

FOOD SCANNER APP-SNAP FIT

PMH



FROM: KRITIKA BISSESSUR & RANDRIANARIJAONA FY FALINA

TO: MR SHIAM BEEHARRY

26TH February 2024

Table of Contents

1	Introduction	1
2	Methodology.....	2
3	Project overview	3
3.1	Design & Implementation	4
3.1.1	Wireframe of Snap Fit	4
3.1.2	Storyboard of Snap Fit	5
3.2	Prototype of Snap Fit	6
3.2.1	Overview	6
4	Activities Overview	11
4.1	Home Activity	11
4.2	Scanner Activity	12
4.3	Upload Image Activity	13
4.4	Product Details Activity	14
4.5	The Saved Product Activity	15
4.6	Barcode Scanning.....	16
5	Training the model	17
6	Dart Main Section of Codes	20
6.1	Barcode Scanning Function:.....	20
6.2	Image Recognition Functions.....	21
6.3	Data Retrieval Functions	25
6.4	CRUD operation functions	27
7	Weakness and strength	32
8	Conclusion	33
9	Reference	34

List of Figures

Figure 3.1: Wireframe of Snap Fit	4
Figure 3.2: Storyboard of Snap Fit	5
Figure 3.3: Overview	6
Figure 3.4: Overview (2)	6
Figure 3.5: Scan a product	7
Figure 3.6: Scanning	8
Figure 3.7: Product example	9
Figure 3.8: Search Page	10
Figure 4.1: Home Activity	11
Figure 4.2: Scanner activity	12
Figure 4.3: Upload Image Activity	13
Figure 4.4: Product Details Activity	14
Figure 4.5: The Saved Product Activity	15
Figure 5.1: Barcode Scanning	16
Figure 5.2: Preparing the Dataset	17
Figure 5.3: Training the model	17
Figure 5.4: Training the model (2)	18
Figure 5.5: Integration with the App	18
Figure 5.6: CRUD Operations using Sqflite	19
Figure 6.1: Barcode Scanning Function	20
Figure 6.2: _loadModelAndLabels()	21
Figure 6.3: _getImageAndProcess()	22
Figure 6.4: _classifyImage(File image)	23
Figure 6.5: getRemoteProduct	25
Figure 6.6: _fetchSearchResults()	26
Figure 6.7: get database	27
Figure 6.8: initDatabase	28
Figure 6.9: getSavedProducts	29
Figure 6.10: removeProduct	29
Figure 6.11: updateProductName	30
Figure 6.12: saveProduct	30

1 Introduction

In collaboration with my colleague, Fy Falina, for our PMH course project, we've developed a groundbreaking application named Snap Fit. This innovative tool serves as a comprehensive solution for scanning food products and retrieving detailed ingredient information via the OpenFoodFast API. Within this report, we aim to explore Snap Fit's functionalities, challenges encountered during development, and its promising prospects.

At the core of our food scanner app lies its primary objective: to empower users with convenient and efficient access to crucial information about the food products they consume. Leveraging advanced technologies such as barcode scanning and image classification, users can swiftly retrieve essential data including product names, ingredients, country of origin, and more. Moreover, Snap Fit offers additional features such as the ability to save scanned products for future reference, edit product details, and even upload images for enhanced identification purposes.

Throughout this report, we'll delve into a comprehensive examination of Snap Fit, covering its key features, implementation intricacies, the challenges we faced during its development, and the exciting possibilities for future enhancements. By shedding light on Snap Fit's potential impact on food transparency and consumer awareness, we hope to underscore its significance as a pioneering solution in today's ever-evolving landscape of nutrition technology.

2 Methodology

The development methodology behind Snap Fit was meticulously structured to ensure a seamless and effective process from inception to deployment. It commenced with a comprehensive analysis of project requirements, incorporating input from stakeholders and conducting an in-depth assessment of user needs. This initial phase laid the groundwork for subsequent research and planning, wherein our team delved into existing food databases, APIs, and relevant technologies to inform decisions regarding the app's architecture and features.

Armed with a thorough understanding of the project scope and objectives, the design phase prioritized crafting an intuitive user interface focused on ease of use and accessibility. Employing wireframing, mockups, and iterative feedback sessions, we refined the app's layout, navigation flow, and interaction patterns. Continuous collaboration with stakeholders and end-users ensured that design decisions aligned with evolving needs and preferences, resulting in a user-centric app experience.

Snap Fit's development was executed using the Flutter framework, leveraging its cross-platform capabilities to deliver a consistent user experience across various devices. Embracing a modular and agile development approach, we fostered flexibility, collaboration, and rapid iteration. Features were implemented incrementally, enabling frequent testing and validation to ensure compliance with project requirements and user expectations.

In summary, our methodology for developing Snap Fit underscored a commitment to collaboration, adaptability, and the relentless pursuit of delivering a high-quality product that exceeds user expectations. Through a structured and iterative approach, we navigated the complexities of app development with precision and effectiveness, culminating in the successful creation of Snap Fit – a cutting-edge food scanner application poised to transform the way individuals interact with their food environment.

3 Project overview

Snap Fit stands at the forefront of a new era in dietary management, offering a transformative solution to the modern challenge of maintaining a balanced diet amidst busy lifestyles and overwhelming food choices. With the rise of processed foods and dietary misinformation, individuals are increasingly seeking tools that empower them to make informed decisions about their nutrition. Snap Fit emerges as a beacon of innovation, leveraging cutting-edge technology to provide users with a seamless and intuitive platform for managing their dietary intake.

Unlike traditional dietary tracking apps, Snap Fit redefines the user experience by harnessing the power of image recognition and barcode scanning. With a simple snap of a photo or a quick scan of a barcode using their smartphone camera, users gain instant access to comprehensive nutritional information about a wide range of food products.

The development of Snap Fit was guided by a commitment to excellence and innovation. Extensive research into existing food databases, APIs, and technologies ensured that Snap Fit leverages the most advanced tools and resources available. The design process focused on creating a sleek, intuitive, and user-friendly interface that prioritizes ease of use and accessibility for users of all backgrounds.

With Snap Fit, our vision is to empower individuals to take control of their nutrition and transform their relationship with food. By providing easy access to accurate and reliable nutritional information, Snap Fit empowers users to make informed choices that support their health and well-being. In a world where dietary decisions can feel overwhelming, Snap Fit offers a beacon of clarity, guidance, and empowerment, helping users navigate their journey towards a healthier, happier lifestyle.

3.1 Design & Implementation

3.1.1 Wireframe of Snap Fit

The wireframe of our food scanner app encapsulates a user-friendly interface designed to streamline the process of identifying and cataloging food items. At the forefront of this design is the snap-fit feature, which enables users to effortlessly capture images of various food products using their device's camera. The wireframe illustrates a sleek and intuitive camera interface, allowing users to align the snap-fit feature with the desired food item. Once the image is captured, the app initiates a seamless scanning process, leveraging advanced image recognition algorithms to identify the food product accurately. The wireframe delineates clear visual cues and feedback mechanisms to guide users through the snapping process, ensuring a smooth and efficient experience.

Additionally, the wireframe encompasses a comprehensive display layout for presenting scan results and relevant information to the user. Upon successful identification, the app showcases detailed nutritional data, ingredient lists, and possible allergens associated with the scanned food item. The wireframe delineates an organized and visually appealing presentation of this information, featuring distinct sections for essential details and interactive elements for further exploration. Furthermore, the wireframe incorporates intuitive navigation controls, allowing users to easily browse through scan history, manage saved items, and access additional features of the app. Overall, the wireframe of our food scanner app prioritizes user-centric design principles, emphasizing simplicity, efficiency, and accessibility in every aspect of the user experience.

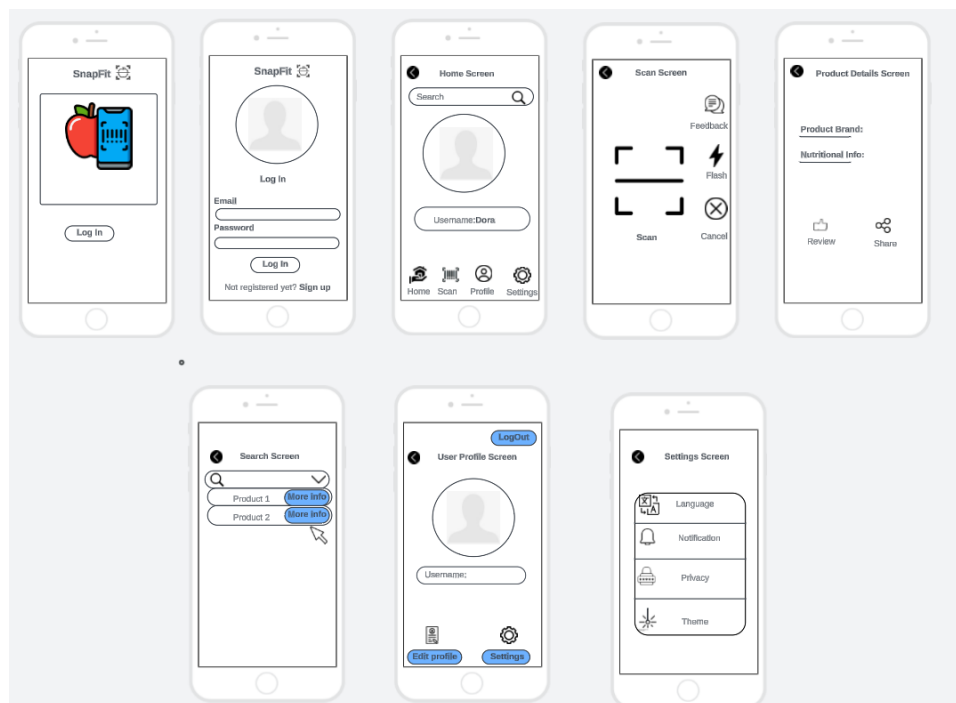


Figure 3.1: Wireframe of Snap Fit

3.1.2 Storyboard of Snap Fit

The storyboard for the snap-fit feature in our food scanner app follows a health-conscious shopper exploring the condiment aisle of a grocery store. They encounter a new brand of mayonnaise and are intrigued but cautious about its origins. The user opens the app on their smartphone and selects the snap-fit feature. Holding the phone steady, they aim the camera at the mayonnaise jar, ensuring it fits within the snap-fit guidelines displayed on the screen. With a quick tap, the user captures the image, and the app swiftly analyzes it. Within moments, the app not only identifies the mayonnaise but also provides detailed provenance information, including where the ingredients were sourced and how the product was manufactured. Additionally, the app presents nutritional information, such as calories, fat content, and allergen warnings. Satisfied with the transparency provided by the app, the user adds the mayonnaise to their shopping list, confident in their ability to make informed and ethically conscious choices about their purchases.

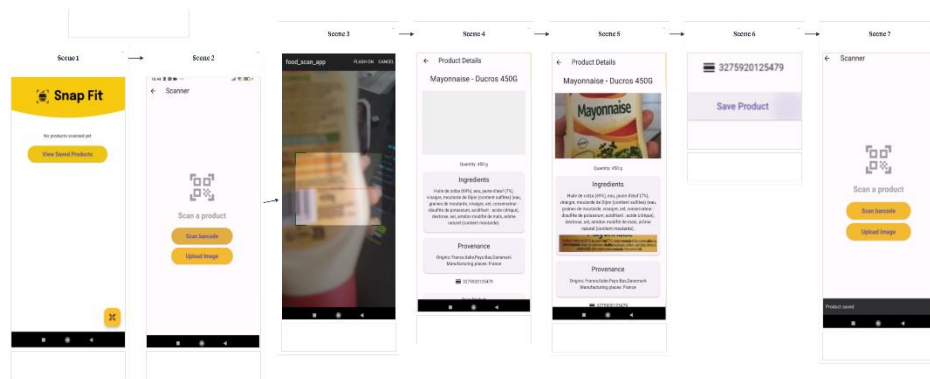


Figure 3.2: Storyboard of Snap Fit

3.2 Prototype of Snap Fit

3.2.1 Overview

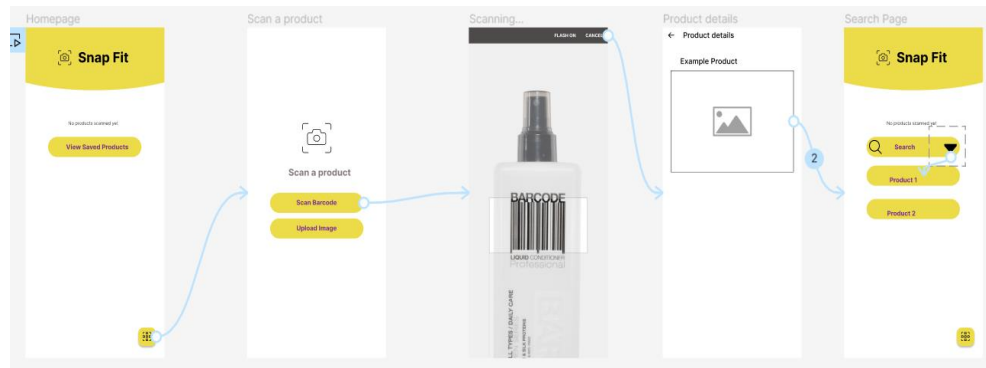


Figure 3.3: Overview

1. Home Page:

- The home page serves as the central hub of the app, welcoming users and providing easy access to key features. It features a clean and intuitive layout with the prominent scan icons to scan the products and also you can view the saved products
- There's also a scan logo at the top which illustrates the purpose of our app in short

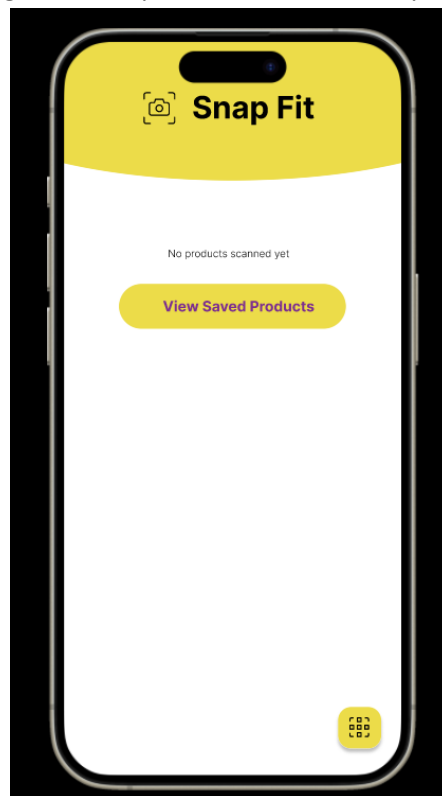


Figure 3.4: Overview (2)

2. Scan a Product:

- The "Scan a Product" feature enables users to scan barcodes or images of food products using their device's camera. It provides clear instructions and guidance on how to properly align the camera for optimal scanning.
- Once a product is successfully scanned, users are seamlessly directed to the product details page for further information.

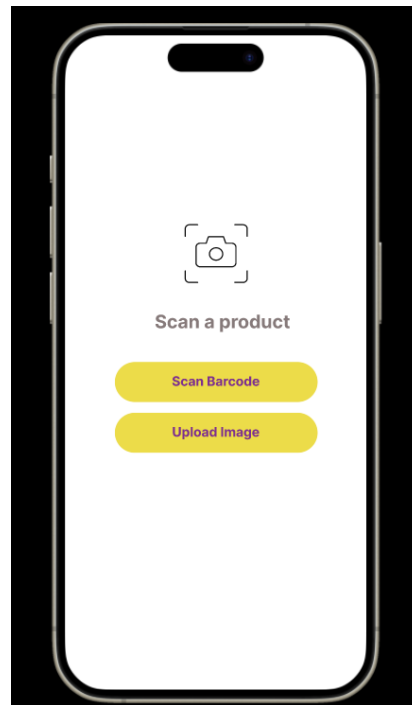


Figure 3.5: Scan a product

3. Scanning:

- During the scanning process, the app utilizes advanced image recognition and barcode scanning technology to identify the scanned product accurately and efficiently.
- In cases where the scanned product cannot be identified, the app may suggest similar alternatives or provide options for manual entry to accommodate various scenarios.



Figure 3.6: Scanning

4. Product Details:

- The product details page offers comprehensive information about the scanned product, including nutritional facts, ingredient lists, allergen warnings, and provenance details.
- Clear and visually appealing layouts present information in a structured manner, making it easy for users to digest and understand.
- Interactive elements such as buttons for saving to favorites



Figure 3.7: Product example

5. Search Page:

- The search page allows users to quickly find specific products or categories of interest. It features a prominent search bar with predictive text or auto-complete functionality to assist users in finding what they're looking for efficiently.
- Filters and sorting options may be available to help users refine their search results based on criteria such as nutritional content, dietary preferences, or price.

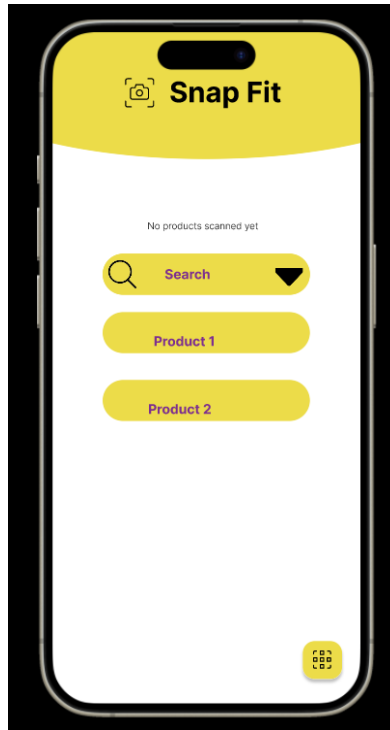


Figure 3.8: Search Page

4 Activities Overview

Activities are the building blocks of an Android app and are used to present different views and handle user input. Each activity represents a distinct screen that the user can interact with.

4.1 Home Activity

Home Activity is the initial screen of the app, where users can view the recently scanned product, view the saved product or start a new scanning process.

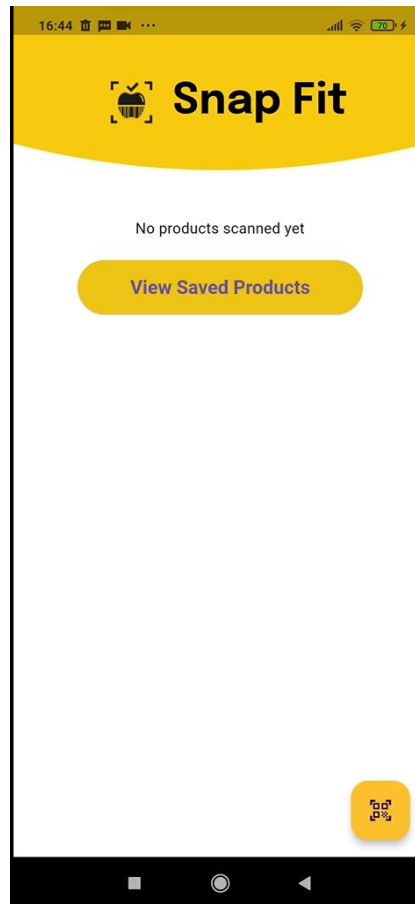


Figure 4.1: Home Activity

4.2 Scanner Activity

The scanner activity allows users to choose between scanning the product barcode or upload an image from the gallery.

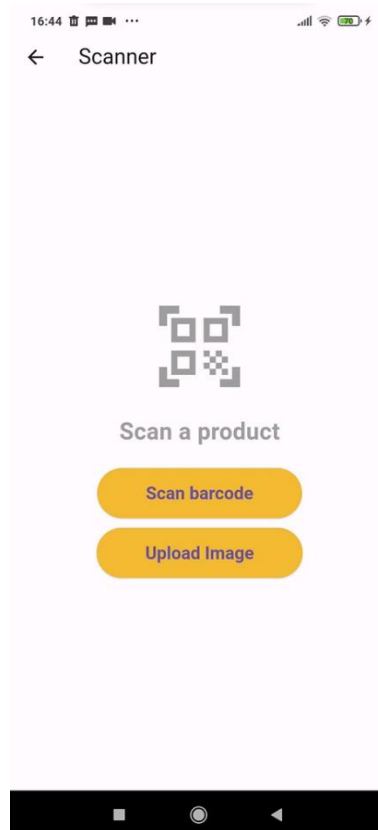


Figure 4.2: Scanner activity

4.3 Upload Image Activity

The Upload Image Activity allows users to pick image from the user's gallery for image recognition.

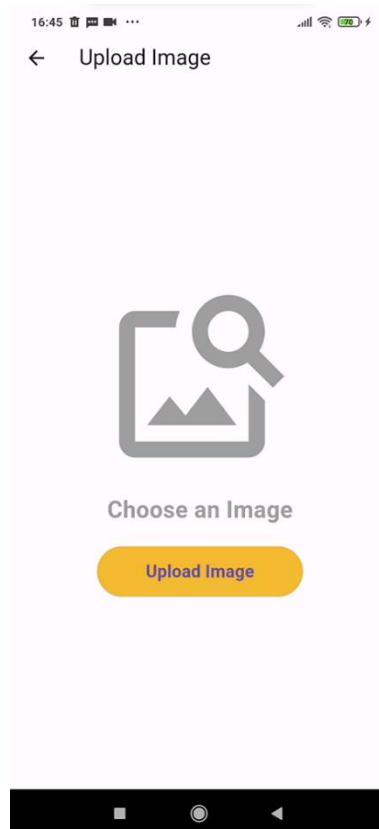


Figure 4.3: Upload Image Activity

4.4 Product Details Activity

The Product Details Activity presents the retrieved information about a food product, including ingredients, origin, and any additional details. Users can view and analyze the information on this screen.



Figure 4.4: Product Details Activity

4.5 The Saved Product Activity

The Saved Product presents all the product the user saved after scanning. It allows the user to edit the product name or remove it from the saved product list.

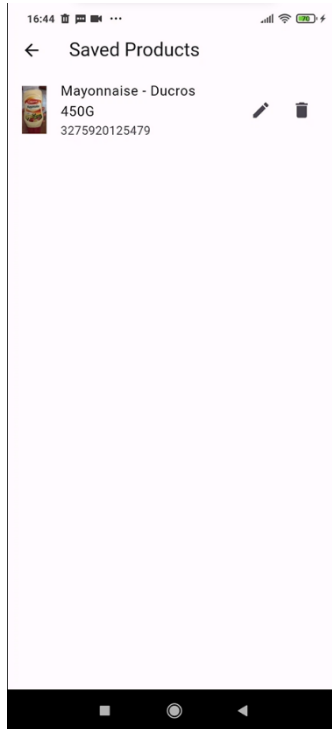


Figure 4.5: The Saved Product Activity

4.6 Barcode Scanning

Implementing barcode scanning in the app typically involved choosing a barcode scanning library compatible with flutter and dart:

barcode_scan2

A flutter plugin for scanning 2D barcodes and QR codes.

This provides a simple wrapper for two commonly used iOS and Android libraries:

- iOS: <https://github.com/mikebuss/MTBBarcodeScanner>
- Android: <https://github.com/dm77/barcodescanner>

Features

- Scan 2D barcodes
- Scan QR codes
- Control the flash while scanning
- Permission handling

Figure 4.6: Barcode Scanning

The barcode scanning focuses on integrating a reliable barcode scanning library and implementing the necessary code to handle camera input, barcode detection, and information retrieval.

5 Training the model

1. Image Recognition

Implementing image recognition in the app implied:

a. Preparing the Dataset

The first step was to find a dataset containing food product images annotated with their corresponding name. Annotated Food image datasets are limited on the web, and some of them concerned imported products we do not have in Mauritius. That's why we chose to use a simple dataset for testing purpose with accessible products.



Figure 5.1: Preparing the Dataset

b. Training the Model

We are using Teachable Machine, as seen in the lab sessions, to train the model based on the chosen dataset.

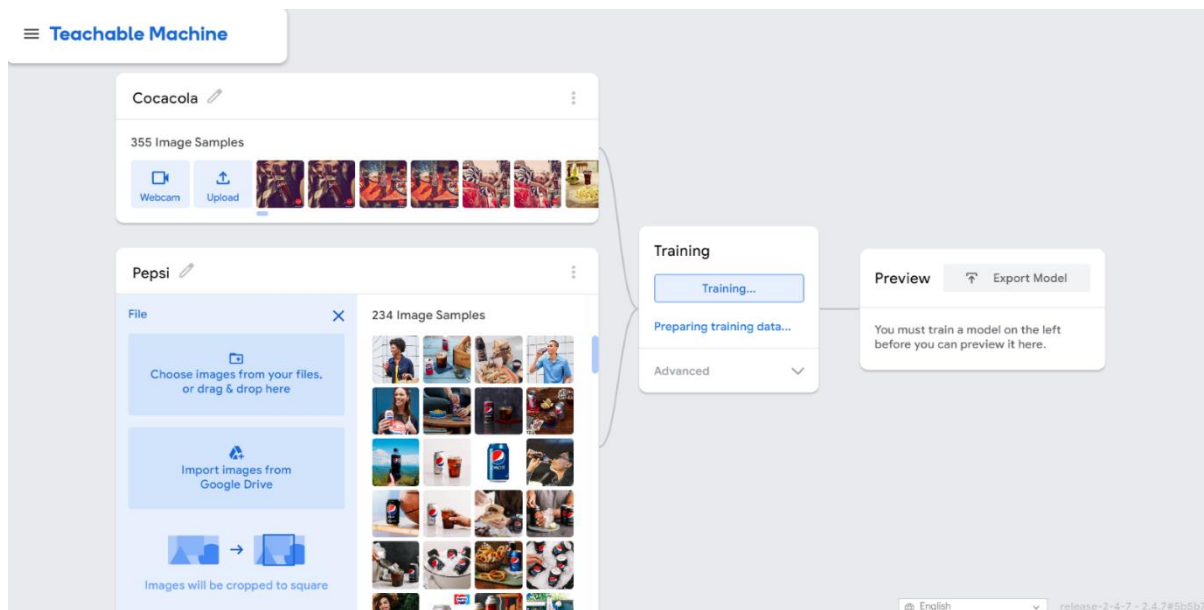


Figure 5.2: Training the model

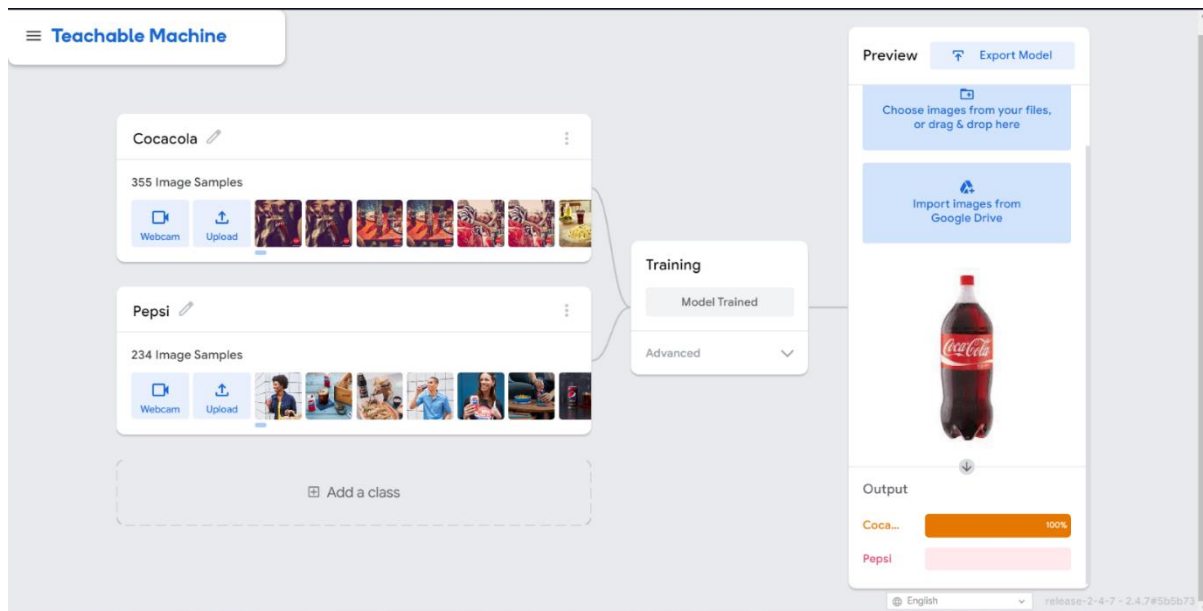


Figure 5.3: Training the model (2)

c. Integration with the App

We integrated the trained model into our mobile app using the framework TensorFlow Lite.

tflite_flutter

572 130 98%
LIKES PUB POINTS POPULARITY

TensorFlow Lite Flutter plugin provides an easy, flexible, and fast Dart API to integrate TFLite models in flutter apps across mobile and desktop platforms.

v 0.10.4 (2 months ago) tensorflow.org Apache-2.0 Dart 3 compatible

SDK FLUTTER PLATFORM ANDROID IOS LINUX MACOS WINDOWS

API results: ▶ tflite_flutter_method_channel/tflite_flutter_method_channel-library.html

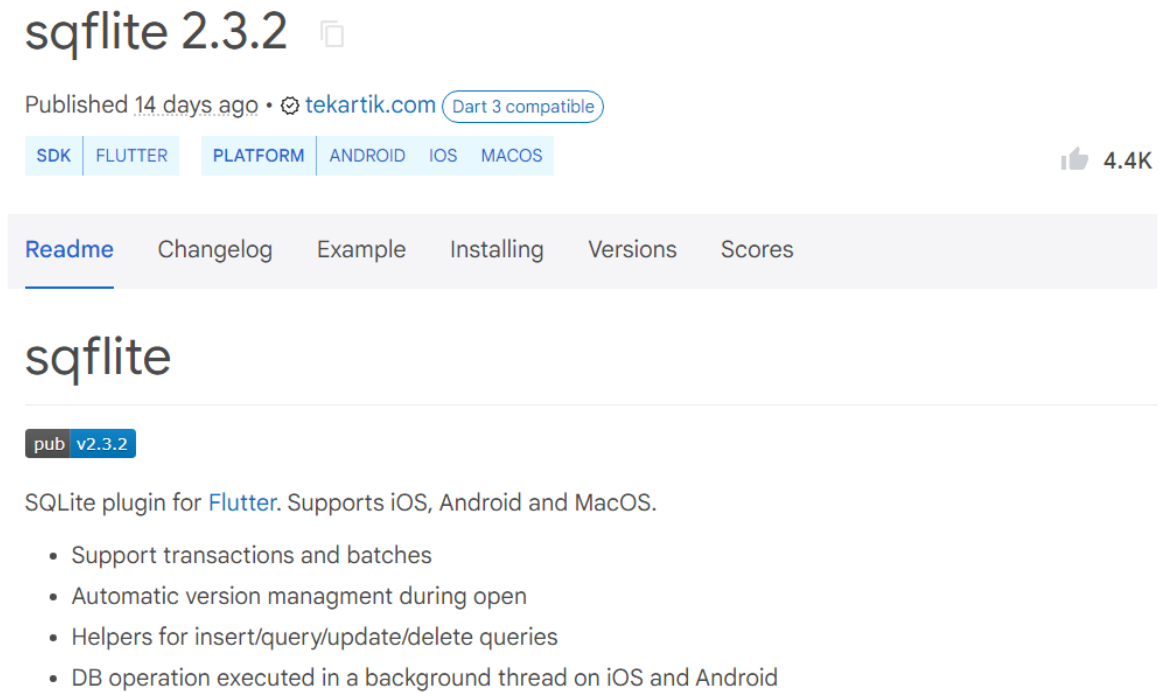
Figure 5.4: Integration with the App

The image recognition methodology involves data preparation, model training, and integration with the mobile app, to ensure accurate and efficient identification of food products from images.

2. CRUD Operations using Sqlite

The CRUD (Create, Read, Update, Delete) operations using Sqlite involves designing the database schema, integrating Sqlite.

To be able to implement the necessary code to perform CRUD operations on the database we are using the sqlite flutter dependencies



The screenshot shows the pub.dev page for the sqflite package. At the top, it displays 'sqflite 2.3.2' with a copy icon. Below this, it states 'Published 14 days ago' and lists the publisher 'tekartik.com' with a 'Dart 3 compatible' badge. A row of tags includes 'SDK', 'FLUTTER', 'PLATFORM', 'ANDROID', 'IOS', and 'MACOS'. On the right, there is a thumbs up icon and '4.4K' likes. A navigation bar contains links for 'Readme', 'Changelog', 'Example', 'Installing', 'Versions', and 'Scores'. The 'Readme' section is active, showing the package name 'pub v2.3.2' and a description: 'SQLite plugin for Flutter. Supports iOS, Android and MacOS.' Below the description is a bulleted list of features.

sqflite 2.3.2

Published 14 days ago • [tekartik.com](#) Dart 3 compatible

SDK | FLUTTER | PLATFORM | ANDROID | IOS | MACOS

4.4K

[Readme](#) | [Changelog](#) | [Example](#) | [Installing](#) | [Versions](#) | [Scores](#)

sqflite

pub v2.3.2

SQLite plugin for [Flutter](#). Supports iOS, Android and MacOS.

- Support transactions and batches
- Automatic version management during open
- Helpers for insert/query/update/delete queries
- DB operation executed in a background thread on iOS and Android

Figure 5.5: CRUD Operations using Sqlite

6 Dart Main Section of Codes

The Dart main section of code contains the main logic and functionality of the Food Scanner app. It includes the implementation of barcode scanning, image recognition, and data retrieval.

6.1 Barcode Scanning Function:

This code snippet demonstrates a method that triggers the barcode scanning process, retrieves the scanned barcode's raw content, and navigates to the Product Details screen in the app to display the scanned product's information.

```
lib > scan.dart > _ScanScreenState > scanBarcode  
26 class _ScanScreenState extends State<ScanScreen> {  
27   Future<void> scanBarcode(BuildContext context) async {  
28     try {  
29       var result = await BarcodeScanner.scan();  
30       Navigator.pushNamed(context, '/product', arguments: result.rawContent);  
31     } on PlatformException catch (e) {  
32       if (e.code == BarcodeScanner.cameraAccessDenied) {  
33         // Handle camera access denied  
34         print("Camera Access Denied");  
35       } else {  
36         // Handle other errors  
37         print("Error: $e");  
38       }  
39     }  
40   }  
}
```

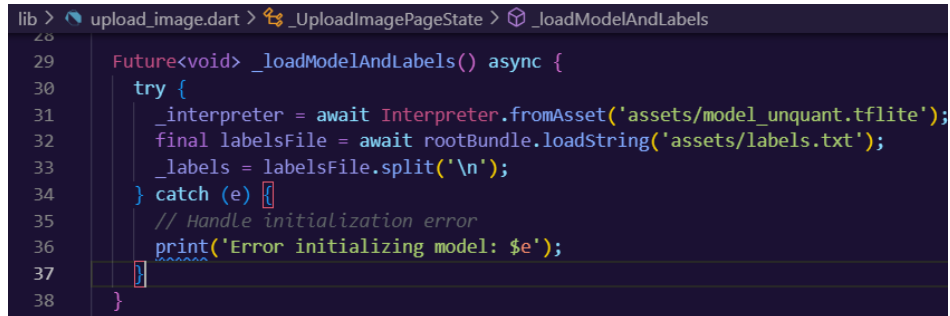
Figure 6.1: Barcode Scanning Function

The scanBarcode function is an asynchronous function that initiates the barcode scanning process in the Food Scanner app. It uses a barcode scanning library to scan the barcode and retrieve its raw content. If the scanning is successful, it navigates to the /product screen with the raw content as an argument. If there is an exception, it checks if it is due to camera access denial or another error and handles it accordingly.

6.2 Image Recognition Functions

We used image recognition technology to analyse the uploaded image and identify the food product. It compares the image features with a pre-trained database of food product images to determine the closest match.

a. `_loadModelAndLabels()`:



```
lib > upload_image.dart > _UploadImagePageState > _loadModelAndLabels
29 Future<void> _loadModelAndLabels() async {
30   try {
31     _interpreter = await Interpreter.fromAsset('assets/model_unquant.tflite');
32     final labelsFile = await rootBundle.loadString('assets/labels.txt');
33     _labels = labelsFile.split('\n');
34   } catch (e) {}
35   // Handle initialization error
36   print('Error initializing model: $e');
37 }
38 }
```

Figure 6.2: `_loadModelAndLabels()`

This function:

- Loads the machine learning model from an asset file (`model_unquant.tflite`) into the app.
- Loads the labels for the model's output classes from a text file (`labels.txt`).
- If an error occurs during initialization, it is caught and handled by printing an error message.

b. `_getImageAndProcess()`:

```
lib > upload_image.dart > _UploadImagePageState > _getImageAndProcess
40 Future<void> _getImageAndProcess() async {
41   setState(() {
42     _isProcessing = true;
43   });
44
45   try {
46     final pickedFile = await _picker.pickImage(source: ImageSource.gallery);
47     if (pickedFile != null) {
48       setState(() {
49         _image = File(pickedFile.path);
50       });
51       await _classifyImage(_image);
52     } else {
53       setState(() {
54         _isProcessing = false;
55       });
56     }
57   } catch (e) {
58     // Handle image picking error
59     print('Error picking image: $e');
60     ScaffoldMessenger.of(context).showSnackBar(
61       SnackBar(
62         content: Text('Error picking image: $e'),
63         behavior: SnackBarBehavior.floating), // SnackBar
64     );
65     setState(() {
66       _isProcessing = false;
67     });
68   }
69 }
```

Figure 6.3: `_getImageAndProcess()`

This function:

- Sets the `_isProcessing` flag to true, indicating that the app is currently processing an image.
- Allows the user to pick an image from the gallery using the image picker.
- If an image is selected, it sets the `_image` variable to the selected file and calls the `_classifyImage()` function to classify the image.
- If no image is selected, it sets the `_isProcessing` flag back to false.

c. `_classifyImage(File image)`:

```
lib > upload_image.dart > _UploadImagePageState > _classifyImage
71 Future<void> _classifyImage(File image) async {
72   if (_interpreter == null) return;
73
74   try {
75     var input = image.readAsBytesSync();
76     var output = List.filled(1, List.filled(5, 0.0));
77     _interpreter!.run(input, output);
78
79     // Assuming output is a list of labels with probabilities, pick the label
80     var predictedLabelIndex =
81     | | output[0].indexOf(output[0].reduce((a, b) => a > b ? a : b));
82     var predictedLabel = _labels[predictedLabelIndex];
83
84     setState(() {
85       _isProcessing = false;
86     });
87
88     // Send the predicted label to SearchResultsPage
89     Navigator.push(
90       context,
91       MaterialPageRoute(
92         builder: (context) => SearchResultsPage(productName: predictedLabel),
93       ), // MaterialPageRoute
94     );
95   } catch (e) {
96     // Handle classification error
97     print('Error classifying image: $e');
98     ScaffoldMessenger.of(context).showSnackBar(
99       SnackBar(
100         content: Text('Error classifying image: $e'),
101         behavior: SnackBarBehavior.floating), // SnackBar
102     );
103     setState(() {
104       _isProcessing = false;
105     });
106   }
107 }
```

Figure 6.4: `_classifyImage(File image)`

This function:

- Checks if the model interpreter is initialized (`_interpreter`) and returns if it's not.
- Reads the selected image file (`image`) as bytes.
- Runs the image through the model interpreter and obtains the output probabilities of the model's predicted labels.
- Picks the label with the highest probability as the predicted label.
- Sets the `_isProcessing` flag to false.
- Navigates to the `SearchResultsPage` with the predicted label as a parameter.
- If an error occurs during image classification, it is caught and handled by printing an error message and displaying a snack bar with the error message.

These functions handle the loading of the model and labels, image selection, image classification, and navigation to the search results page based on the predicted label. Error handling is implemented to display error messages when necessary.

6.3 Data Retrieval Functions

These functions retrieve the relevant information about a food product based on the barcode or the identified image. It communicates with an external API or database to fetch the necessary data.

a. `getRemoteProduct(BuildContext context, String barcode):`

```
lib > get_remote_product.dart > getRemoteProduct
6 // Creating a Dio instance with options.
7 var dio = Dio(BaseOptions(
8   baseUrl: "https://world.openfoodfacts.org/api/v0/",
9   connectTimeout: const Duration(milliseconds: 5000),
10  receiveTimeout: const Duration(milliseconds: 5000),
11 )); // BaseOptions // Dio
12
13 /// Fetches product information from the remote API based on the provided barcode.
14 /// Throws a [DioException] if the request fails or if the response is not as expected.
15 Future<Product> getRemoteProduct(BuildContext context, String barcode) async {
16   log.d("Getting product $barcode");
17   String url = "product/$barcode.json"; // The base URL is set in Dio options.
18
19   try {
20     Response response = await dio.get(url);
21
22     // Check if the response data is not null and contains the expected fields.
23     if (response.data != null && response.data["status"] != null) {
24       return Product.fromJson(response.data);
25     } else {
26       throw DioException(
27         requestOptions: RequestOptions(path: url),
28         response: response,
29         error: "Invalid response format",
30       ); // DioException
31     }
32   } on DioException catch (e) {
33     // Handle Dio errors by displaying an error screen.
34     errorScreen("Can not download product info: ${e.message}",
35       context: context);
36     rethrow; // Rethrow the DioError to indicate failure.
37   }
38 }
```

Figure 6.5: `getRemoteProduct`

The code snippet provided demonstrates a function for retrieving product information from a remote API using the Dio HTTP client library. Let's explain its functionality:

- The code initializes a Dio instance with specified options, including the base URL and connection timeouts.
- The `getRemoteProduct` function takes a `BuildContext` and a `String barcode` as parameters and returns a `Future<Product>`. It retrieves product information from the remote API based on the provided barcode.
- Inside the function, a log message is printed to indicate that the product with the given barcode is being retrieved.

- The URL for the API request is constructed by appending the barcode to the base URL.
- The function then sends a GET request using the Dio instance and the constructed URL.
- If the request is successful, the function checks if the response data is not null and contains the expected fields. If so, it parses the response data into a Product object using the fromJson method and returns it.
- If the response data is null or does not contain the expected fields, the function throws a DioException with a descriptive error message.
- If an exception of type DioException is caught during the request, the function handles the error by displaying an error screen using the errorScreen function and rethrows the exception to indicate the failure.

b. `_fetchSearchResults()`:

```
lib > search_result.dart > _SearchResultsPageState > _fetchSearchResults
31 Future<void> _fetchSearchResults() async {
32   String url = "search.pl?search_terms=${widget.productName}&json=1";
33
34   try {
35     Response response = await dio.get(url);
36
37     // Parse the response and extract search results
38     if (response.statusCode == 200 && response.data != null) {
39       setState(() {
40         _searchResults = (response.data['products'] as List)
41           .map((json) => Product.fromJson(json))
42           .toList();
43         _isLoading = false;
44       });
45     } else {
46       throw Exception('Failed to load search results');
47     }
48   } catch (e) {
49     print('Error fetching search results: $e');
50     setState(() {
51       _isLoading = false;
52     });
53   }
54 }
```

Figure 6.6: `_fetchSearchResults()`

The provided code snippet demonstrates a function `_fetchSearchResults` that retrieves search results from the "Search.pl" website based on a provided product name. Let's break down its functionality:

- The function is asynchronous and does not return a value (void).
- It constructs a URL by appending the `widget.productName` (assumed to be the product name) to the base URL and including the query parameter `json=1`.
- Inside a try-catch block, the function sends a GET request to the constructed URL using the Dio instance.
- If the response's status code is 200 (indicating a successful request) and the response data is not null, the function parses the response data to extract the search results.
- The search results are obtained by mapping the data's 'products' field (assumed to be a list of product JSON objects) to a list of Product instances using the `Product.fromJson` method.
- The `_searchResults` and `_isLoading` variables are updated with the extracted search results and the loading status, respectively, using the `setState` method.
- If the response status code is not 200 or the response data is null, the function throws an exception with the message 'Failed to load search results'.
- If an error occurs during the request or parsing, it is caught in the catch block. The error message is printed, and the `_isLoading` variable is updated to indicate that the loading process is complete.

6.4 CRUD operation functions

The `DatabaseHelper` class provides static and instance methods to handle CRUD operations for a SQLite database. It includes methods to initialize the database, retrieve saved products, remove a product, update a product's name, and save a product. These methods interact with the underlying database using the `sqlite` library.

a. **get database:**

```
lib > database_helper.dart > DatabaseHelper > database
6   static Database? _database;
7
8   static Future<Database> get database async {
9     if (_database != null) {
10      return _database!;
11    }
12
13    // If _database is null, initialize it
14    _database = await initDatabase();
15    return _database!;
16  }
```

Figure 6.7: get database

- This is a static getter method that returns a `Future<Database>`.
- If the `_database` instance variable is not null, it immediately returns the existing `_database` instance.
- If `_database` is null, it initializes the database by calling the `initDatabase` method and assigns the result to `_database`.
- Finally, it returns the `_database` instance.

b. **initDatabase:**

```
lib > database_helper.dart > DatabaseHelper > initDatabase
17
18 static Future<Database> initDatabase() async {
19   String path = join(await getDatabasesPath(), 'products_database.db');
20
21   return openDatabase(
22     path,
23     onCreate: (db, version) {
24       return db.execute('''
25         CREATE TABLE products(
26           id INTEGER PRIMARY KEY,
27           productFound INTEGER,
28           barcode TEXT,
29           name TEXT,
30           image TEXT,
31           ingredients TEXT,
32           quantity TEXT,
33           nutritionImage TEXT,
34           ingredientsImage TEXT,
35           origins TEXT,
36           manufacturingPlaces TEXT
37         )
38       ''');
39     },
40     version: 1,
41   );
42 }
```

Figure 6.8: `initDatabase`

- This is a static method that returns a `Future<Database>`.
- It generates the path for the database file using `getDatabasesPath()` and appends a filename (`products_database.db`).
- It then calls `openDatabase` to create or open the database at the specified path.
- The `onCreate` callback is provided to execute a SQL statement that creates a `products` table with various columns.
- The method returns the opened database.

c. **getSavedProducts:**

```
lib > database_helper.dart > DatabaseHelper > getSavedProducts
43
44 Future<List<Product>> getSavedProducts() async {
45     final Database db = await database;
46     final List<Map<String, dynamic>> maps = await db.query('products');
47     return List.generate(maps.length, (index) {
48         return Product.fromMap(maps[index]);
49     }); // List.generate
50 }
```

Figure 6.9: getSavedProducts

- This is an instance method that returns a Future<List<Product>>.
- It retrieves the database instance by calling the database getter.
- It uses the query method of the database to retrieve all rows from the products table and store the result in a list of Map<String, dynamic>.
- It then maps the list of maps to a list of Product instances by calling Product.fromMap for each map.
- The method returns the list of Product instances.

d. **removeProduct:**

```
lib > database_helper.dart > DatabaseHelper > removeProduct
51
52 static Future<void> removeProduct(Product product) async {
53     final db = await database;
54     await db.delete(
55         'products',
56         where: 'barcode = ?',
57         whereArgs: [product.barcode],
58     );
59 }
```

Figure 6.10: removeProduct

- This is a static method that takes a Product object as a parameter and returns a Future<void>.
- It retrieves the database instance by calling the database getter.
- It uses the delete method of the database to remove a row from the products table where the barcode matches the product's barcode.
- The method does not return anything.

e. **updateProductName:**

```
lib > database_helper.dart > DatabaseHelper > updateProductName
59     }
60
61     static Future<void> updateProductName(String barcode, String newName) async {
62         final db = await database;
63         await db.update(
64             'products',
65             {'name': newName},
66             where: 'barcode = ?',
67             whereArgs: [barcode],
68         );
69     }
```

Figure 6.11: updateProductName

- This is a static method that takes a barcode and a new name as parameters and returns a Future<void>.
- It retrieves the database instance by calling the database getter.
- It uses the update method of the database to update the name column of a row in the products table where the barcode matches the provided barcode.
- The method does not return anything.

f. **saveProduct:**

```
lib > database_helper.dart > DatabaseHelper > saveProduct
69     }
70
71     static Future<void> saveProduct(Product product) async {
72         final Database db = await database;
73
74         await db.insert(
75             'products',
76             product.toMap(),
77             conflictAlgorithm: ConflictAlgorithm.replace,
78         );
79     }
```

Figure 6.12: saveProduct

- This is a static method that takes a Product object as a parameter and returns a Future<void>.
- It retrieves the database instance by calling the database getter.
- It uses the insert method of the database to insert or replace a row in the products table with the values from the Product object.
- The conflictAlgorithm parameter is set to ConflictAlgorithm.replace to handle conflicts by replacing existing rows with the new values.
- The method does not return anything.

7 Weakness and strength

Snap Fit showcases significant strengths in its smooth integration with the Open Food Fact API, ensuring seamless communication and access to nutritional data. The application's barcode scanning feature is particularly noteworthy for its exceptional speed and versatility, even capable of accurately recognizing barcodes presented upside down. Users also benefit from the convenience of efficient product management capabilities within the app, enabling them to easily save, modify, and organize product information. However, the application faces occasional challenges with image recognition, which may impact the accuracy of product identification. Additionally, while the product detail page retrieves information from the API, there is a noticeable absence of in-app analysis features. This limitation results in the lack of functionalities such as calorie calculation or allergen flagging, which could greatly enhance the overall user experience and utility of the application.

8 Conclusion

In conclusion, the development journey of Snap Fit has been a dynamic blend of learning, overcoming challenges, and seizing opportunities. While we encountered setbacks, particularly in refining the image classification aspect, we've established a robust foundation for an impactful food scanning application.

Snap Fit epitomizes our dedication to enhancing food transparency and empowering consumers to make informed decisions about their food choices. By harnessing technologies like barcode scanning and image classification, our aim is to narrow the information gap between consumers and comprehensive product details.

Looking ahead, we recognize the imperative for continuous refinement and enhancement. Tackling challenges such as classification errors demands ongoing collaboration, research, and experimentation. Additionally, we aspire to broaden Snap Fit's capabilities by integrating features like allergen detection and nutritional analysis, thereby delivering even greater value to users.

In summation, Snap Fit transcends being merely an application—it stands as a testament to our commitment to innovation, consumer empowerment, and the pursuit of a more transparent and informed food industry. We eagerly anticipate the continued evolution of Snap Fit and its potential to catalyze meaningful change in the way we engage with food products.

9 Reference

<https://www.kaggle.com/datasets/serhanayberkkl/pepsi-cocacola-images>

<https://world.openfoodfacts.org/files/api-documentation.html>

<https://pub.dev>

<https://app.milanote.com/1RyKHx1GJnnK8m?p=7Mx6FEsVzAG>

<https://www.figma.com/file/qlIwiqFq4wNGbl0WhHW7IK/scanning-app?type=design&node-id=0%3A1&mode=design&t=9KU27HGxNmH9K5MV-1>

https://lucid.app/lucidspark/48093320-26fc-4f50-8ca8-71e42523af76/edit?viewport_loc=1274%2C-189%2C5480%2C2480%2C0_0&invitationId=inv_3eda4bf4-95f7-43fd-a1fc-43bb7c525b25