data. The application's barcode scanner was implemented using the expo-camera library. For barcode scanning, Fridge Buddy retrieves the item's UPC when a barcode is detected within the camera view, and for smartscanning, it uses the Clarifai API and a food item recognition pre-trained model to classify food items in the camera view. Both scanning methods call a Lambda function to search for item information (name, category, and calorie count) from the FoodCentral API. From there, barcode-scanned items use the Spoonacular API to retrieve generic product names from the branded name (e.g. 'apple' from 'granny smith apple') to be stored and used for recipe suggestions. Next, both scanners use the item names to get expiration date predictions from the FoodKeeper database based on how long the food item is expected to expire after the current date. Lastly, all of the gathered information is stored inside of the user's items in DynamoDB.

Recipe suggestion tests will also be performed to demonstrate the ways Fridge Buddy can help users utilize their item inventory. Recipe suggestion is handled through users either selecting specific or all of their items to be used to generate recipes. From there, a Lambda function is called that queries the Spoonacular API for information on the top 5 recipes that use as many of the inputted ingredients as possible, as well as each recipe's name, image (if applicable), calorie count, instructions, and ingredients.

The frontend user interface was developed using the React Native framework and TypeScript. It is run and deployed to the tablet / mobile device using the React Native developer build to bundle the application. The tests will show that the frontend application can accurately display the current user's inventory including item names, expiration dates, categories, quantities, and calorie counts (if the item has those fields), as well as showing that the item information is synchronized with the user's data stored in DynamoDB.

3.0 Pre-Testing Setup Procedure

Fridge Buddy application tests

1. Ensure that AWS services are available and React Native can connect to the AWS backend
2. Prepare test input barcodes and items
3. Build and package React Native application for both touchscreen unit and mobile device using developer build
    a. Connect the device to a Windows machine with the UI code via a USB cable
    b. Load the application onto the connected device
        i. Execute npx expo prebuild (if necessary)
4. Execute npx expo run:android

4.0 Testing Procedure

FridgeBuddy Application Tests:

1. Test that application successfully runs on tablet and mobile device
2. Test user sign-up, confirmation, and login process
3. Test that user can perform CRUD operations on items
4. Manually add items
    a. Input item fields for:
        i. Name
        ii. Expiration date
        iii. Category
        iv. Quantity
        v. Calories
    b. Manual add item date prompts with calendar pop-up
5. Test that user can generate recipes using selected items
6. Test that user can search and sort their existing items
7. Edit existing item
    a. Name
    b. Quantity
    c. Expiration date (prompts with calendar pop-up)
8. Remove existing item
9. Add items using UPC barcodes
    a. Busch's garbanzo beans can
    b. Bottle of milk
    c. Original kettle cooked potato chips

10. Add items using item recognition
    a. Banana
    b. Apple
    c. Pizza slice
11. Test generating recipes using new items
12. Test signing out and signing into a different user
13. Mobile Application tests
    a. Test that mobile application can query user items
    b. Test that mobile application can add item that shows up on tablet
14. Test that tablet housing fits onto tablet and can mount on magnetic surface

5.0 Measurable Criteria

The criteria for successful running and output are as follows:

1. User is able to successfully create a user, verify the user using email confirmation, and log in to their user account
2. User is able to query correct item information from DynamoDB
3. User is able to display correct item information from DynamoDB
4. User is able to manually add items on application
5. User is able to input expiration date using calendar prompt
6. Manually added items and inputted fields show up on DynamoDB
7. User is able to edit items on application
8. User is able to remove items from user interface (X) button, and items are removed from DynamoDB
9. Edited item fields are updated in on DynamoDB
    a. Name
    b. Quantity
    c. Expiration date
10. Items entered that are expired or soon to be expired visually notify the user (highlighted in red)
11. User is able to sort items by expiration date, name, quantity, or calories, in ascending or descending order
12. User is able to search current items
13. User is able to generate relevant recipes using selected items within 20 seconds
14. User is able to scan barcodes using the tablet or mobile application's camera (barcode scanner)
    a. UPC information is displayed once scanned
    b. After the UPC code is scanned, item information is added to user's item within 5 seconds
15. User is able to scan objects using the tablet or mobile application's camera (smart scanner)

    a. Possible items are displayed to be selected
    b. Once items are confirmed, item information is added to user's items within 5 seconds
16. Changes made to the mobile application appear on the tablet once refreshed (within < 2 seconds)
17. Mobile application displays changes made during the tablet tests (upon refresh) and is able to repeat same functionalities
18. User is able to add items to the mobile application
19. Tablet housing fits securely around tablet and does not obscure camera or USB port

<u>6.0 Score Sheet</u>

FridgeBuddy Tablet Tests		
Test #	**Functionality Working**	**Correct? (Y/N)**
1	User is able to create a user, verify that user, and login	Y
2	User is able to query correct item information from DynamoDB within 2s	Y
3	User is able to display correct item information from DynamoDB	Y
4	User is able to manually add items on application using Add Items field and + button	Y
5	User is able to input expiration date using calendar prompt	Y
6	Manually added items and inputted fields show up on DynamoDB	Y
7	User is able to edit items on application using Edit button	Y
8	Edited item fields show up on DynamoDB (Name, quantity, and expiration date)	Y
9	Items entered that are expired or soon to be expired visually notify the user	Y
10	User is able to generate relevant recipes using selected items within ≤ 20 seconds	Y

11	User is able to remove items from user interface (X) button	Y
12	Removed items are deleted from DynamoDB	Y
13	Mobile application displays changes made during the tablet tests	Y
14	User is able to add items to the mobile application	Y
15	Changes made to the mobile application appear on the tablet once refreshed (within < 2 seconds)	Y
16	Tablet is able to comfortably fit in tablet housing and attach to magnetic surfaces	Y
Result		15 / 15

FridgeBuddy Barcode Scanning Tests							
Test #	**UPC**	**Name**	**Category**	**Calories**	**Exp. date***	**Correct Info (Y/N)?**	**Appears in Database (Y/N)?**
1	021130070107	Whole Milk	Milk	67.0	1 week	Y	Y
2	039400017080	Garbanzo Organic Low Sodium Beans, Garbanzo	Canned & bottled beans	9	5 years	Y	Y
3	020685000294	Original Kettle Cooked Potato Chips, Original	Chips, pretzels & snacks	500	2 months	Y	Y
Result						3 / 3	3 / 3

FridgeBuddy Food Item Recognition Tests						
Test #	**Name**	**Category**	**Calories**	**Exp. date***	**Correct Info (Y/N)?**	**Appears in Database (Y/N)?**
1	Banana	Bananas	85.0	3.15 days	Y	Y
2	Apple	Apples	0.0	6 weeks	Y	Y
3	Pizza	Pizza	235.0	4 days	Y	Y
Result					3 / 3	3 / 3

FridgeBuddy Recipes Generated						
Test #	**Inputs**	**Name**	**Ingredients**	**Steps**	**Calories**	**Correct (Y/N)?**
1	recipeID = 662665 search = ["apple"]	'Baked Cinnamon Apple Slices'	{'apple', 'muesli', 'yogurt', 'milk'}	['Put Muesli in a bowl', 'Pour milk to cover the muesli and mix in yoghurt', 'Cover and refigerate for a few hours or overnight', 'When ready to eat, if the mixture is too thick, loosen with milk.Grate apple and stir into the muesli mix', 'Eat']	363.34	Y
2	recipeID = 655477 search = ["cheese", "peas"]	'Peas And Tarragon With Fresh Goat Cheese'	{'tarragon', 'parsley', 'sauce', 'mint', 'peas', 'goat cheese'}	['Blend goat cheese and tarragon in medium sauce pot.', 'Bring to a gentle simmer.', 'Add peas and simmer until thoroughly heated about 5 minutes.', 'Serve in a casserole dish.', 'Garnish with fresh chopped parsley.', 'Replace tarragon with mint.']	37.6	Y
3	recipeID = 991625 search = ["milk", "egg", "onion", "green beans"]	'Nutella Buttercrea m Cupcakes with Hidden Cadbury Egg'	{'cake mix', 'egg', 'vanilla', 'cooking oil', 'cupcakes', 'milk'}	['Preheat oven to 350 degrees.', 'Grease 6 jumbo muffin tin.', 'Combine cake mix milk, vanilla and oil. Mix till combined.', 'Add eggs beat till mixed well. Bake for 18- 21 min', 'After cooking cupcakes use a spoon and cut out enough area to insert Cadbury egg.', 'Making sure the smaller part of the egg is facing up.', 'Frost the cupcake hiding the Cadbury egg.']	241.29	Y
Result						3 / 3

7.0 Conclusion

Each of our tests ran successfully. The React Native application on both the mobile device and the tablet demonstrated the correct working functionalities that have been completed. This included showcasing user authentication, CRUD operations on user items, adding items through barcode scanning and object recognition, expiration date prediction, and recipe suggestion features. This means that our application has successfully been connected to the AWS cloud backend and 3rd party APIs, as well as having the frontend styling and layout largely finished. Because of this, the majority of the project has been completed, with the remaining steps being to polish what has already been completed and to work on reach goals.

Currently, the Clarifai API call is done directly from our front-end code for the Smart Scanner page. To improve the security and overall design of our application, we will be creating an additional Lambda function that handles the Clarifai API call from the backend. Another design improvement we will make is to switch the Smart Scanner page to use the react-native-vision-camera library instead of the expo-camera library for camera and image handling. We recently made this change for the Barcode Scanner page due to expo-camera's barcode scanning component not being functional when compiling and packaging our app into a development build using Gradle. The react-native-vision-camera library also provides easy implementation of other camera features such as the pinch-to-zoom gesture. Therefore, we will be modifying the Smart Scanner to use this library for consistency of design.

The original functional requirements of the project specified in previous reports are described in the following table:

Requirement	**Value, range, tolerance, units**
Attachment setup time	< 5 minutes
Touchscreen stability	Unit stays mounted with ≤ 15 lbs of force exerted on refrigerator door
Cross-platform interface	Compatible with iOS and Android systems
Size	< 13.8in (35cm) x 9.8cm(25cm) x 3.1cm (8cm)
Weight	< 5lb (2.27kg)
Data synchronization	< 2 minutes
Food recognition	90% accuracy
Text recognition	90% accuracy
Automatic inventory input	< 10s after scanning
Item storage	≥ 200 items
Item removal	≤ 1 button press per item
On-screen list updating	< 3s
Recipe generation	≥ 2 recipes, < 10s
Total component cost	≤ $250
Power cord cross-section	< 0.25in^2 (1.6cm^2)

Power cord stability	Cord stays plugged in with ≤ 15 lbs of force exerted on refrigerator door

So far, we have accomplished most of these specifications and almost all of the minimum viable product. There are some remaining polishes to the existing design, such as testing that the development build works (so that users at ECE day can test putting it on their phone), testing the build with iOS (since the testing thus far has largely been Android-based), exploring adhesive alternatives to the current magnet setup for attaching the tablet, and updating the housing to prevent side-to-side movements of the tablet while secured in the housing. We are also working on possibly getting the app published on the Apple App Store or Google Play Store for ease of user access.

Some of the existing reach goals that we want to be completed by ECE day include a notification system for letting users know when their devices are going to expire soon, allowing for searching for recipes by name, and improving application performance. We plan for the notification system to work via push notifications when items reach a certain threshold of expiration date (e.g. 5 days until it expires or if it has expired). For the recipe search functionality, we have implemented the search bar and have the function for getting recipes by name, so the only steps remaining are to set up a Lambda function and connect it to GraphQL. Lastly, we noticed during the testing that our cheap tablet's low resources make the app run slightly slow (e.g. slight delay between pressing on an item and selecting it on the UI). To improve the app performance, we want to work on optimizing our existing code to put less of a strain on the hardware.