FoodNut Implementation Breakdown

Logan Farrow, Emily Brouillet, Melak Senay, Jose Gutierrez, Mitchell

API's Used:

- Open Food Facts Database
    - While creating FoodNut, the open food facts database has faced numerous outages requiring us to switch api links. If you encounter loading issues please let us know so we can check in the Open Food Facts slack for server outages. Please also note that this is a public open source project, and perfect information is not 100% available!
    - We use the open food facts API for a variety of different functions throughout the application, the main use of the APIs to get all of the food data including nutrition score, NOVA (preservative score) and additives. Specifically, we have a list of additives that will get flagged if they are found within your food and you'll see a warning given the data from the API. Since the API did not explicitly name the preservatives (it instead had a code for each preservative), we had to manually create a String of Strings and map each code to the correct additive.
    - The open food facts database also gives a nutrition score, and we store the information about what a nutrition score is so that users can click on it in the pop-up table, and understand what the score means.
    - We were also able to have users click on the NOVA score, so they could understand what their food is made out of.
    - Using this API, we were able to cover all of our deliverables set at the beginning of the project for the end-user.

Backend Used:

- Firebase
    - FireStore, FireStorage, FireAuth
    - This app uses FirebaseAuth to handle critical components such as logging in and logging out and managing storage for our users.
    - We also designed our own Schema that holds information about our users by mapping their firebase authentication ID to a collection of data that includes favorites and username. This was done using Firebase FireStore
    - We set up Firebase to also allow for edits to the documents as the users update their favorites and gave them the option to update their username
    - We also initialize Firebase FireStorage to hold our default profile picture as backup when the app is used.
- UserDefaults
    - User defaults are used to store recent scans to show on the home screens on the device.
    - User defaults are also used to store data on the categories of each product that is scanned given by the API to present them as a total scans made on that device in a graphical format on the homescreen.

ViewController Breakdowns:

1. SignUp
    - Data Collection
        - Users are prompted to create an account after pressing the ‘get started’ button on the splash screen
        - Users can pick their own identifying username, enter their email, and choose a password
            - Passwords must be over 6 characters to be considered strong
        - Multiple accounts may not use the same email
        - If the user already has an account, there’s an option to login instead, where the user is only prompted to enter email and password
        - The changes between creating an account and logging in are reflected in the ViewController appropriately(button labels and label text is altered, the name text field is removed from the view)
        - After pressing create account/login the user is brought directly to the home screen
    - Login / Logout Swapping
        - The user can seamlessly switch between logout and login modes without pushing a new view controller due to hiding and text reformatting of values on the signup page
        - Both processes are easily handled by Firebase so that users can log into different accounts and still track their favorites across devices
    - Warnings
        - Warnings are displayed to the user if they try to sign up with wrongly formatted emails, emails already associated with an account, or weak typed passwords
2. Home
    - Welcome User or Guest
        - If a user is logged out, the phrase ‘Welcome, Guest’ is displayed at the top of the view. If a user is logged in, the user is greeted with their username, which is fetched from firebase using the unique userID associated with their account.
    - Chart Breakdown of All Scans
        - A dynamic pie chart is displayed that breaks down the broad categories of the user’s most recent scans. The placement and design of this addition was extremely intentional as it’s convenient for the user to see their breakdown as soon as they open the app and the pie chart organization is easily readable, users can compare the different categories almost instantly.
        - When a user scans a new product the chart is automatically updated
            - The organizational category of the item is retrieved and the current pie chart is searched for items of that category. If the category is present in the pie chart, the count for that category is incremented and the change is reflected in the chart. If none is found, the category is added to the pie chart and a color is randomly generated to represent that category on the wheel.

        - The object category counts and their colors are correlated using dictionaries, and these are stored on one's device eternally. This allows users to gauge their overall consumption trends over time, allowing users to make adjustments to their eating habits accordingly in the interest of balance or healthy choices.
    - This pie chart was created using the DGCharts package
- Recent Scans
    - The lower half of the screen displays a horizontally scrolling collection view that shows the user images of their most recent scans. The images are fetched from the API, with a default image set up. Because this collection view displays 2-3 cells to be seen on the screen at any time, it is not necessary for the images to be cached in this specific view.
    - These recent objects are stored in userDefaults, and the three most recent scans are displayed. This choice promotes a lifecycle in the recently scanned objects, as after a certain amount of time their old scans become irrelevant and objects that are scanned often would be more efficiently stored and accessed in favorites. Thus, this collectionView is a convenient way for users to revisit what they've most recently scanned.

3. Search
    - Search Bar & API Call
        - Users can search for a product based on product name
        - The API will return a list of photos and titles for different products. Each of those can be clicked on to pull up a detailed view
    - Collection View
        - API results of the search are displayed as Collection View Cells with a Image View and Text Label within each Cell
        - Each cell hosts a picture of the product along with it's name
    - Detailed view controller
        - Users can click on an item to pull up the detailed view of its healthiness. they can use this information to select what product they would like to add to their favorites
    - All images are cached

4. Camera + Details ViewController
    - If you scan an item that is not supported by the open food facts API you will get a message about it the same goes for barcodes and instructions on how to reset your camera to scan again.
    - The details view controller is also a table view controller so that when you click on either the NutriScore or the NOVA score, you learn more information about it
    - The details of the detailed view controller makes calls out to the open food facts API to populate all the categories to give you the best perspective about how healthy your food is
    - There are preset, default, values in case any value is not returned by the API
    - All handlers for the camera and API calls are thread safe and handled in their own threads

5. Favorites
    - Firebase Storage of Favorites
        - Every user has its own firebase list of favorites, which are pulled down into the favorites list on viewed load.
    - Image Caching
        - The images are cached on the first load of favorites, so that it does not have to keep constantly reloading the images. Every time you click away from that view.
    - Detailed View Saved
        - The detailed view for each of the favorites is also saved so that it can be pulled up whenever a favorite is clicked
        - Users can sign in and out and then favorites will load based on which account is signed in
6. Profile
    - Every user has a default profile photo under which their account username is displayed.
    - The account username is pulled from Firebase and displayed under the profile photo.
    - Users have the option to change this by pushing an ‘update profile’ button, which pushes a new view controller.
    - When changing their username a user must correctly input their current username, which is checked against the one stored in Firebase. If they match, the username is updated correctly in all ViewControllers that display the user's name(favorites & profile), and this change is reflected in Firebase.
    - Features a logout button option, which returns the user to the onboarding screen and prompts the user to log in again or create an account and updates the API listener so it is aware no user is logged into the app.