Cairo University
Faculty of Computers and Artificial Intelligence
Software Engineering

Cosmo Care

Supervised by

Dr. Khaled Wassef
Dr. Rasha El-Banna

Graduation Project Dissertation by:

20216027	*Bassma Khaled Mohamed*
20216148	*Mirna Hatem Aly*
20216108	*Nada Osama Ahmed*
20216144	*Nadine Hazem Alaa*
20206012	*Toqa Ahmed Moataz*

Graduation Project
Academic Year 2023-2024
Project Documentation

Table of Contents

LIST OF Figures

List of abbreviations

IDE	Integrated Development Environment
UI	User Interface
UX	User Experience
CI	Continuous Integration
CD	Continuous Deployment
HW	Hardware
SW	Software
IOS	IPhone Operating System
USB	Universal Serial Bus
RAM	Random-Access Memory
USB	Universal Serial Bus
AI	Artificial Intelligence
Info	Information
ERD	Entity Relationship Diagram
GUI	Graphical User Interface
API	Application Programming Interface

Chapter 1: Introduction

Introduction to the main area of the project

1.1 Motivation

Our mobile app is designed to help patients know more about the skin care cosmetics, find out what the kind of their skin is and identify specific skin concerns. The app serves as a comprehensive platform, offering users a convenient and personalized experience in exploring, selecting, and purchasing beauty items. It aims to streamline the skincare and cosmetics routine, providing tailored recommendations based on individual preferences, skin types, and budgets. The benefits encompass enhanced user satisfaction, time efficiency, and informed decision-making. By leveraging the mobile app, consumers can access expert advice, virtual try-on features, and a seamless shopping journey, ultimately promoting self-care and confidence through optimal cosmetic product choices.

1.2 Problem definition

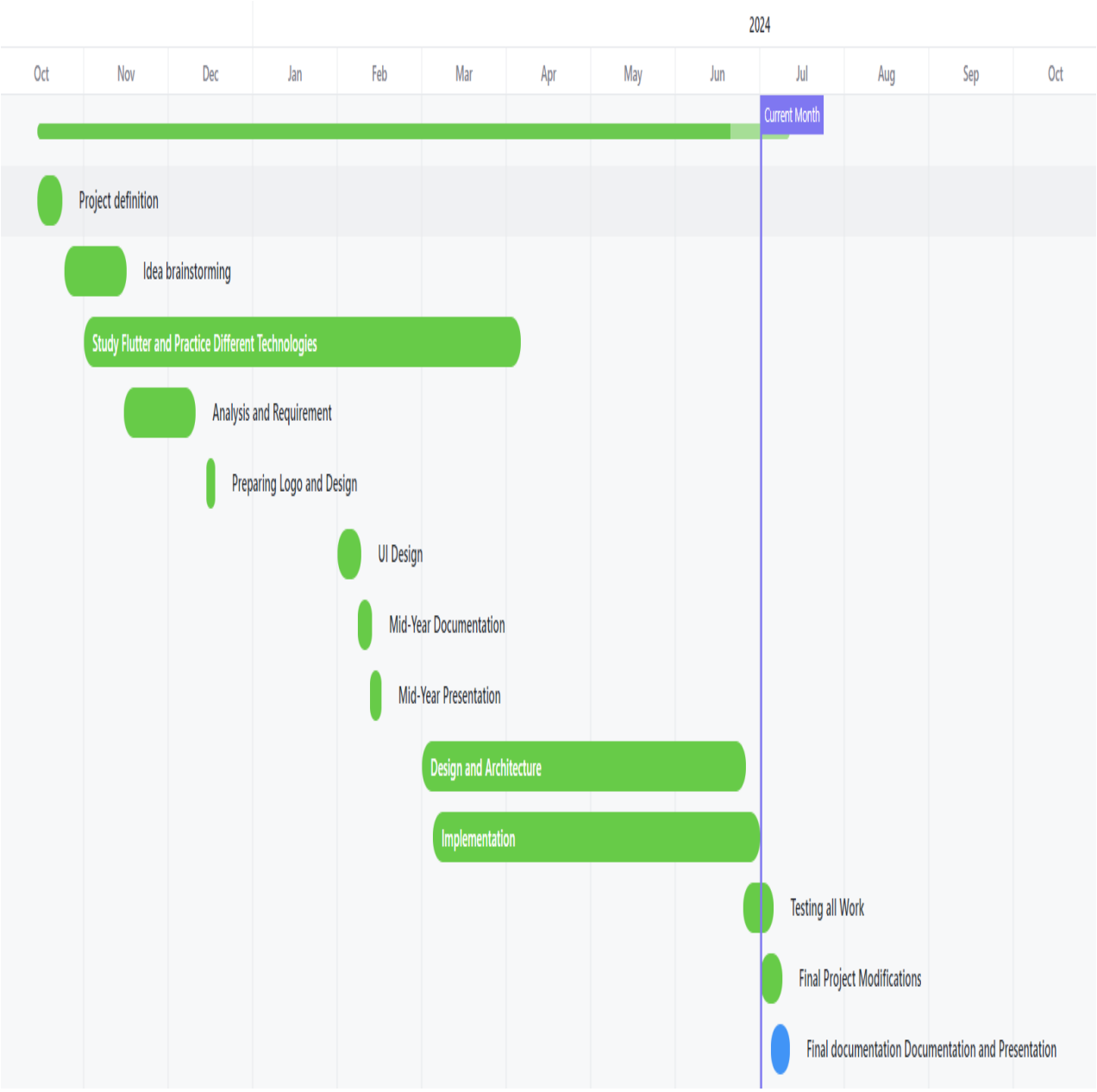
There is a lack of centralized, accessible, and personalized skincare guidance and product recommendations for individuals seeking to improve their skin wellness. Many people struggle to identify their specific skin type and the appropriate skincare products to address their concerns. Additionally, the skincare market is vast and often overwhelming, making it difficult for consumers to navigate and select products that align with their needs and budget. Without reliable information and guidance, consumers may waste time and money on ineffective products or potentially exacerbate their skin issues.

1.3 Project Objective (suggested solution)

The objective of our application is to address this and provide a user-friendly platform with valuable resources and guidance to provide the users with personalized skincare guidance, product recommendations, and educational resources. Also, the application aims to empower individuals to make informed skincare decisions and achieve their skin wellness goals efficiently and effectively, in addition to saving the users time and money.

1.4 Gantt chart of project time plan

Task Title	Start Date	End Date	Duration
Project Definition	15/10/2023	23/10/2023	8
Idea Brainstorming	25/10/2023	15/11/2023	20
Study Flutter and Practice Different Technologies	1/11/2023	5/4/2024	155
Analysis and Requirement	15/11/2023	10/12/2023	25
Preparing Logo and Design	15/12/2023	17/12/2023	2
UI Design	1/2/2024	8/2/2024	7
Mid-Year Documentation	8/2/2024	12/2/2024	4
Mid-Year Presentation	12/2/2024	15/2/2024	3
Design and Architecture	1/3/2024	25/6/2024	117
Implementation	5/3/2024	30/6/2024	117
Testing all Work	25/6/2024	5/7/2024	10
Final Project Modifications	1/7/2024	6/7/2024	5
Final-documentation	1/7/2024	6/7/2024	5
Final-Presentation Documentation	8/7/2024	13/7/2024	6


2024
Oct
Nov
Dec
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Current Month
Project definition
Idea brainstorming
Study Flutter and Practice Different Technologies
Analysis and Requirement
Preparing Logo and Design
UI Design
Mid-Year Documentation
Mid-Year Presentation
Design and Architecture
Implementation
Testing all Work
Final Project Modifications
Final documentation Documentation and Presentation

1.5 Project development methodology

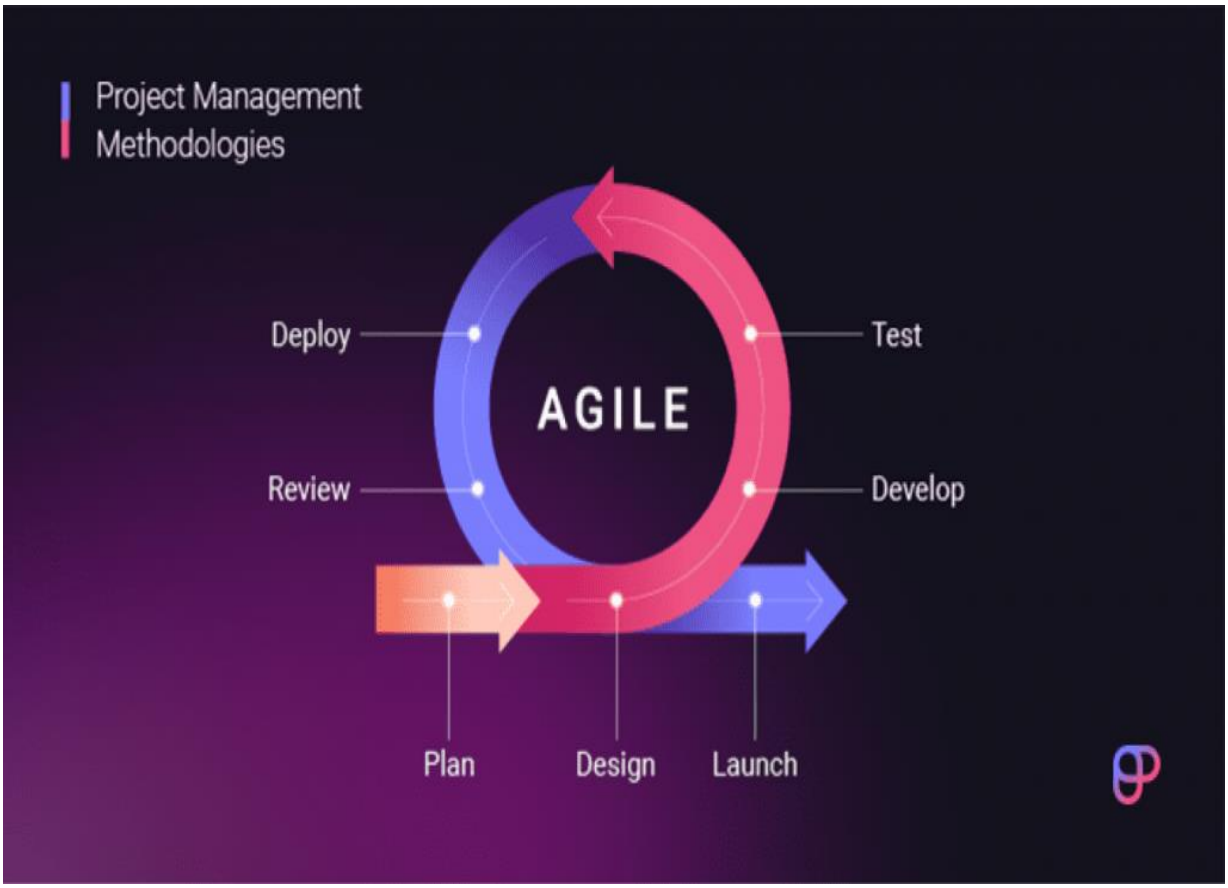
Our Project Development Methodology is Agile

Description: An iterative and incremental approach that focuses on flexibility, customer feedback, and continuous improvement.

Frameworks: Scrum, Kanban, Extreme Programming (XP).

Advantages: Highly adaptable, encourages stakeholder involvement, continuous delivery of valuable software.

Disadvantages: Can be challenging to predict timelines and costs, requires a high level of collaboration and communication.



1.6 The used tools in the project (SW and HW)

Software (SW) Tools:

1. Development:

- **IDE (Integrated Development Environment):** Android Studio (for Android), Visual Studio Code (for cross-platform development with Flutter)
- **Programming Languages:** Dart (for Flutter), Python (for machine model)

2. Design:

- **UI/UX Design:** Figma

3. Version Control:

- **Source Code Management:** GitHub

4. Project Management:

- **Task Management:** Trello
- **Collaboration:** Microsoft Teams, Discord

5. Continuous Integration/Continuous Deployment (CI/CD):

- **CI/CD Tools:** GitHub Actions

6. Analytics and Monitoring:

- **Analytics:** Firebase Analytics.
- **Crash Reporting:** Firebase Crashlytics.

Hardware (HW) Tools:

1. Development Devices:

- **Computers:** Laptops with sufficient processing power, RAM, and storage.
- **Mobile Devices:** Smartphones and tablets (IOS and Android devices for testing purposes).

2. Network:

- **Wi-Fi Routers:** For testing app performance on different network conditions.

3. Peripheral Devices:

- **USB Cables:** For connecting mobile devices to computers for development and debugging.
- **Adapters:** For connecting different types of devices and peripherals.

4. Testing Devices:

- **Multiple Device Models:** To ensure app compatibility across different screen sizes, resolutions, and operating system versions.
- **Emulators:** Built into Android Studio for virtual testing of applications.

1.7 Report Organization (summary of the rest of the report)

Cosmo Care application is a mobile app designed to assist users in understanding their skincare needs, identifying their skin type, and addressing specific skin problems. Key features include:

Personalization: Uses AR and AI to provide tailored skincare recommendations.

User Experience: Offers expert advice, virtual try-ons, and a seamless shopping journey.

Functionality: Users can sign up, search and filter products, add to cart, leave reviews, and get customer support.

Differentiators: Helps users identify skin type, offers AI-driven product suggestions, and includes a chat bot for advice.

System Architecture:

Front-end: Flutter.

Back-end: Dart.

Database: Firebase.

Stakeholders: Pharmacists, dermatologists, project managers, manufacturers, users, developers, marketing team, designers, QA team, supply chain/logistics.

Non-functional Requirements: 24/7 availability, compatibility with various platforms, strong security, quick response times, scalability, and good performance.

SWOT Analysis:

Strengths: Personalization, user engagement, e-commerce integration, innovative technology.

Weaknesses: Privacy concerns with face detection technology.

The app aims to make skincare routines more informed and efficient, improving user satisfaction and confidence.

Chapter 2: Related work

Charm and **trove Skin** are the most similar applications to ours

Similarities:

- How to use the products.
- Product ingredients.
- Products concerns.
- Overview about the product.
- Products Ratings/Reviews.
- Concerns Description.

Differences:

- If the user does not know his skin type our app will help him in knowing it.
- Prices of every product is available.
- Products suggestion based on user's budget, concerns and skin type (But in the above apps suggestion based on user's routine and the products he uses).
- We do not have a selfie log to compare the skin progress.
- We have chat Bot (For expert help and more knowledge) helping the user in getting more information like why he is facing some concerns.
- We do not have morning and night routine reminders.
- Users can order the suggested products from our application but in the above applications users cannot order and there is no cart.
- We have barcode scanner that help the user to get the products info faster and easier

	charm	***Trove Skin***	***Cosmo Care***
How to use the Products	✓		✓
Products ingredients	✓	✓	✓
Skin Concerns	✓	✓	✓
Overview about the product	✓		✓
AI skin problems detection		✓	✓
Prices of every product			✓
Selfie log to compare the skin progress	✓	✓	
Products recommendations	✓		✓
Chat Bot		✓	✓
Shopping cart			✓
Free trial	7 days	7 days	1 month
Skin care tips			✓
Bar code scanning	✓	✓	
Morning and night reminders	✓	✓	

Chapter 3: System Analysis

3.1 Project specification

3.1.1 Functional requirement

The user should be able to:

1) Sign up.
2) Log in.
3) Log out.
4) Restore account (if he forget password)
5) View profile.
6) Edit profile.
7) Scan product Bar Code.
8) Get Product Info.
9) View product ratings.
10) Add review.
11) View product reviews.
12) Add review
13) Upload face image
14) Get his skin type
15) Choose his skin type if he already knows it.
16) Update his skin type (Edit/Change it).
17) Choose skin concerns.
18) Show concerns info.
19) Specify his budget.
20) Get personalized products suggestion.

21) Add to cart.

22) View cart.

23) Remove item from cart.

24) Make order.

25) Get total amount before proceed.

26) Show order details and its confirmation.

27) Choose payment method.

28) Put order info.

29) Search for product

30) Get more help and information using Chat Bot.

3.1.2 Non-functional requirement

1) Availability:

- The system shall be available 24/7.
- The system should be available in any platform and operating systems (flutter).

2) Compatibility:

- The system should be compatible with the latest versions of platforms as well as the old ones.

3) Security:

- The database must not store the user scanned picture to protect his privacy, it will just store final analysis of the picture.

4) Response time:

- The system should respond to user inputs within 2 seconds maximum.

5) Scalability:

- System can effectively handle increasing loads and increasing number of users.

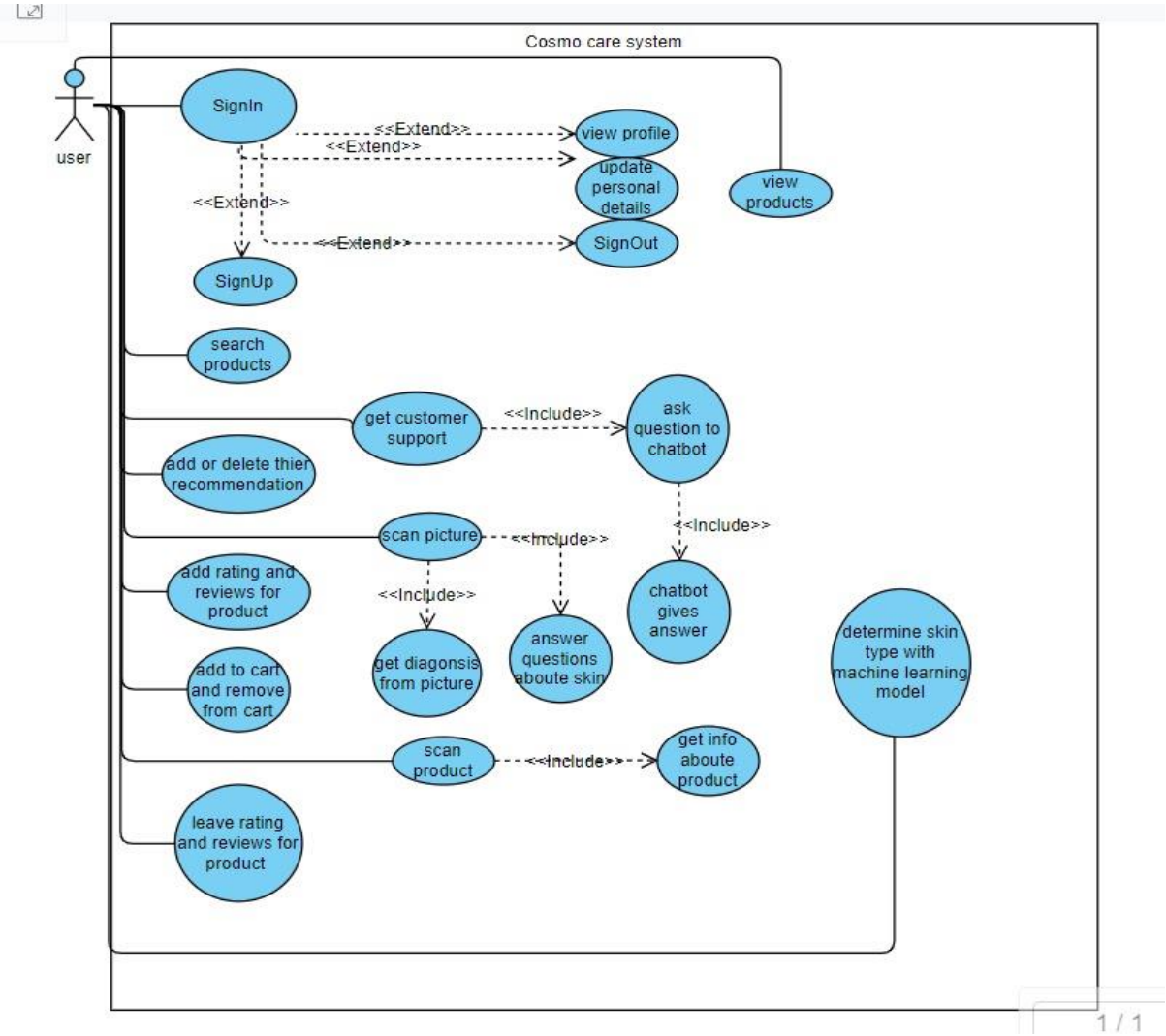
6) Performance:

- The system shall support a database of at least 50 cosmetic products without performance degradation.

7) Usability:

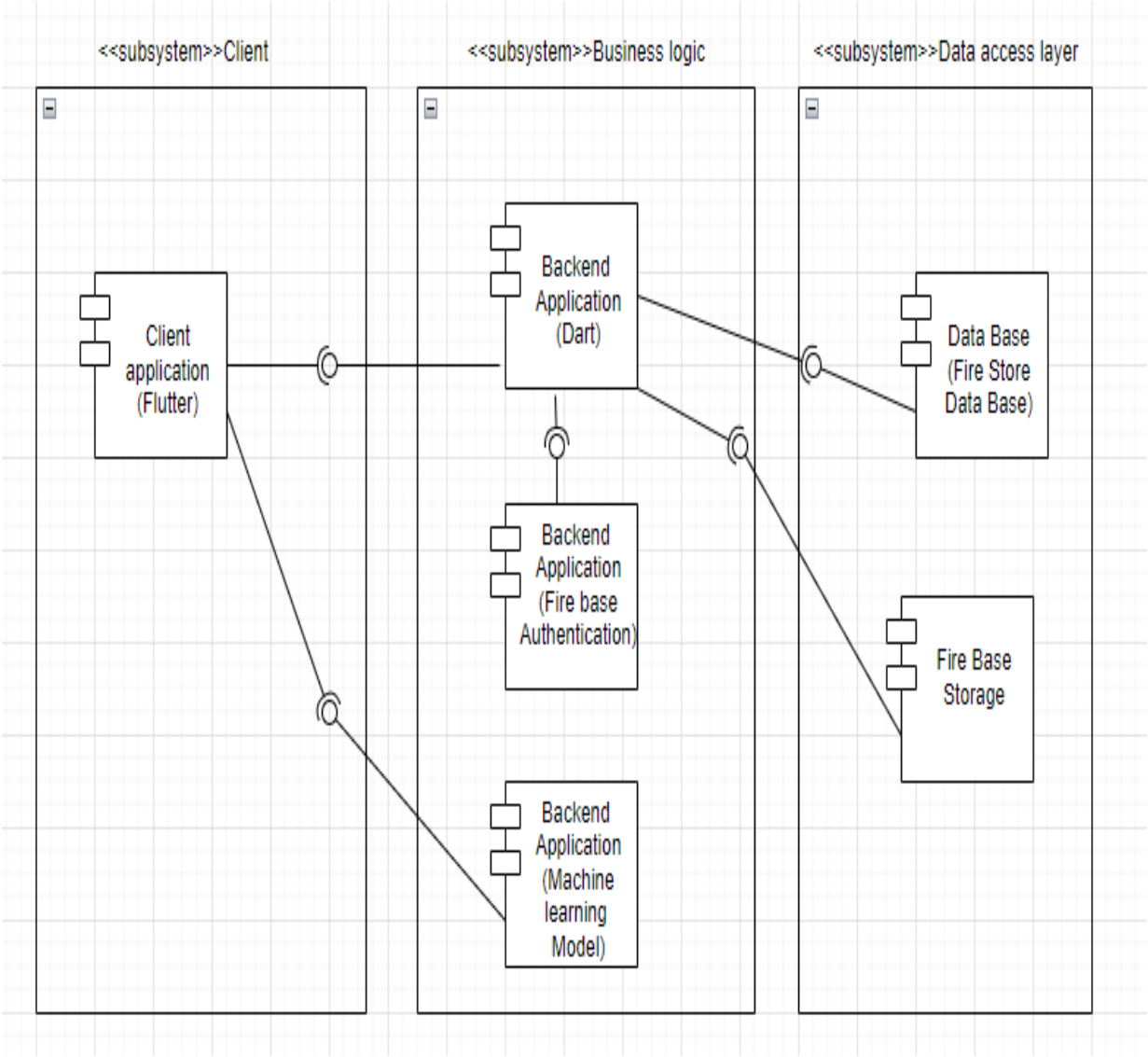
- The user interface shall be easy to navigate and understand, providing clear instructions for doing any task and accessing information.
- User experience should be accepted by most of users.

3.2 Use case Diagrams

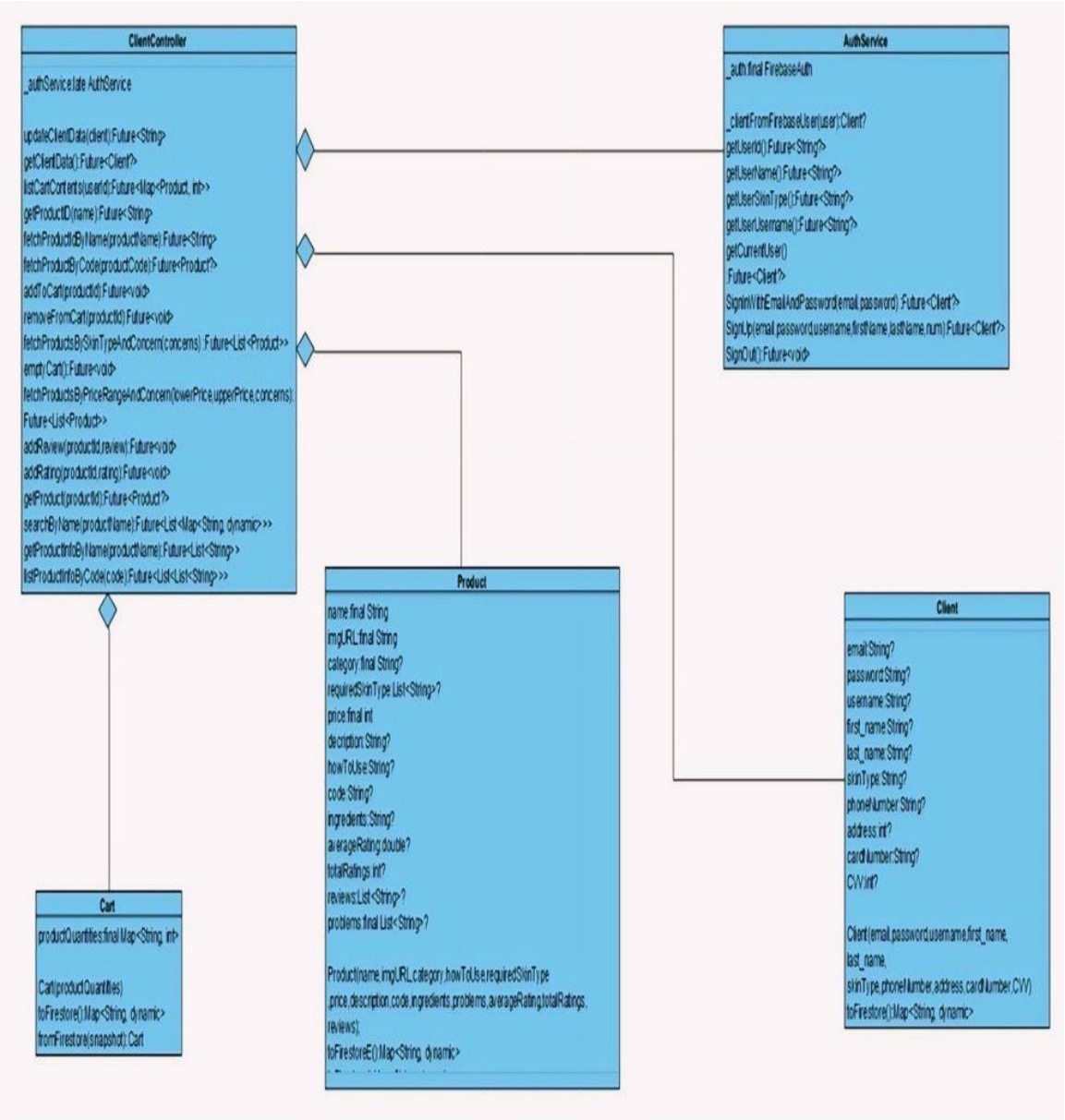


Chapter 4: System Design

1. System Component Diagram



2. System Class Diagrams



3. Sequence Diagrams

3.1 Search for a product

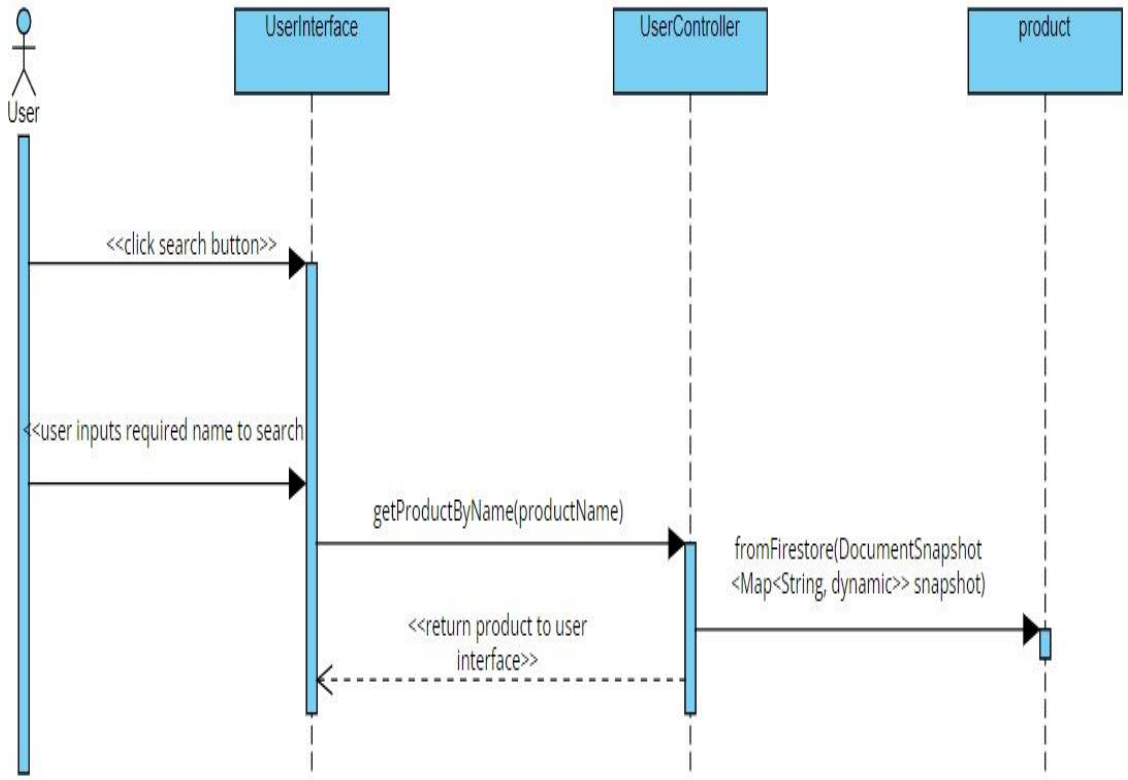


figure 3.1: Search of product

When a user desires to search for a product, they input its name and initiate the search process by clicking the designated search button. Subsequently, the user controller class undertakes the task of matching the provided product name with the entries stored within the database. Following this matching process, if the product is found, the system proceeds to display the corresponding results.

3.2 Scan product bar code

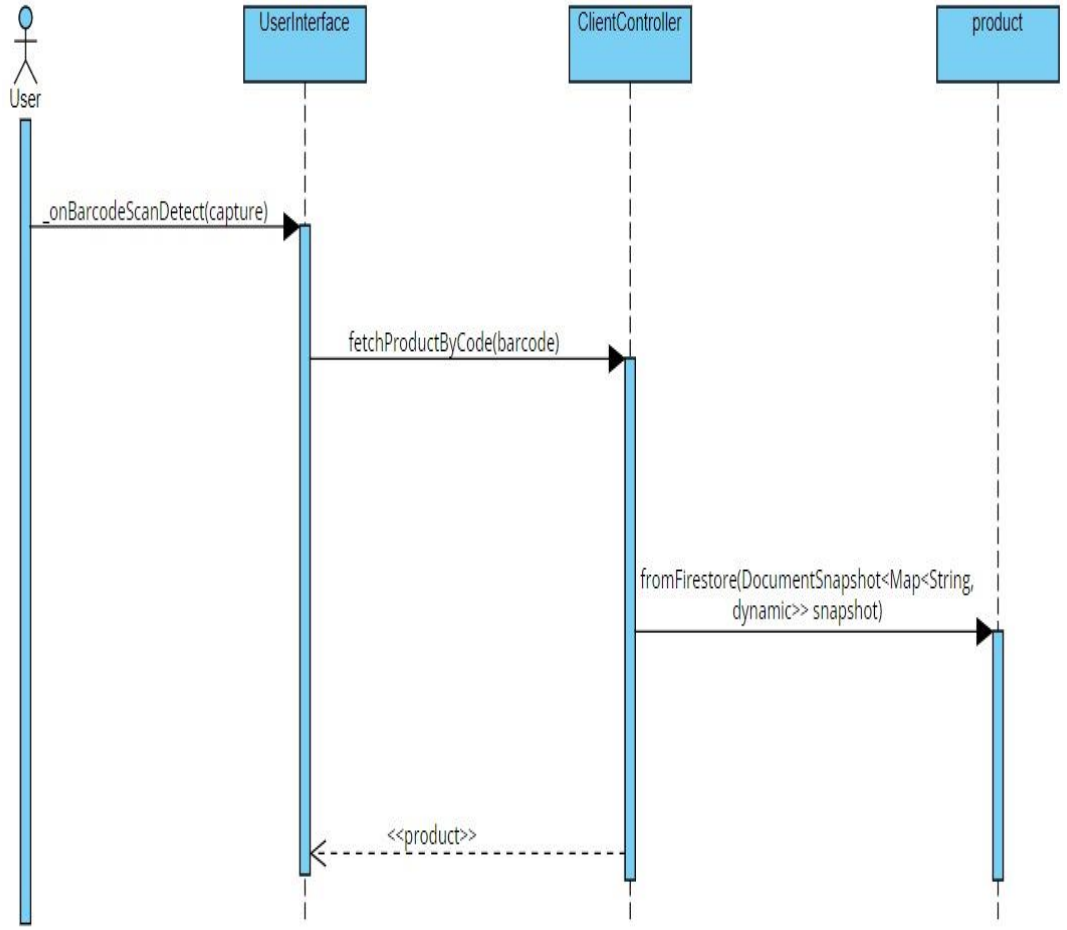


figure 3.2: Scan product bar code

When the user utilizes the camera to scan the barcode of a product, the user controller initiates a process to identify the corresponding product. This involves querying the database to locate the entry with a matching barcode. Upon successful identification, the user controller proceeds to extract and display the relevant information pertaining to the scanned product.

3.3 Scan user picture to predict skin type

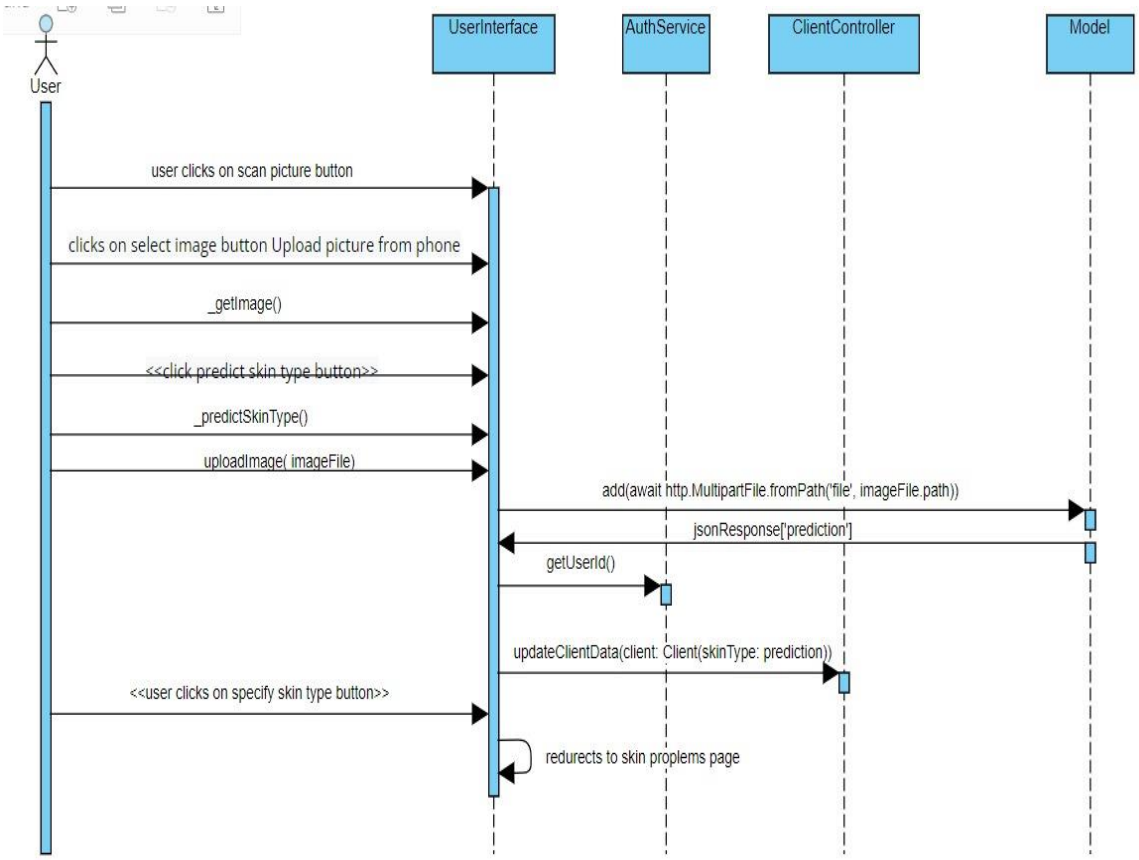


figure 3.3: Scan user picture to predict skin type

The user clicks the "scan picture" button and selects an image from their phone. The user interface retrieves and uploads the image to the server. The authentication service processes the image and returns the skin type prediction. The client controller retrieves the user's ID and updates their profile with the new skin type. The user interface redirects the user to a page to view or address their skin problems.

3.4 Chat bot

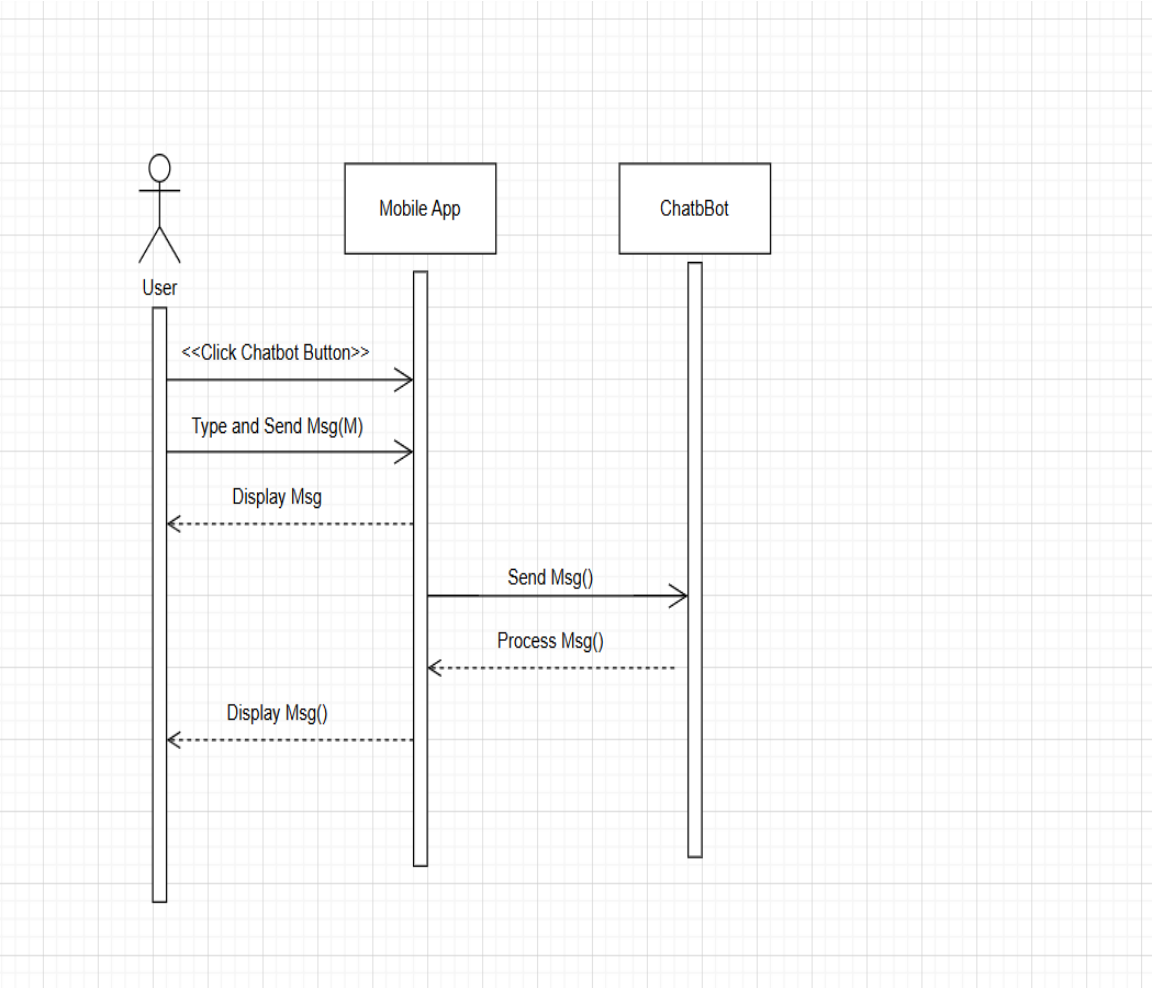


figure 3.4: Chat bot

The user clicks the chat bot button in the mobile application. The user then types and sends a message (Msg) through the mobile app. The mobile app displays the user's message and sends it to the chat bot for processing. The chat bot processes the received message and sends a response back to the mobile app. Finally, the mobile app displays the chat bot's response to the user.

3.5 Add Review

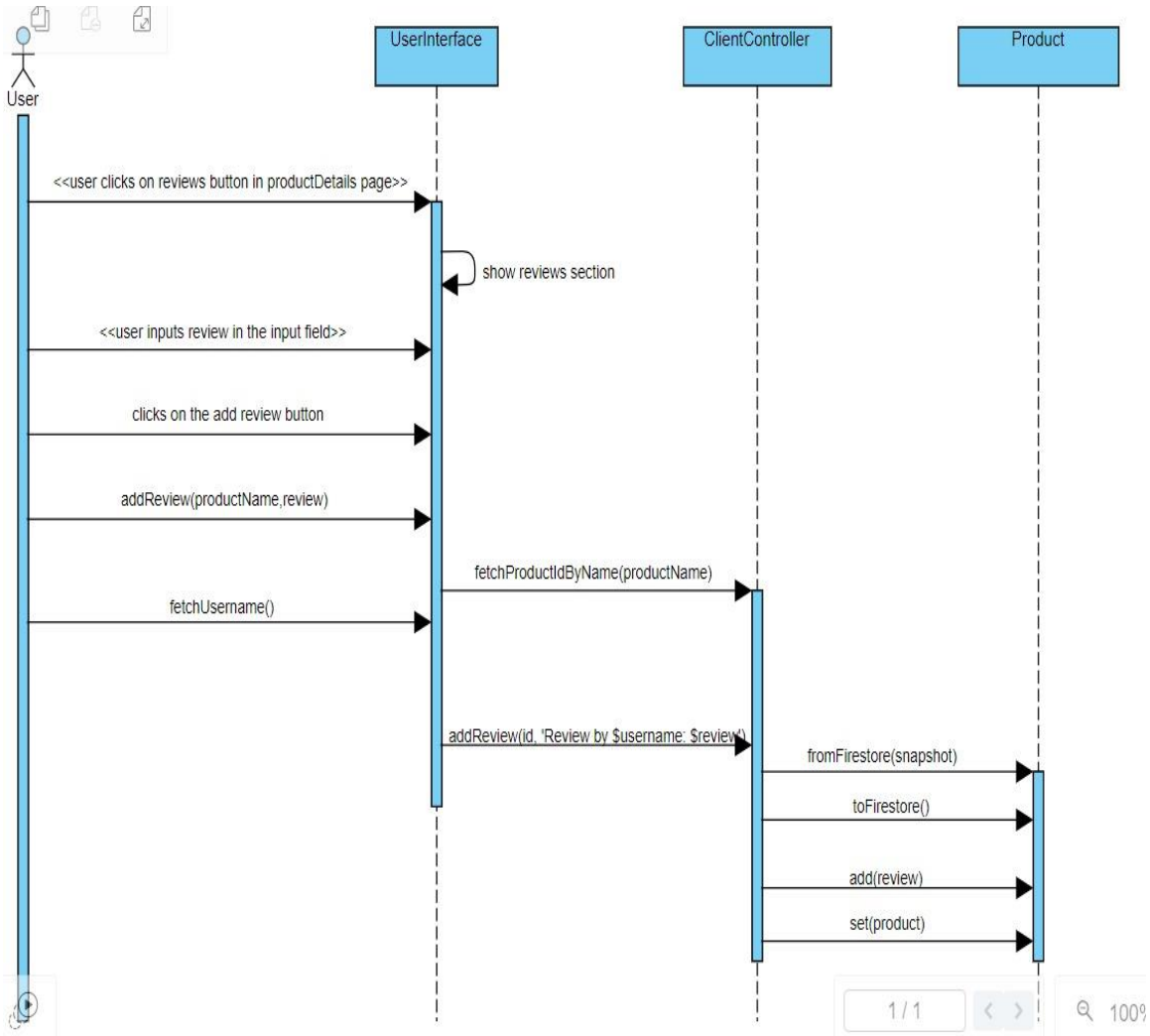


figure 3.5: Add Review

- The user navigates to the specific product details page.
- The user clicks on the "Reviews" section.
- Adding a Review: The user enters their review in the "Add Review" field.
- The user clicks the "Add Review" button.

Database Update:

The review, along with the user's username, is added to the Fire store database.

3.6 Add Rating

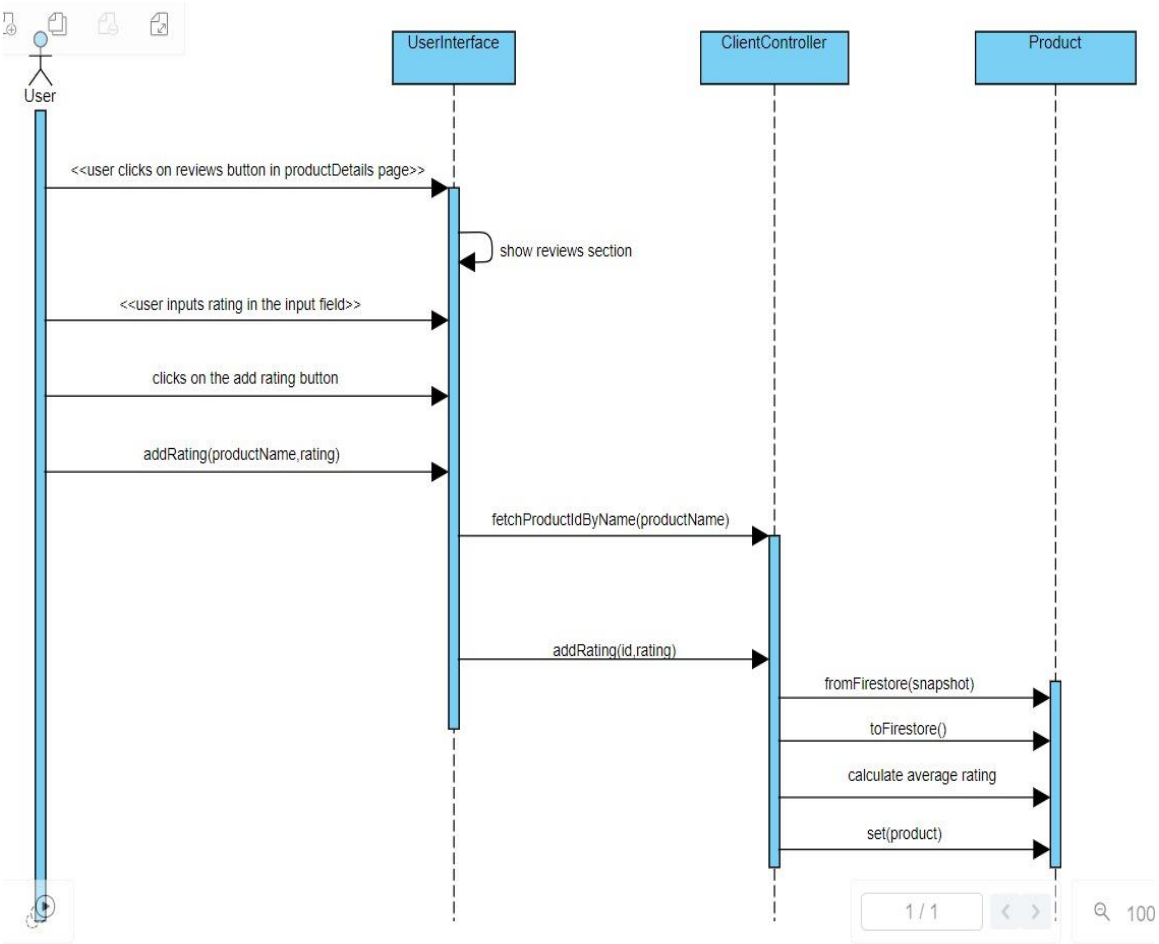


figure 3.6: Add Rating

- The user navigates to the specific product details page.
- The user clicks on the "Reviews" section.
- Adding a Rating: The user enters their rating in the "Add Rating" field.
- The user clicks the "Add Rating" button.
- Database Update: The rating is added to the Fire store database.
- The average rating for the product is recalculated and updated in the database.

3.7 Add Product to Cart

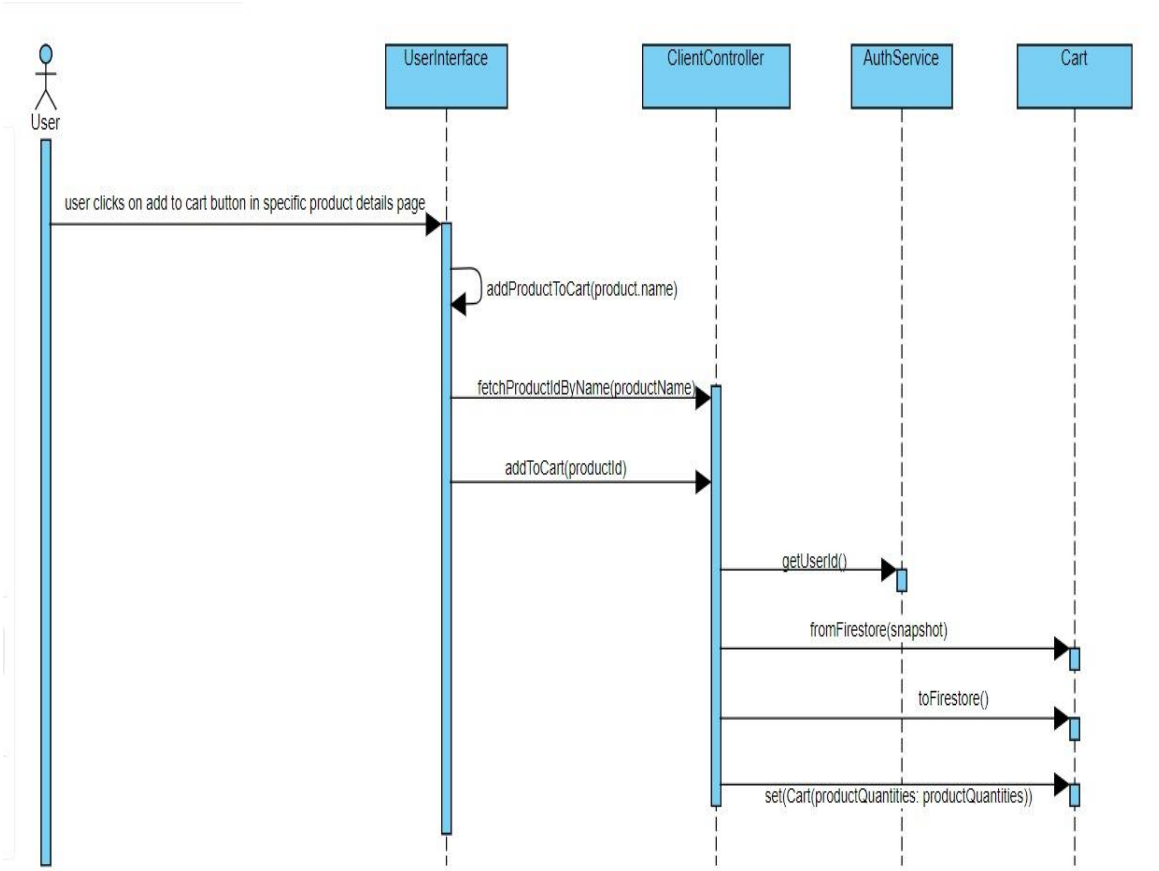


figure 3.7: Add Product to Cart

When the user clicks the "add to cart" button, the user interface communicates with the client controller to handle the action. The client controller retrieves the product ID based on the product name and then proceeds to update the cart. It interacts with the authentication service to obtain the current user's ID and fetches the existing cart data from Fire store. After adding the new product to the cart, the updated cart information is saved back to Fire store, ensuring the cart state reflects the changes. This sequence demonstrates the coordination between the user interface, client controller, authentication service, and cart components to manage the add-to-cart functionality.

3.8 Signup

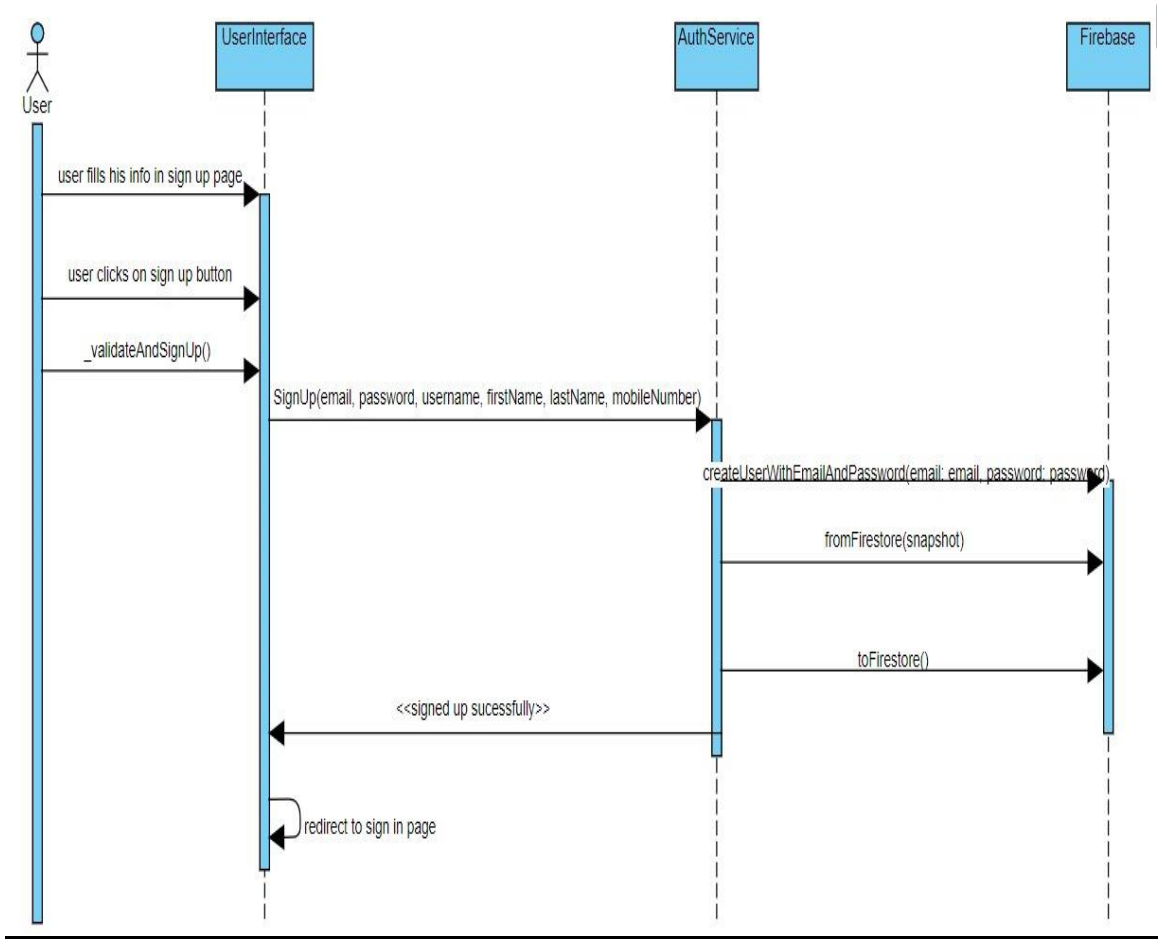


figure 3.8: Signup

1-user clicks on create a new account button if their does not have account.
2-system redirects user to sign up page
3-user fills his info.
4-user clicks on sign up button
5-user account saved on fire store database
6-system alerts user that they are signed up successfully and redirects user to sign in page.

3.9 sign-In

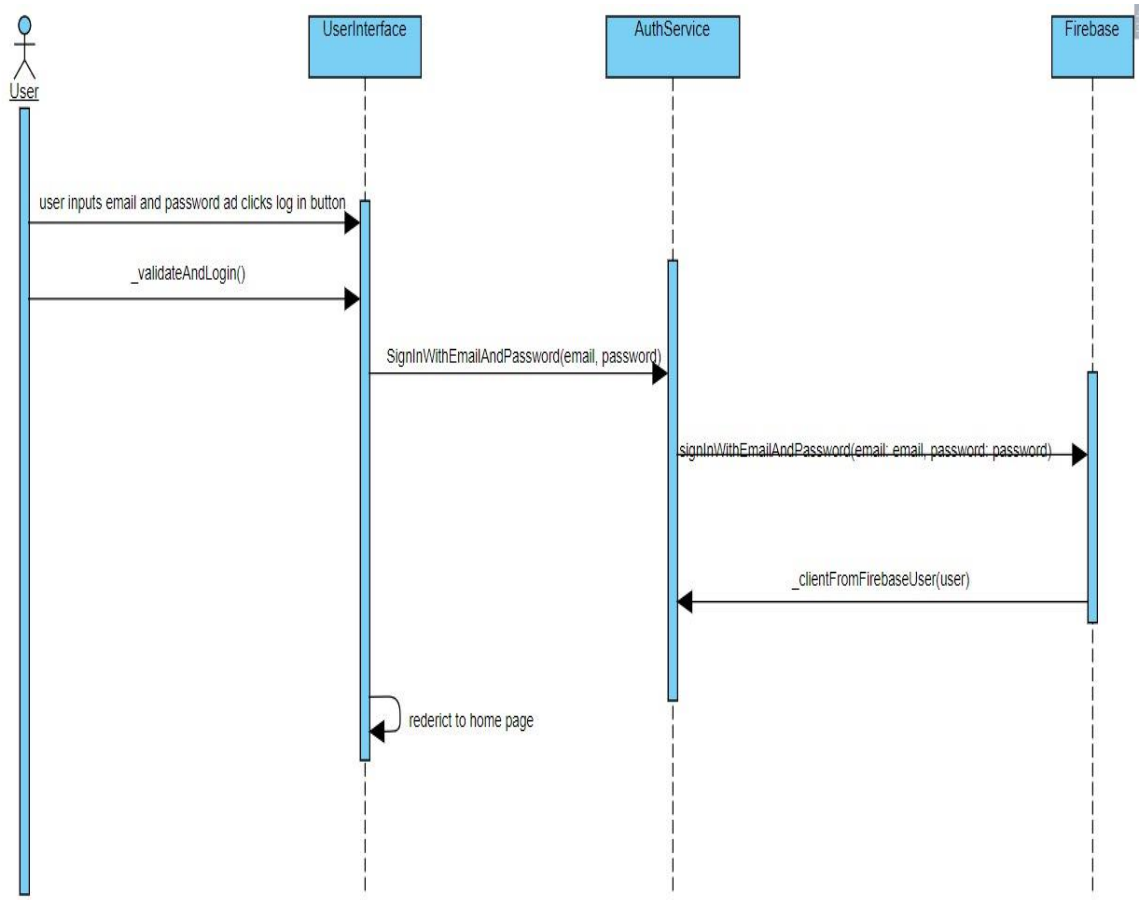


figure 3.9: Sign-In

1-user enters his correct email and password.
2-user clicks on sign in Log in button if them did not forget the password.
3-system redirects user to the home page if email and password is correct.

3.10recommend product

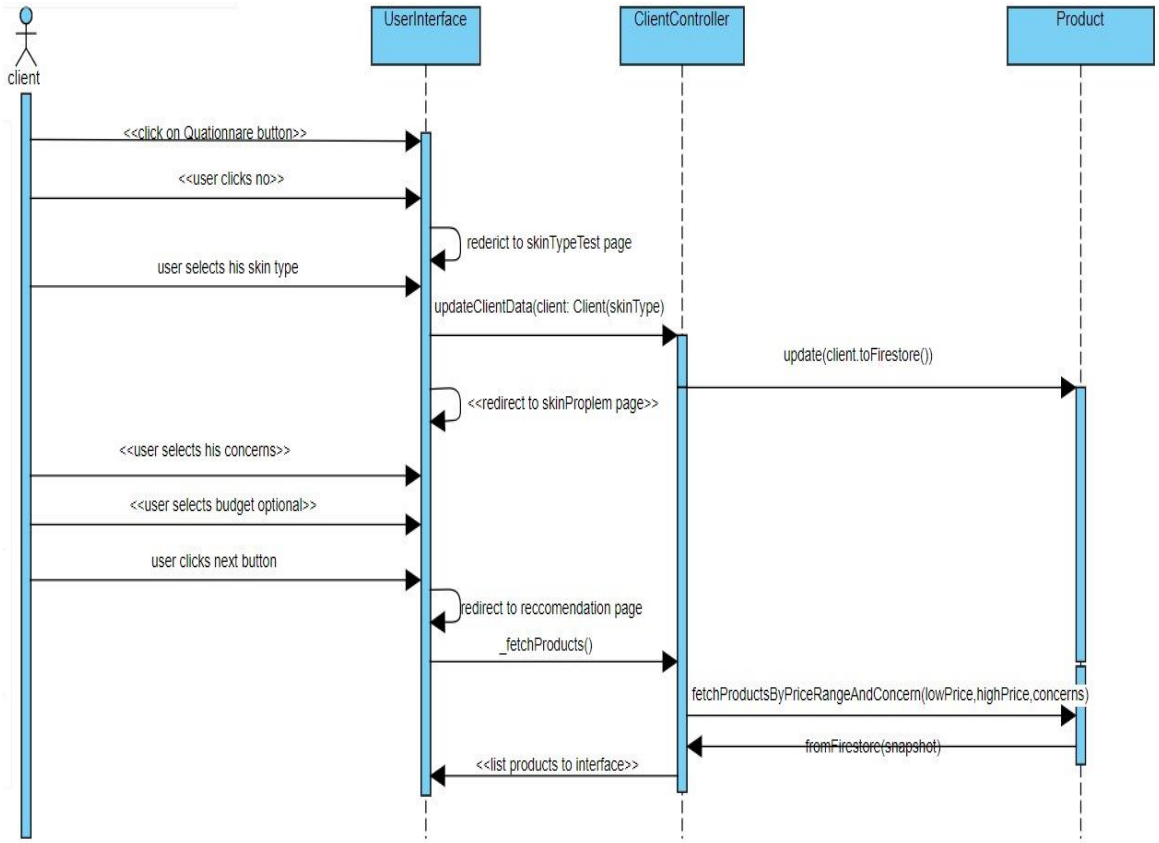
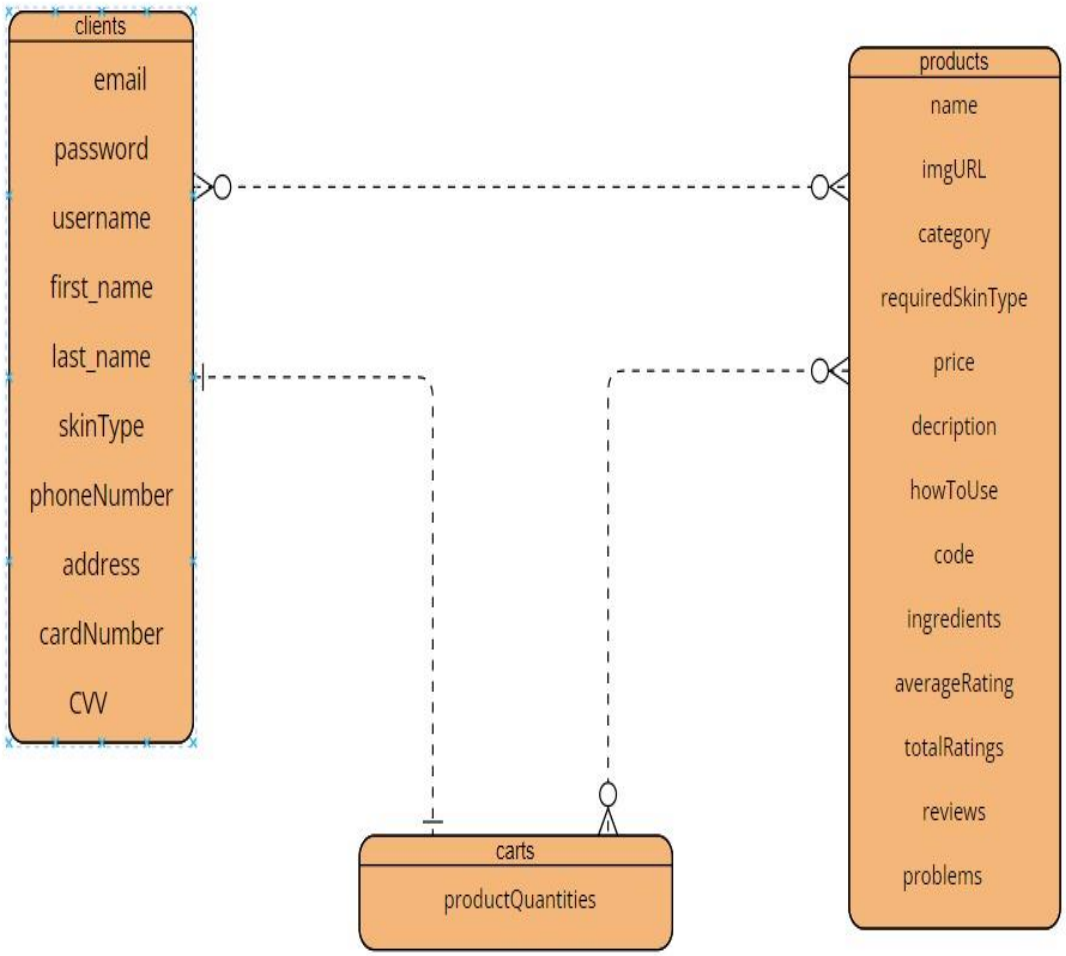


figure 3.10: recommend product

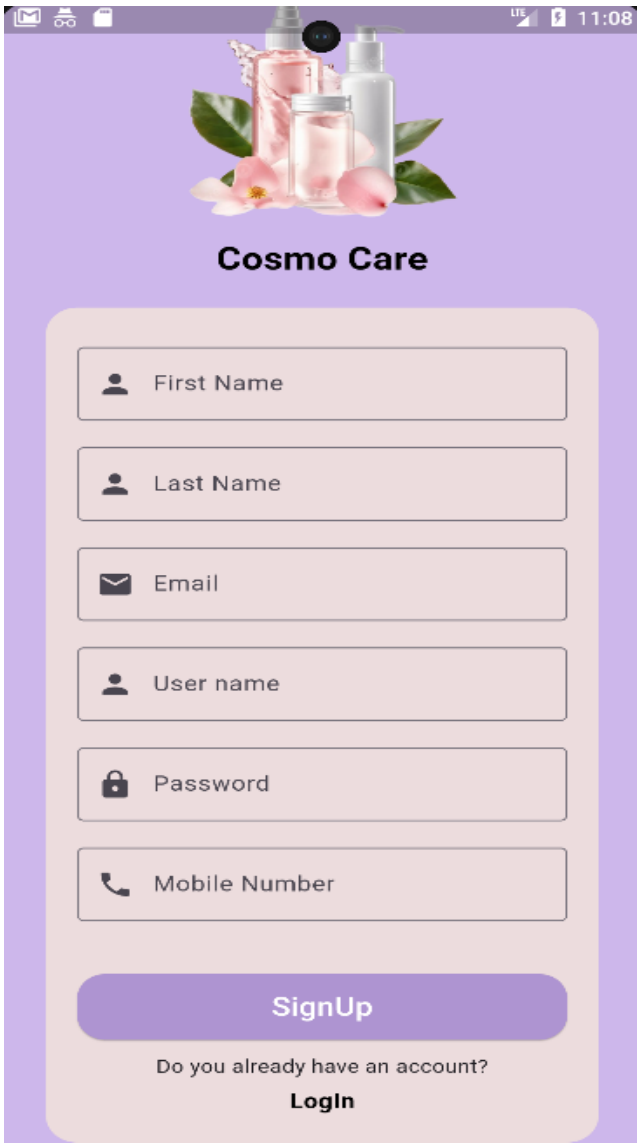
- The client interacts with the user interface by clicking a questionnaire button.
- The user selects "no" for a particular option.
- The user selects their skin type.
- The system updates the client's data with the selected skin type and redirects to the skin problem page.
- The user selects their skin concerns.
- The user optionally selects a budget.
- The user clicks the next button.
- The system redirects to the recommendation page and fetches relevant products from Fire store based on the client’s skin type, concerns, and budget.

• Project ERD

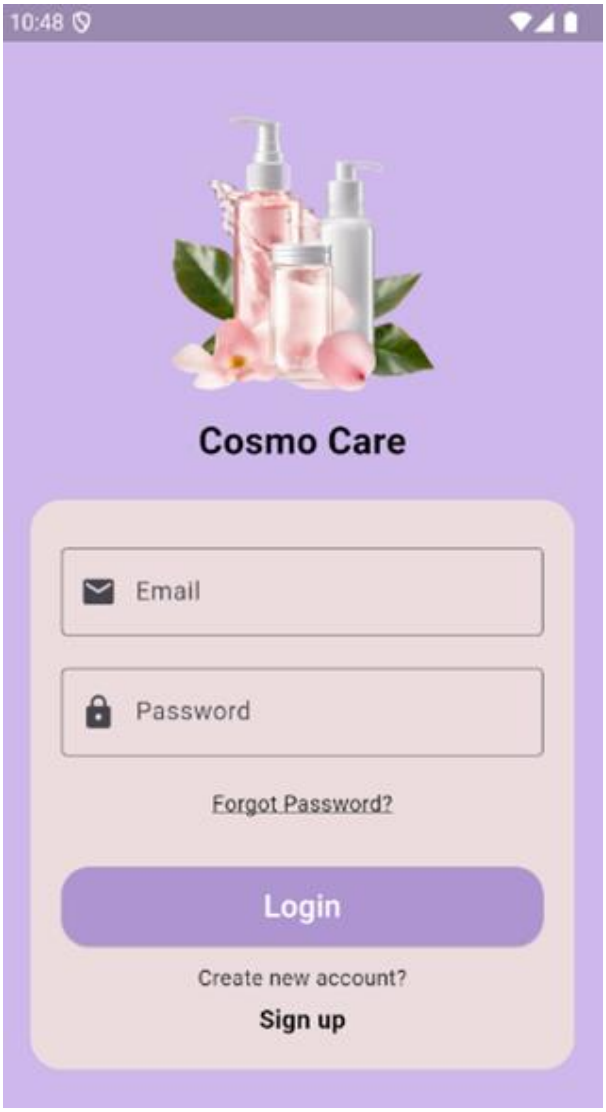

clients
email
password
username
first_name
last_name
skinType
phoneNumber
address
cardNumber
CVV
products
name
imgURL
category
requiredSkinType
price
decription
howToUse
code
ingredients
averageRating
totalRatings
reviews
problems
carts
productQuantities

• System GUI Design

Sign Up Page

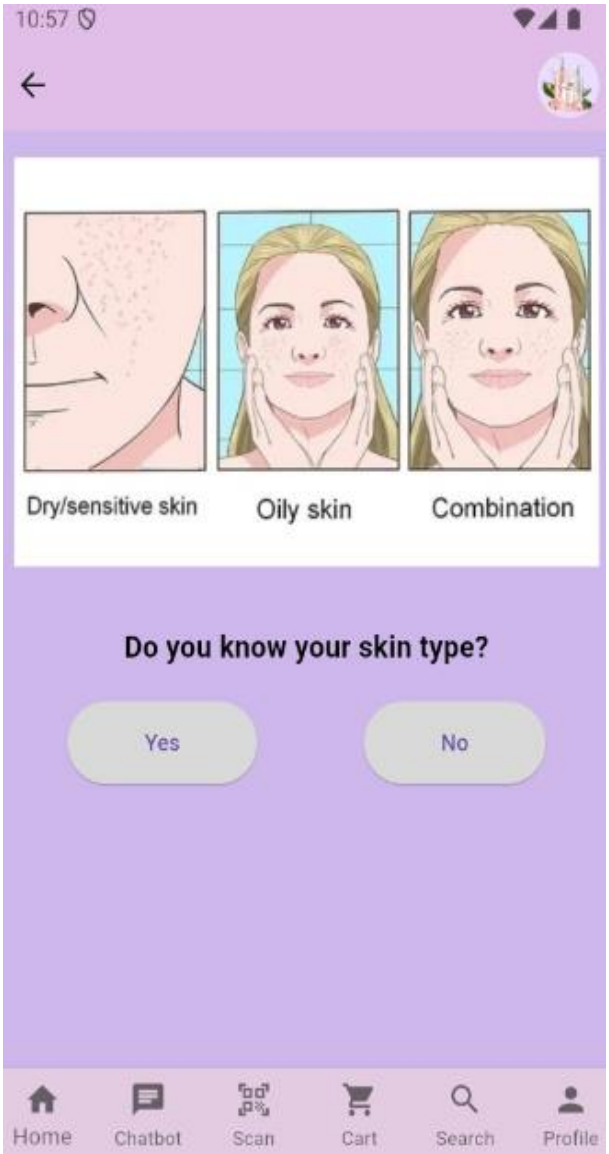


Login page

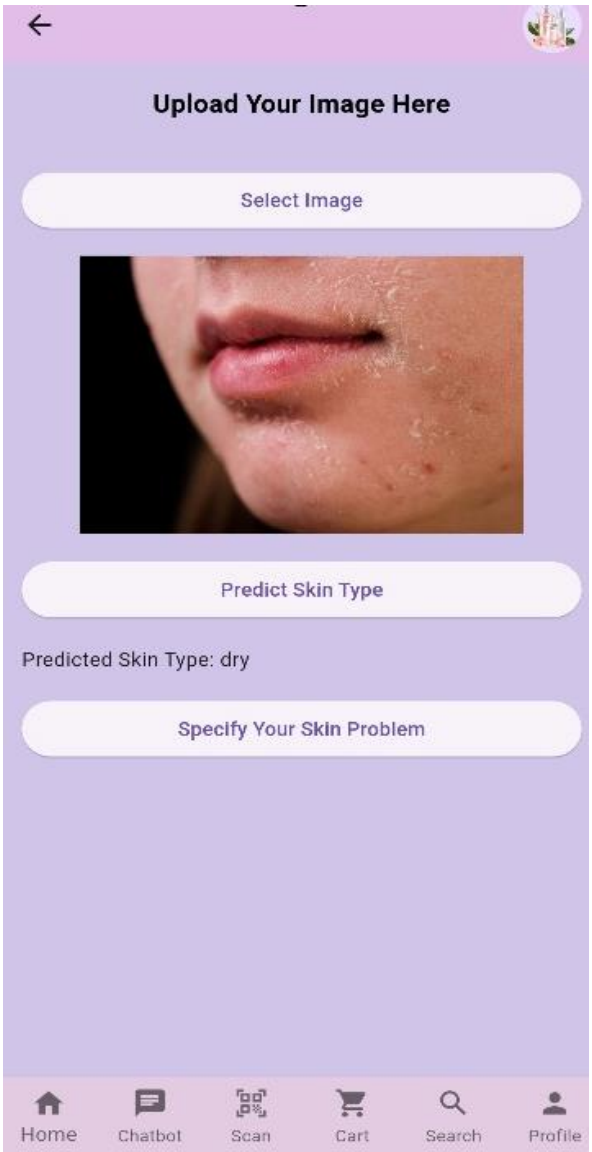


Home Page

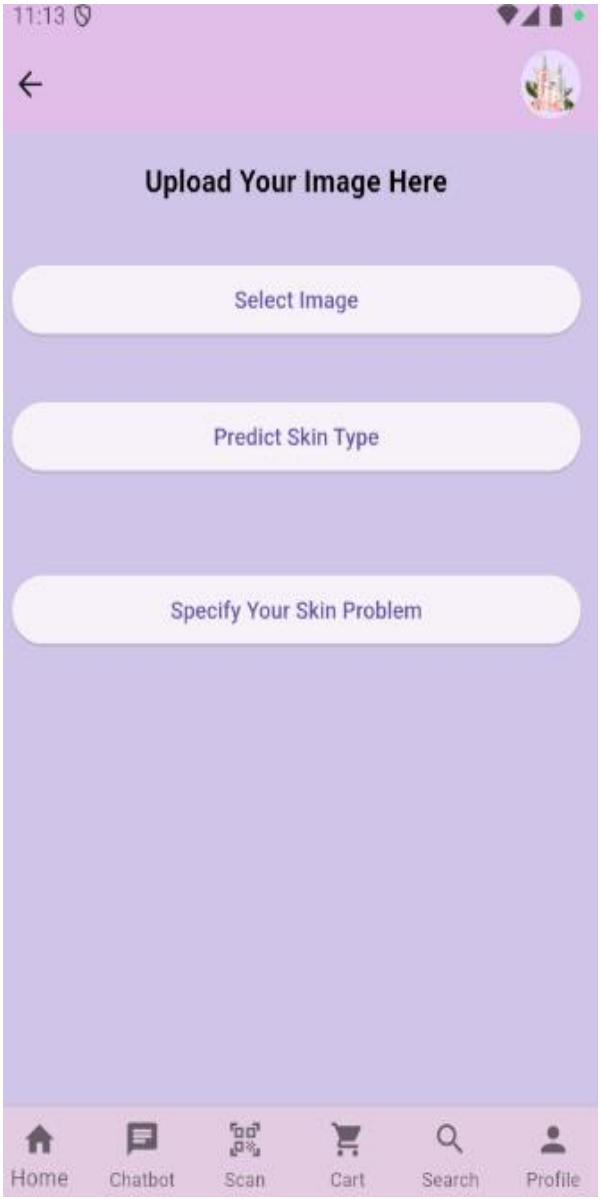
Questionnaire Page



Skin Type Test Page



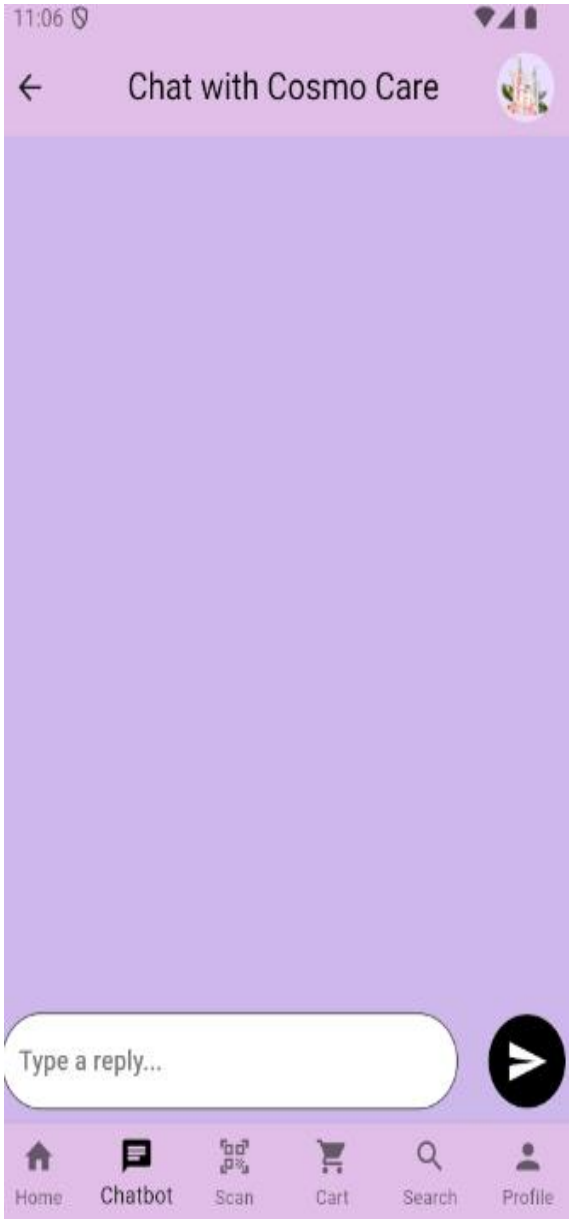
Scan Model Page



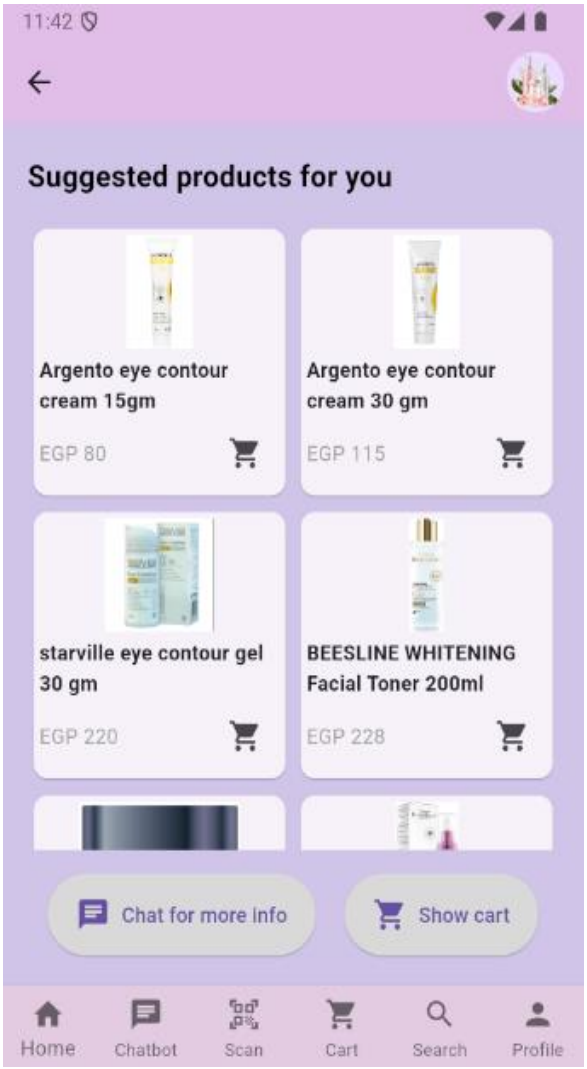
Search Page

Cart Page

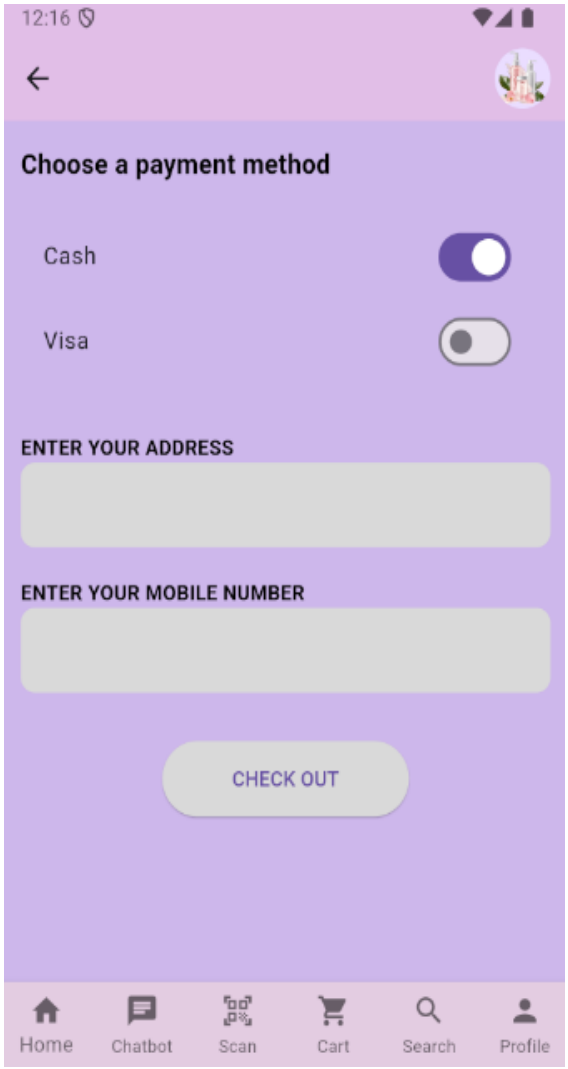
Chat Bot Page



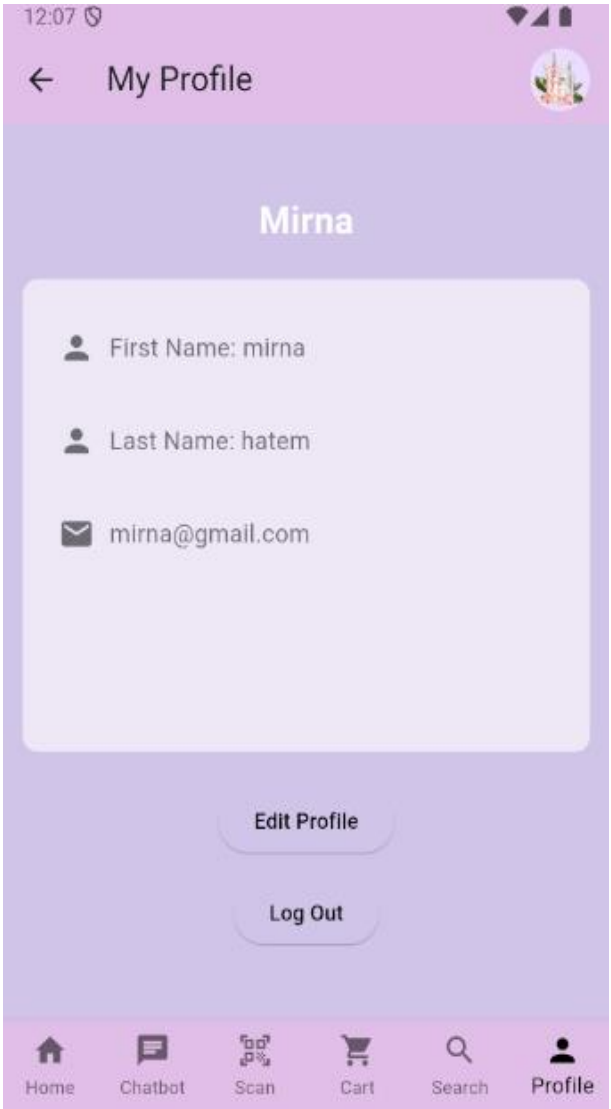
Recommendation Page



Payment Method Page

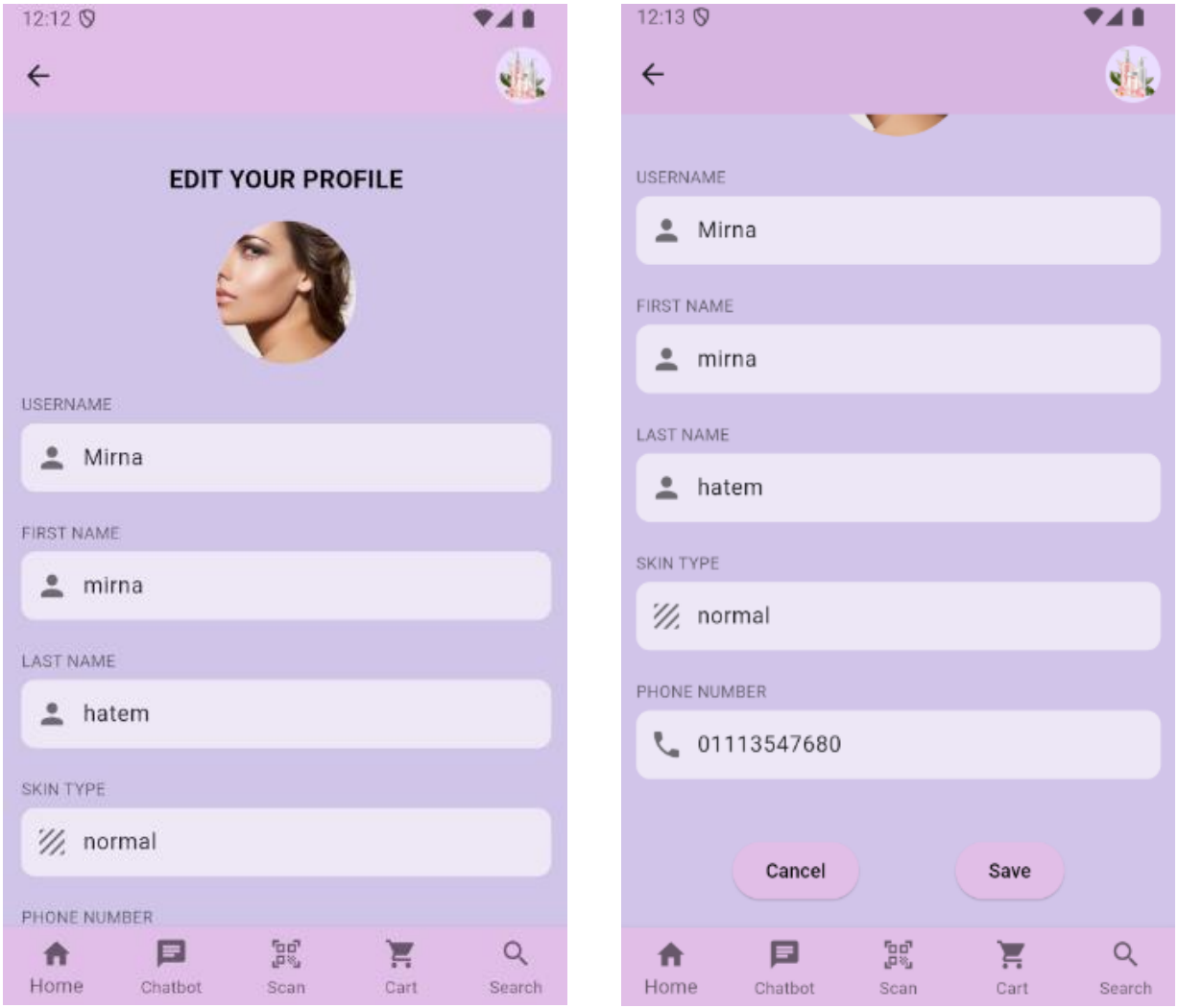


My Profile Page



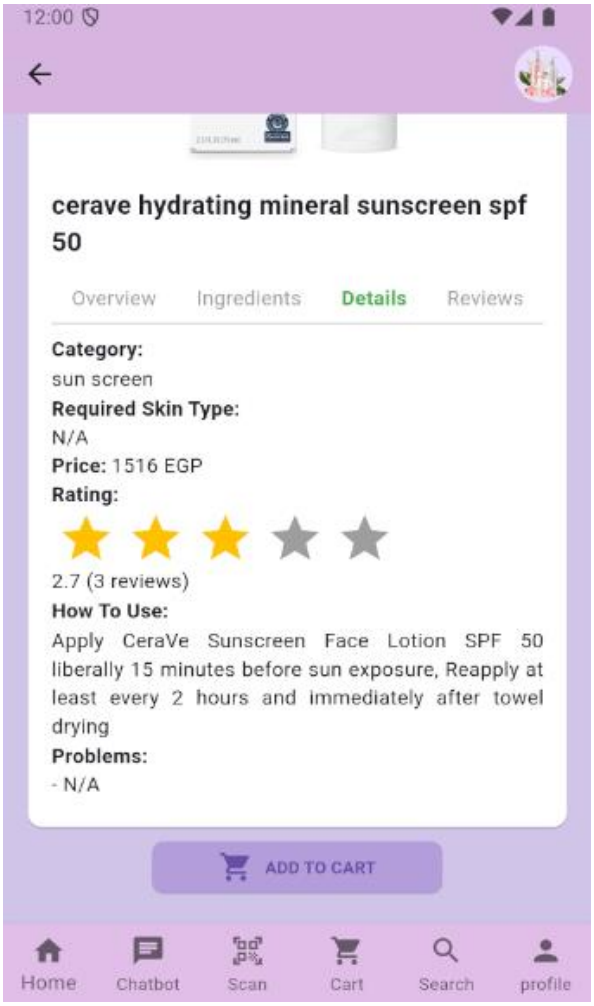
Bar Code Scanning Page

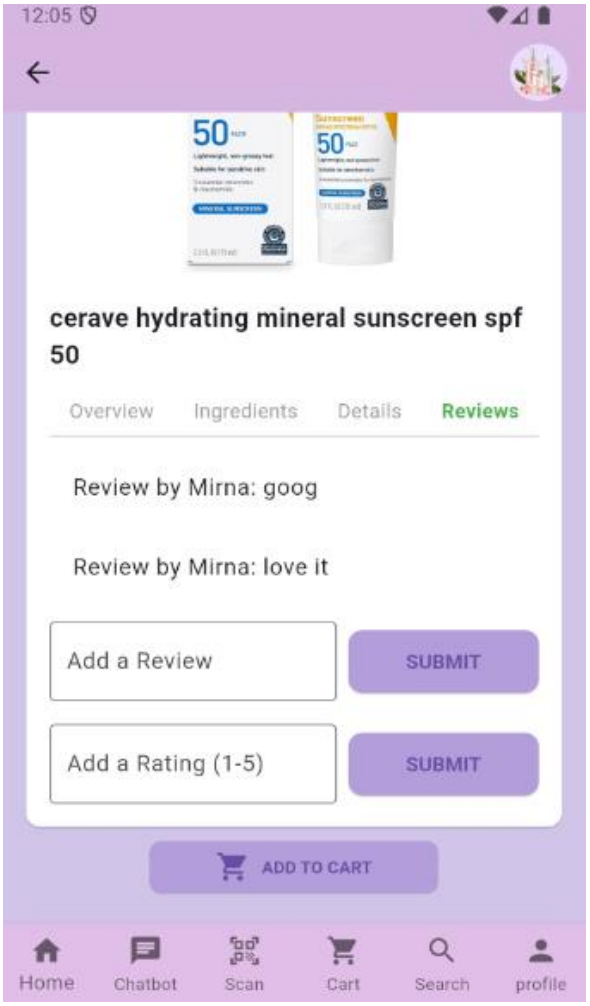
Edit Profile Page



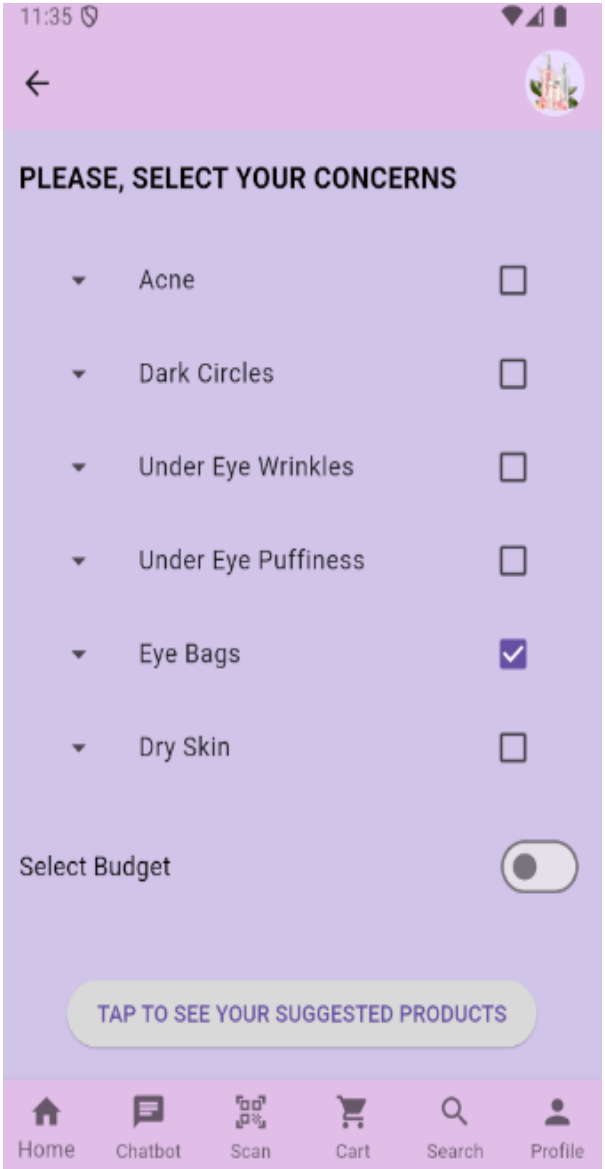
Product Details Page

Products Review and Rate Page





Skin problem Page



Final Order Verification Page

Chapter 5: Implementation and Testing

System running and samples of the applied test cases (System test cases)

5.1 EfficientNet V2 Model

5.1.1 Machine Learning Model Overview:

- Objective: Classify skin types into three categories: Dry, Normal and Oily.
- Approach: Utilize EfficientNet V2 S for image classification.
- Tools and Libraries: PyTorch, Open CV, Numpy
- Techniques : Data Augmentation
- **Dataset Resource** : [Oily, Dry and Normal Skin Types Dataset | Kaggle]
- **Classes in Dataset:**

	Dry	Normal	Oily
Number of Records	888	1162	962

5.1.2 Data Processing:

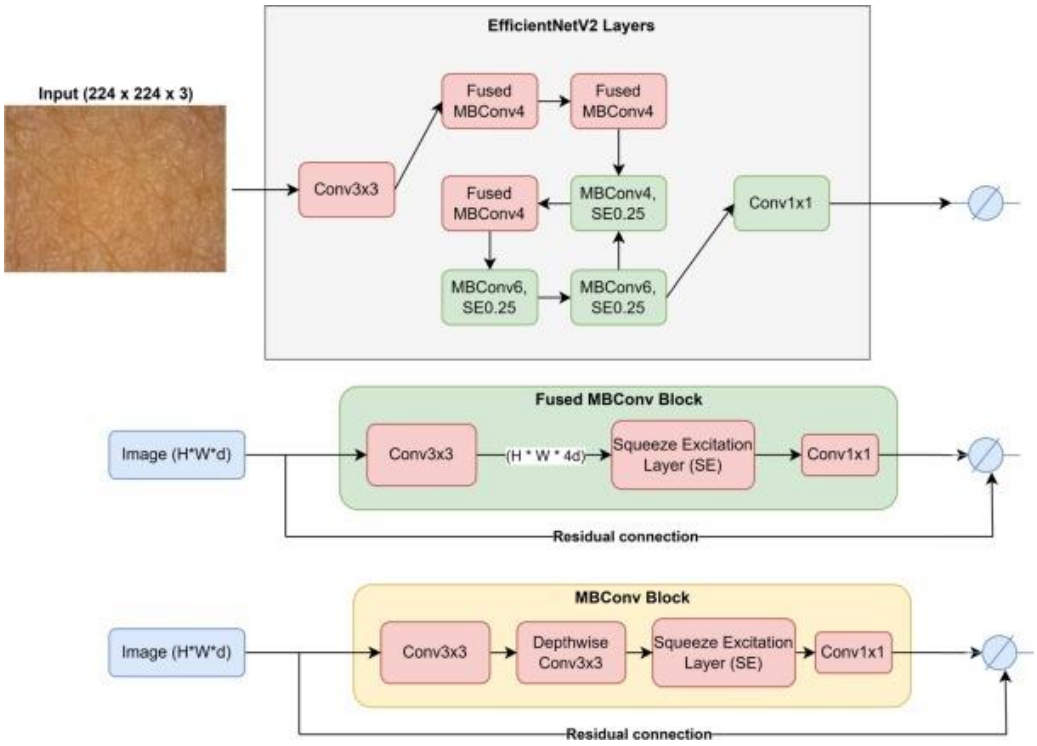
Steps:

- **Resizing**: Images resized to 224x224 pixels.
- **Normalization**: Using mean and standard deviation.
- **Data Augmentation**: Random vertical flips

```
Welcome        app.py ×
app.py > ...
train_transform = transforms.Compose([
    transforms.ToPILImage(),
    transforms.Resize((224, 224)),
    transforms.RandomVerticalFlip(0.6),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

5.1.3 Model Architecture:

- **Model**: EfficientNet V2 S
- **Pretrained**: On ImageNet
- **Fine-tuning**: Modified for skin type classification
- **Diagram**:

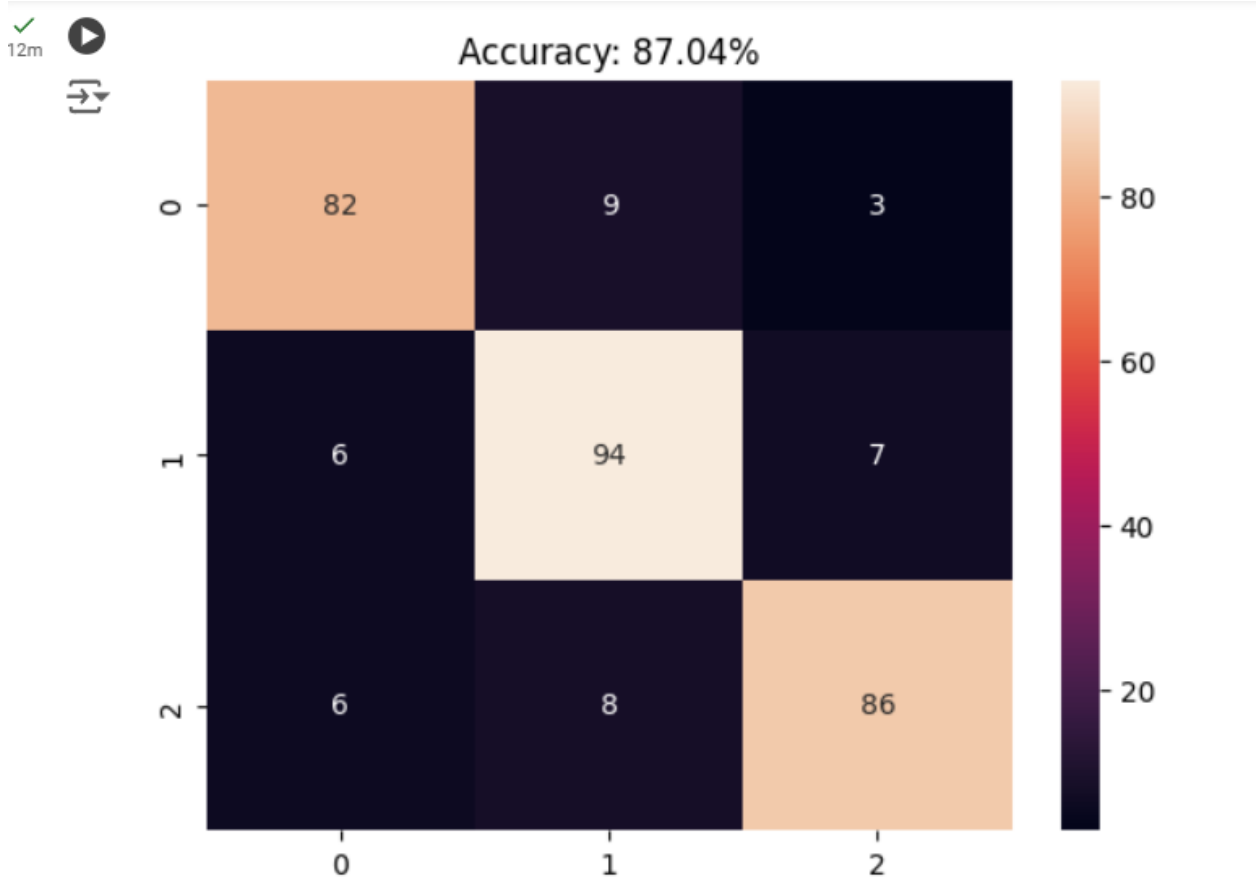


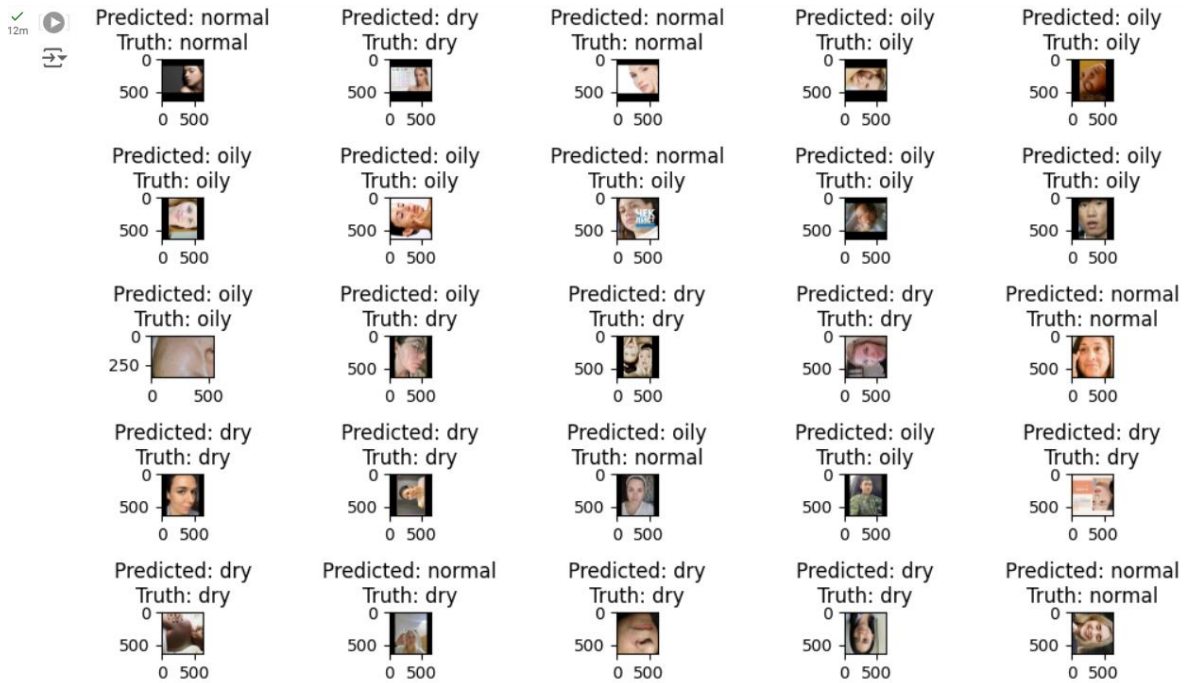
The EfficientNet-V2 model design is with an inverted residual block or MBConv Block, both of which contribute to compact size and rapid training times. A residual block is used in image models to solve efficiency issues, and this technique has been widely implemented in mobile CNN models. The 1x1 convolution in the inverted residual block minimizes the parameters, while 3x3 convolution layers use depth-wise convolution to conserve processing resources. Depth-wise convolution is a separate operation from the standard convolution, which typically transforms the feature by multiplying it repeatedly by some factor. This approach is commonly used in CNN models tuned for use

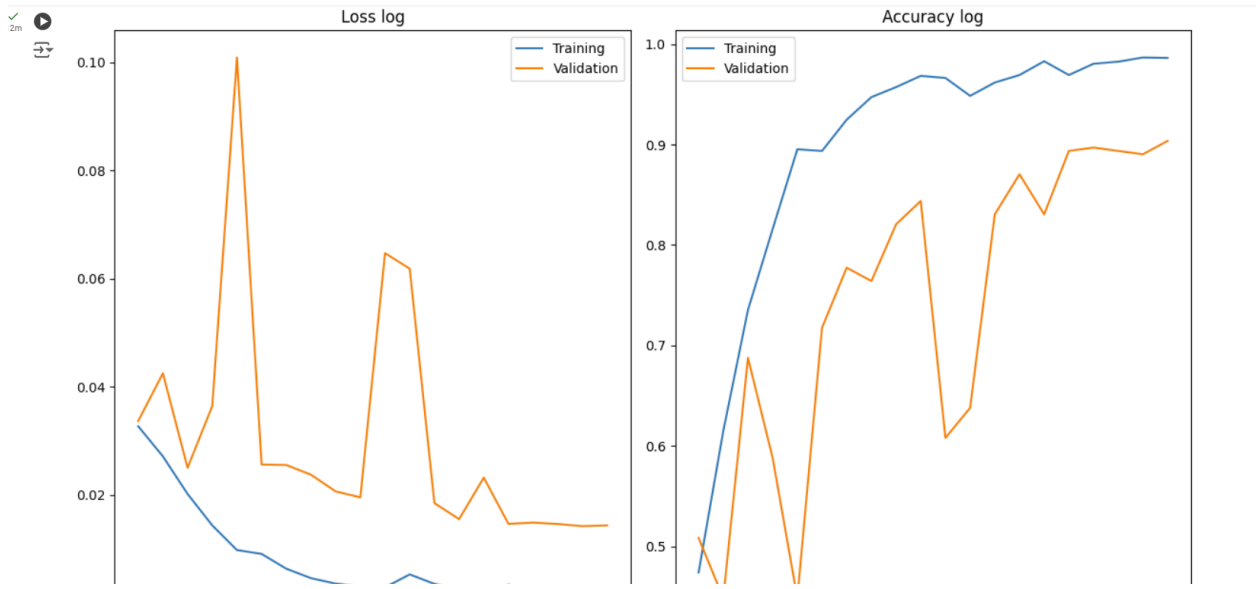
on mobile devices. Once the feature has been extracted from the 3x3 convolution layer, another round of field restriction with the 1x1 convolution layer is required. The MBConv building block struggles to deal with huge feature images or large overall image sizes. To speed up the traditional MBConv procedure, the Fused Inverted Residual or Fused MBconv was introduced by combining the 1x1 Conv and 3x3 Conv into a single 3x3 Conv to accelerate the model to the next level.

5.1.4 Evaluation:

Test Set Performance: Achieved an accuracy of 87.04% on the test set, demonstrating the model's effectiveness in classifying skin types.




Predicted: normal
Truth: normal
Predicted: dry
Truth: dry
Predicted: normal
Truth: normal
Predicted: oily
Truth: oily
Predicted: oily
Truth: oily
Predicted: oily
Truth: oily
Predicted: oily
Truth: oily
Predicted: normal
Truth: oily
Predicted: oily
Truth: oily
Predicted: oily
Truth: oily
Predicted: oily
Truth: oily
Predicted: oily
Truth: dry
Predicted: dry
Truth: dry
Predicted: dry
Truth: dry
Predicted: normal
Truth: normal
Predicted: dry
Truth: dry
Predicted: dry
Truth: dry
Predicted: oily
Truth: normal
Predicted: oily
Truth: oily
Predicted: dry
Truth: dry
Predicted: dry
Truth: dry
Predicted: normal
Truth: dry
Predicted: dry
Truth: dry
Predicted: dry
Truth: dry
Predicted: normal
Truth: normal


Loss log
Training
Validation
Accuracy log
Training
Validation

5.1.5 Comparison Between Models:

Algorithms	Accuracy
ResNet Baseline	**65%**
Knn	**78%**
ResNet50	**80%**
EfficientnetV2	**87.04%**

5.1.6 Conclusion:

Summary:

- EfficientNet V2 S effectively classifies skin types.
- Achieved high accuracy.

Challenges: Data preprocessing, model fine-tuning.

Solutions: Data augmentation, optimal parameter tuning.

Future Work:

- **Explore other model architectures**: Consider experimenting with different architectures.
- **Increase dataset size**: Collect more data for better generalization.
- **Integrate model into application**: Deploy the model for real-time skin type classification.

5.1.7 Machine Learning Model Snapshot:

Skin Detection Model

```python
@app.route('/predict', methods=['POST'])
def predict_route():
    if 'file' not in request.files:
        return jsonify({'error': 'No file part'}), 400

    file = request.files['file']

    if file.filename == '':
        return jsonify({'error': 'No selected file'}), 400

    if file:
        allowed_extensions = {'jpg', 'jpeg', 'png', 'gif'}
        if '.' in file.filename and file.filename.rsplit('.', 1)[1].lower() not in allowed_extensions:
            return jsonify({'error': 'Unsupported file type'}), 400

        try:
            image_bytes = file.read()
            prediction = predict(image_bytes)
            return jsonify({'prediction': prediction})
        except Exception as e:
            return jsonify({'error': str(e)}), 500

    return jsonify({'error': 'Failed to process the file'}), 400

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5001, debug=True)
```

5.2 Backend Snapshot

Add To Cart:

```dart
ClientController.dart
Future<void> addToCart(String productId) async {
  try {
    final uid = await _authService.getUserId();  // Ensure this method is correct and returns a valid UID
    final cartRef = FirebaseFirestore.instance
        .collection('carts')
        .doc(uid)
        .withConverter<Cart>(
      fromFirestore: (snapshot, _) => Cart.fromFirestore(snapshot),
      toFirestore: (cart, _) => cart.toFirestore(),
    );

    final snapshot = await cartRef.get();

    if (!snapshot.exists) {
      // Create a new cart with one unit of the product
      final newCart = Cart(productQuantities: {productId: 1});
      await cartRef.set(newCart);
    } else {
      final cart = snapshot.data();
      if (cart == null) {
        print("Failed to retrieve cart data");
        return;
      }
      final productQuantities = Map<String, int>.from(cart.productQuantities);
      // Add one unit of the product to the cart
      if (productQuantities.containsKey(productId)) {
        productQuantities[productId] = productQuantities[productId]! + 1;
      } else {
        productQuantities[productId] = 1;
      }
      await cartRef.set(Cart(productQuantities: productQuantities));
    }
```

Remove From Cart:

```dart
ClientController.dart

Future<void> removeFromCart(String productId) async {
  try {
    final uid = await _authService.getUserId();
    DocumentReference<Cart> cartRef = FirebaseFirestore.instance
        .collection('carts')
        .doc(uid)
        .withConverter<Cart>(
      fromFirestore: (snapshot, _) => Cart.fromFirestore(snapshot),
      toFirestore: (cart, _) => cart.toFirestore(),
    );

    DocumentSnapshot<Cart> snapshot = await cartRef.get();

    if (snapshot.exists) {
      Cart cart = snapshot.data()!;
      final productQuantities = Map<String, int>.from(cart.productQuantities);
      if (productQuantities.containsKey(productId)) {
        productQuantities.remove(productId);
        await cartRef.set(Cart(productQuantities: productQuantities));
      }
    }
  } catch (error) {
    print("Failed to remove product from cart: $error");
  }
}
```

Update Client Data:

```dart
Future<String> updateClientData({required Client client}) async {
final uid = await _authService.getUserId();
if (uid != null) {
  try {
    await FirebaseFirestore.instance.collection('clients').doc(uid).update(client.toFirestore());
    return "Client data updated successfully.";
  } catch (error) {
    return "Failed to update client data.";
  }
} else {
  return "User ID not found.";
}
}
```

Add Review:

```dart

Future<void> addReview(String productId, String review) async {
 final FirebaseFirestore _db = FirebaseFirestore.instance;
 try {
   final CollectionReference<Product> productsRef = _db
       .collection('products')
       .withConverter<Product>(
         fromFirestore: (snapshot, _) => Product.fromFirestore(snapshot),
         toFirestore: (product, _) => product.toFirestore(),
       );
   DocumentReference<Product> productDocRef = productsRef.doc(productId);

   DocumentSnapshot<Product> snapshot = await productDocRef.get();

   if (snapshot.exists) {
     Product product = snapshot.data()!;
     product.reviews ??= [];
     product.reviews!.add(review);

     await productDocRef.set(product);
   } else {
     throw Exception('Product not found.');
   }
 } catch (error) {
   print('Failed to add review: $error');
   rethrow;
 }
}
```

Add Rating:

```dart
Future<void> addRating(String productId, int rating) async {
  final FirebaseFirestore _db = FirebaseFirestore.instance;
  try {
    final CollectionReference<Product> productsRef = _db
        .collection('products')
        .withConverter<Product>(
          fromFirestore: (snapshot, _) => Product.fromFirestore(snapshot),
          toFirestore: (product, _) => product.toFirestore(),
        );
    DocumentReference<Product> productDocRef = productsRef.doc(productId);

    DocumentSnapshot<Product> snapshot = await productDocRef.get();

    if (snapshot.exists) {
      Product product = snapshot.data()!;

      double newAverageRating = ((product.averageRating ?? 0) * (product.totalRatings ?? 0) + rating) /
          ((product.totalRatings ?? 0) + 1);

      product.averageRating = newAverageRating;
      product.totalRatings = (product.totalRatings ?? 0) + 1;

      await productDocRef.set(product);
    } else {
      throw Exception('Product not found.');
    }
  } catch (error) {
    print('Failed to add rating: $error');
    rethrow;
  }
```

Filter Product By Category:

```dart
Future<List<List<Product>>> filterProductsByCategories(List<String> categories) async {
 FirebaseFirestore firestore = FirebaseFirestore.instance;
 try {
   QuerySnapshot<Map<String, dynamic>> snapshot = await firestore
       .collection('Products')
       .where('category', whereIn: categories)
       .get();

   List<List<Product>> productsArray = [];
   for (var category in categories) {
     List<Product> categoryProducts = snapshot.docs
         .where((doc) => doc.data()['category'] == category)
         .map((doc) => Product.fromFirestore(doc))
         .toList();
     productsArray.add(categoryProducts);
   }

   return productsArray;
 } catch (e) {
   // Handle error if any
   print('Error fetching products by categories: $e');
   return [];
 }
}
```

List Cart Content:

```dart
ClientController.dart
static Future<Map<Product, int>> listCartContents(String userId) async {
  FirebaseFirestore firestore = FirebaseFirestore.instance;
  try {
    DocumentSnapshot<Map<String, dynamic>> snapshot = await firestore
        .collection('carts')
        .doc(userId)
        .get();
    if (snapshot.exists) {
      Cart cart = Cart.fromFirestore(snapshot);
      Map<String, int> productQuantities = cart.productQuantities;
      Map<Product, int> productsWithQuantities = {};

      for (String productId in productQuantities.keys) {
        DocumentSnapshot<Map<String, dynamic>> productSnapshot = await firestore
            .collection('products')
            .doc(productId)
            .get();

        if (productSnapshot.exists) {
          Product product = Product.fromFirestore(productSnapshot);
          int quantity = productQuantities[productId] ?? 0;
          productsWithQuantities[product] = quantity;
        } else {
          print('Product document with ID $productId does not exist.');
        }
      }
      return productsWithQuantities;
    } else {
      print('Cart document with ID does not exist.');
      return {};
    }
```

- **Implementation**

Frontend

Flutter + Dart

Our main technology for the front-end of our mobile application was Flutter. We chose Flutter because it offers the following benefits:

- Fast Development: With Flutter's hot reload feature, we can quickly and easily experiment, build UIs, add features, and fix bugs faster.

-Expressive and Flexible UI: Flutter's modern reactive framework allows for building beautiful UIs that react to changes in the state.

-Native Performance: Flutter compiles to ARM or Intel machine code as well as JavaScript, for fast performance on any device.

-Cross-Platform Development: Flutter allows us to write code once and deploy it across multiple platforms including iOS, Android, web, and desktop.

Backend

Dart + Firebase Database and Storage

Our main technology for the back-end of our mobile application was Dart combined with Firebase Database and Storage. We chose this stack because it offers the following benefits:

- Real-time Database: Firebase Realtime Database allows for storing and syncing data in real-time across all clients, making it easy to build collaborative applications.

- Scalable and Flexible: Firebase Storage provides a powerful, simple, and cost-effective object storage service, allowing us to store and serve user-generated content such as photos and videos.

- Integrated Authentication: Firebase Authentication provides backend services, easy-to-use SDKs, and ready-made UI libraries to authenticate users to our app.

- Minimal Configuration: With Firebase, we can focus on building the features our users need and less on managing infrastructure, authentication, or networking.

Machine Learning

Python

Python was the language of choice for the Machine Learning model because

almost all ML frameworks and libraries are supported in Python.

Flask

Flask was used to build a REST API that is used by the back-end application to

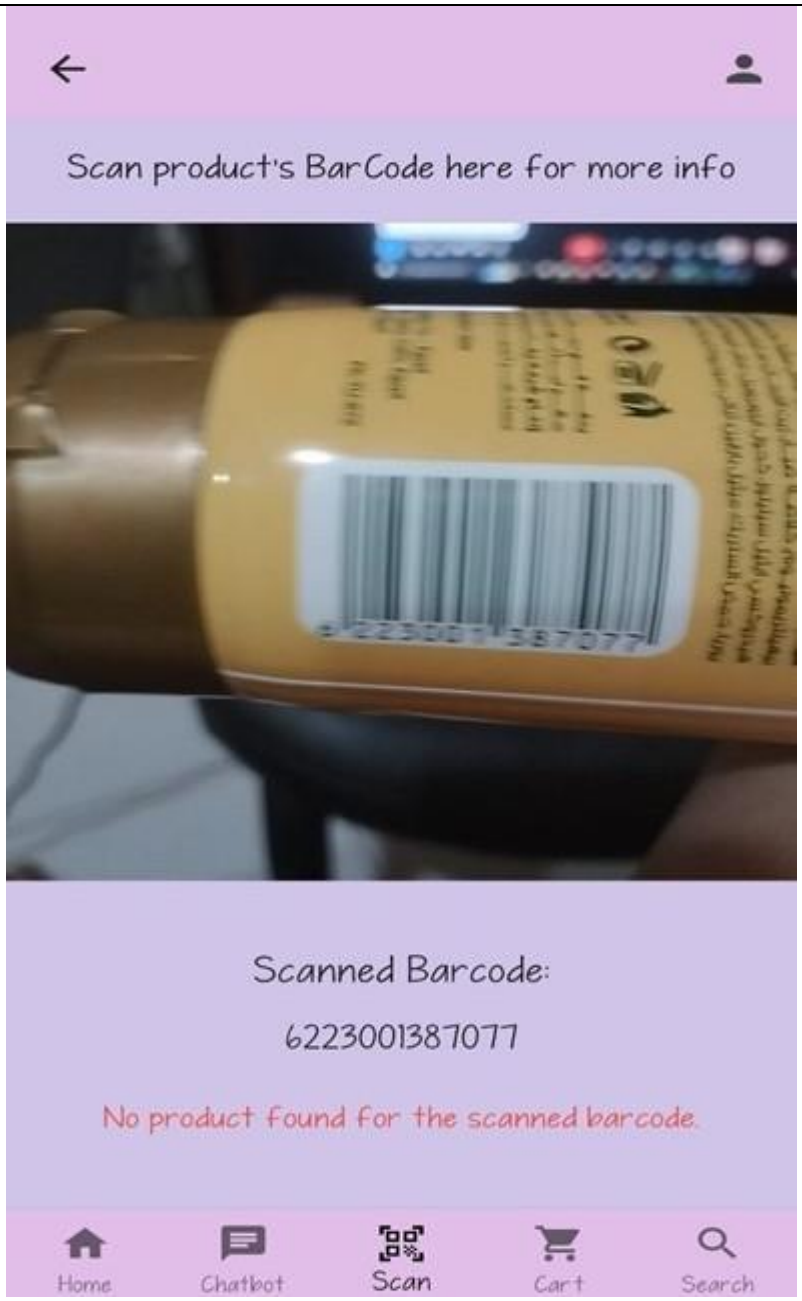
find suggested peers based on personality.

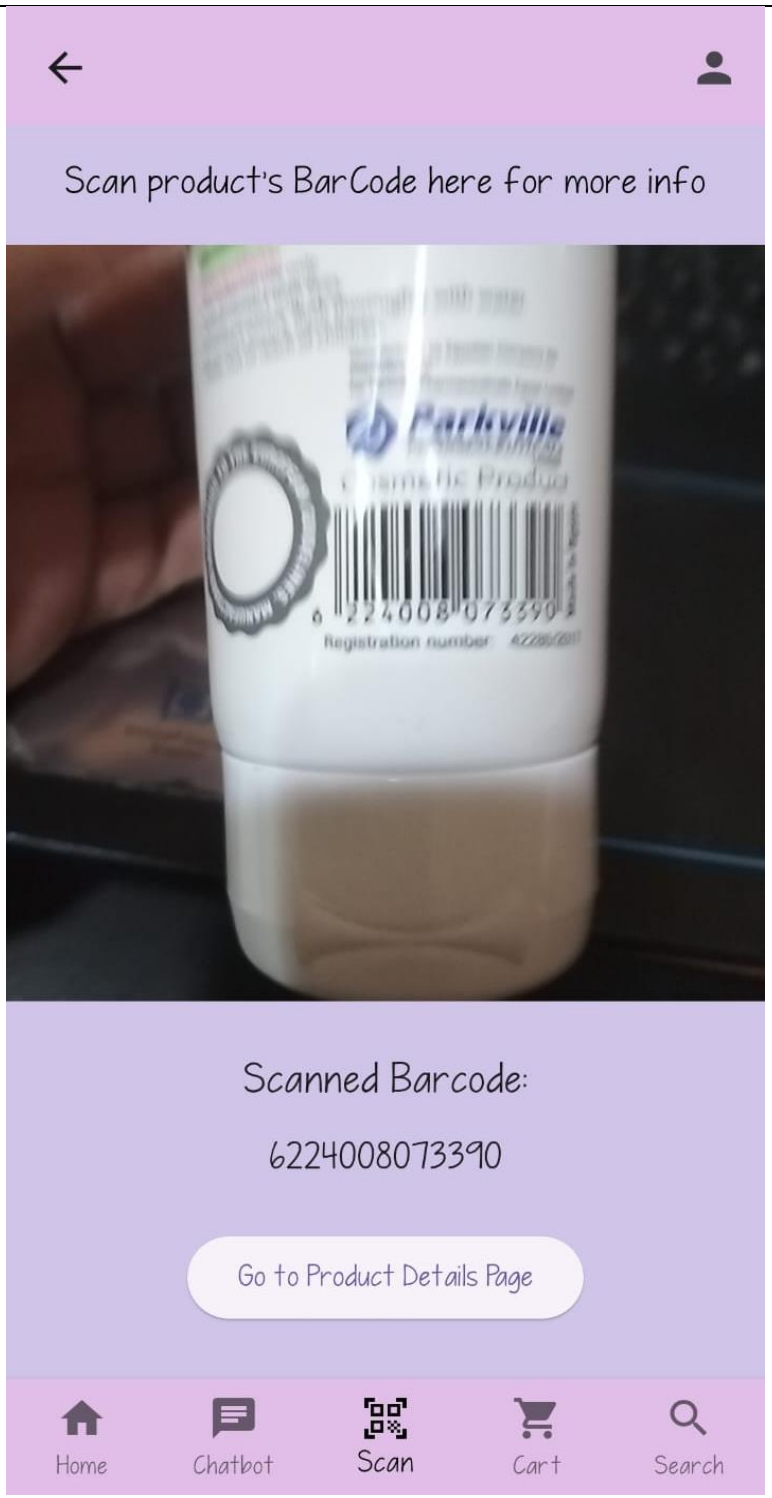
Testing

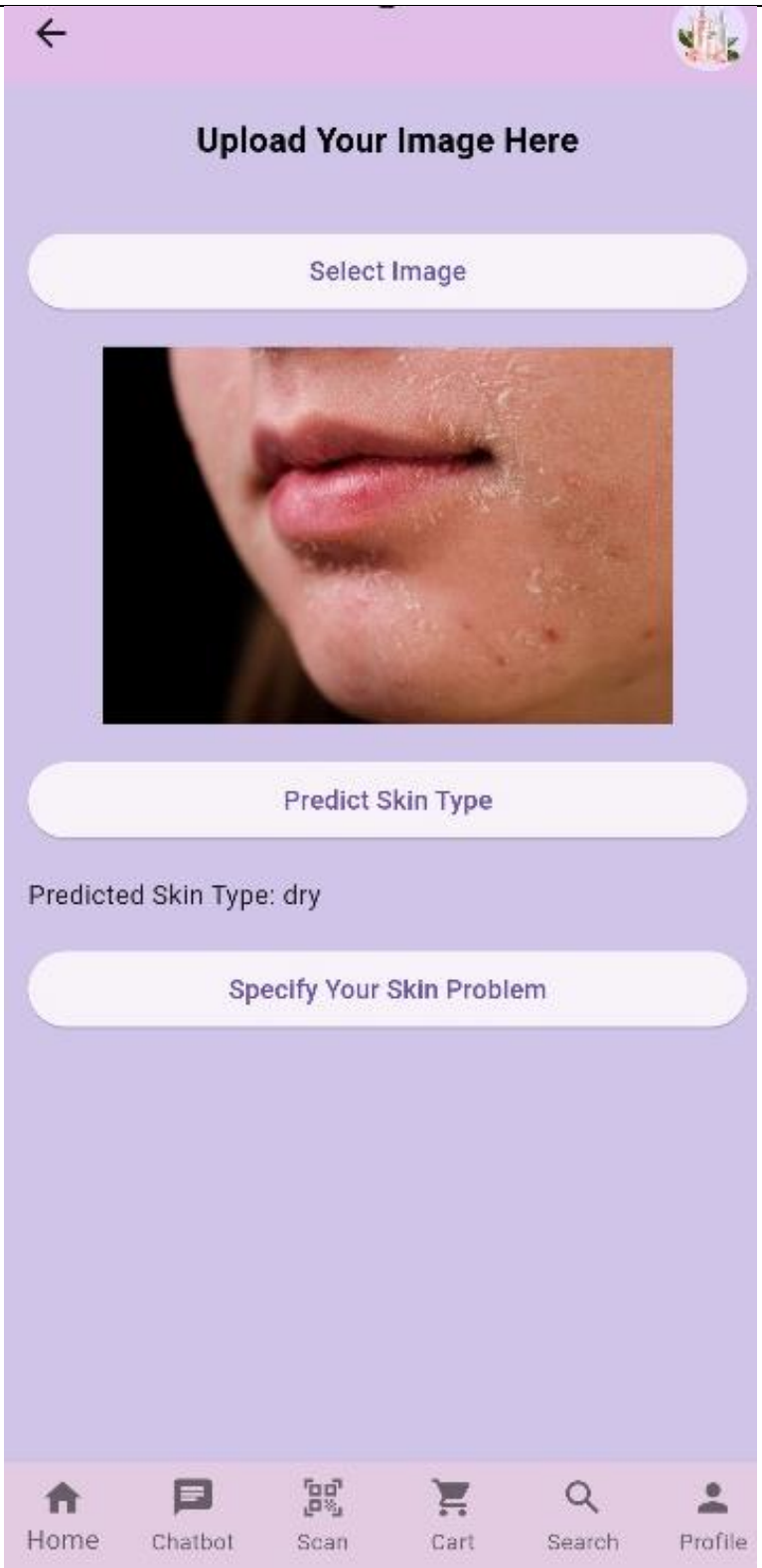
Functionality	Test Case	Input	Expected Output
scan product barcode	1	Barcode detected by the camera which is in the database	"No product found for the scanned barcode" message appears to the user.
	2	Barcode detected by the camera which is not in the database	"Go to Product Details Page" button appears.
skin type detection	3	User picture	"Predicted Skin Type: skin type" message appear to the user.
	4	Empty	"please upload image before detecting" alert is shown.
Specify skin concern	5	Skin concern and budget are selected.	Redirect to recommendation page.
	6	Skin concern is selected but budget not selected.	Redirect to recommendation page.
	7	Skin concern is not selected but budget is selected.	"please choose your concerns" alert is shown to the user.
suggest products	8	Skin type, budget and concern.	Products that follows the previous constraints.

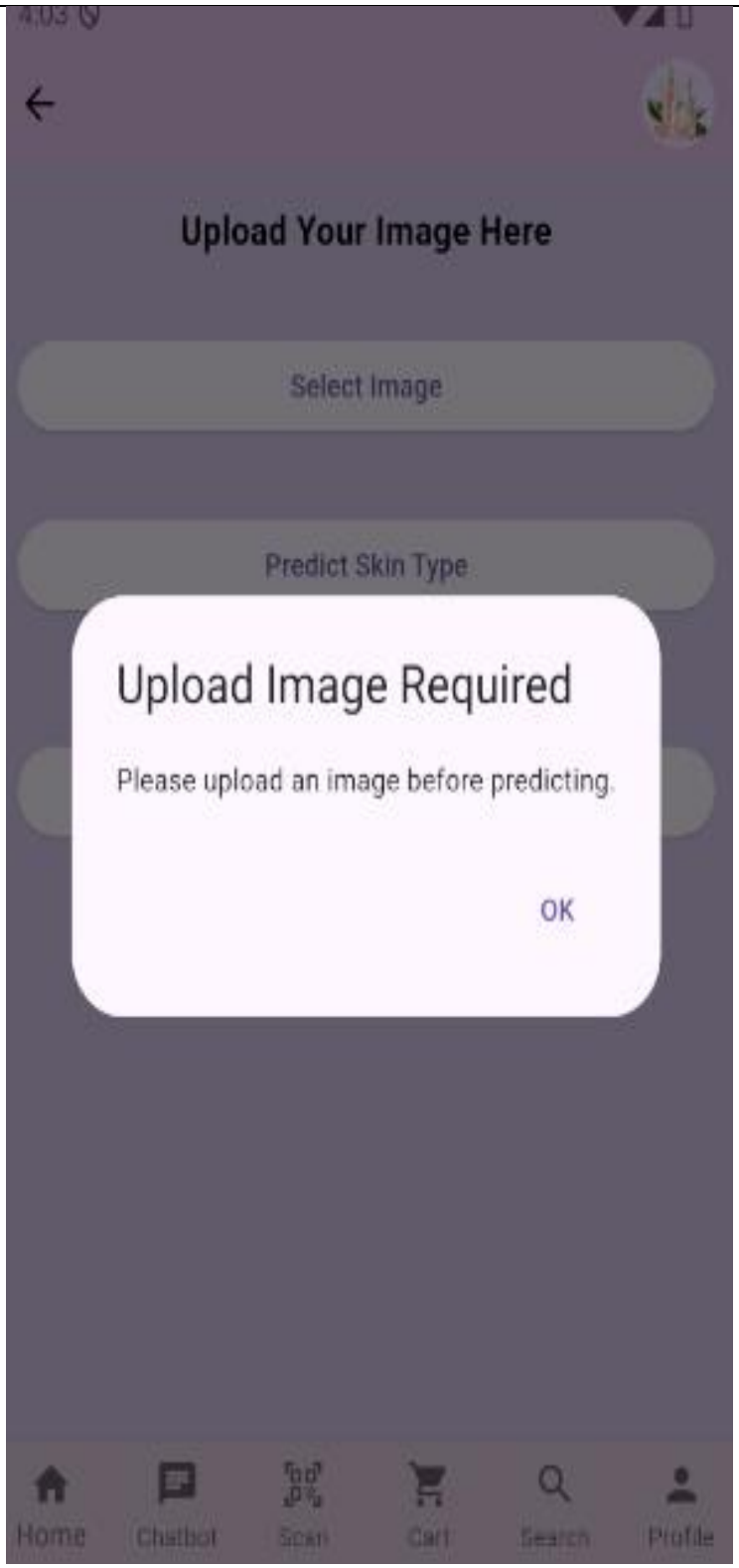
product shopping	9	Click add to cart in the product details page.	Empty
Make order	10	Click check out, choose visa or cash and enter all required data.	Redirect to final page.
	11	Choose visa or cash and enter part of the required data.	“Please fill in all required fields” message appear to the user.
Add rating	12	Rating from 1 to 5.	Empty
Add review	13	Review for the product.	Empty
search products	14	Add product name in the search.	Show the name, picture and price of the search results.
	15	Add wrong product name in the search.	Show “No results found” message.
chat bot	16	ask the chat bot, ex: please recommended a sun screen.	Chat bot recommend a product and give info about it.

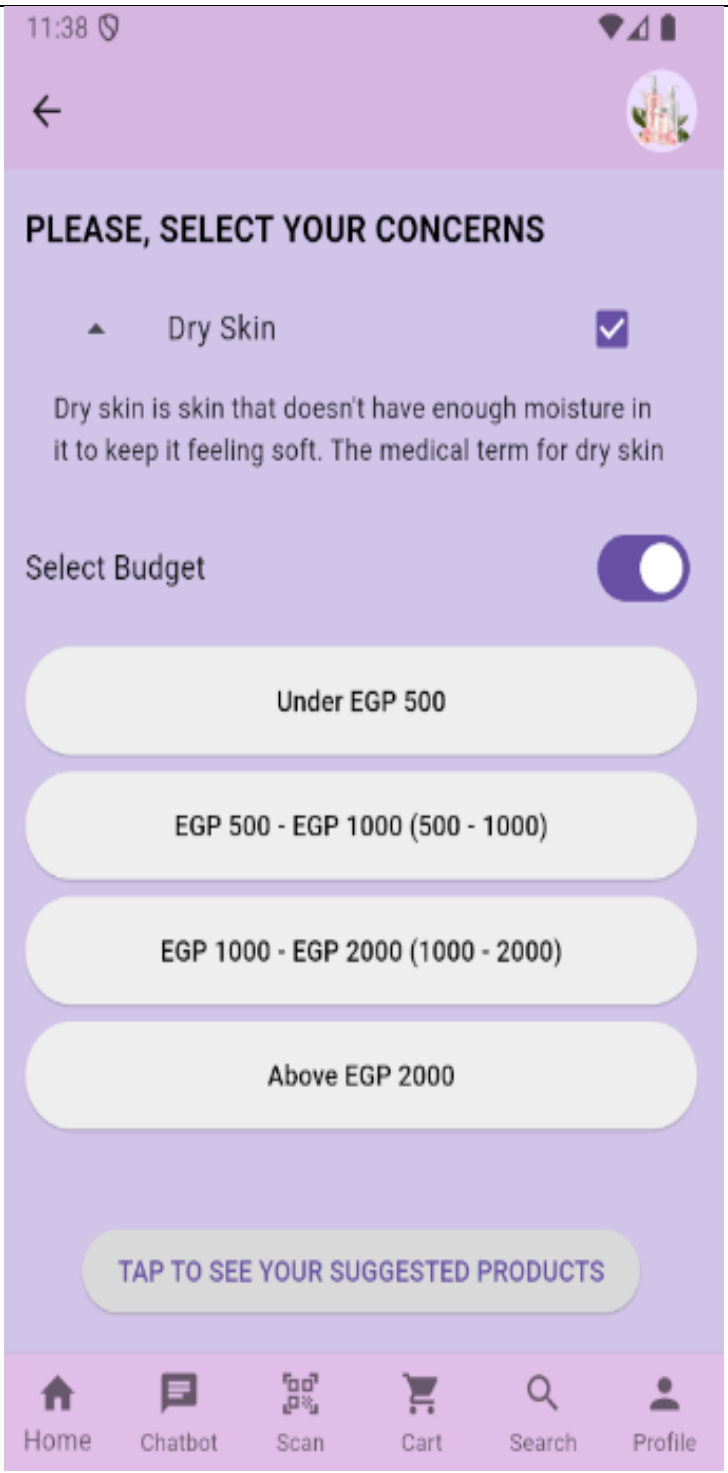
Applying the Test Cases

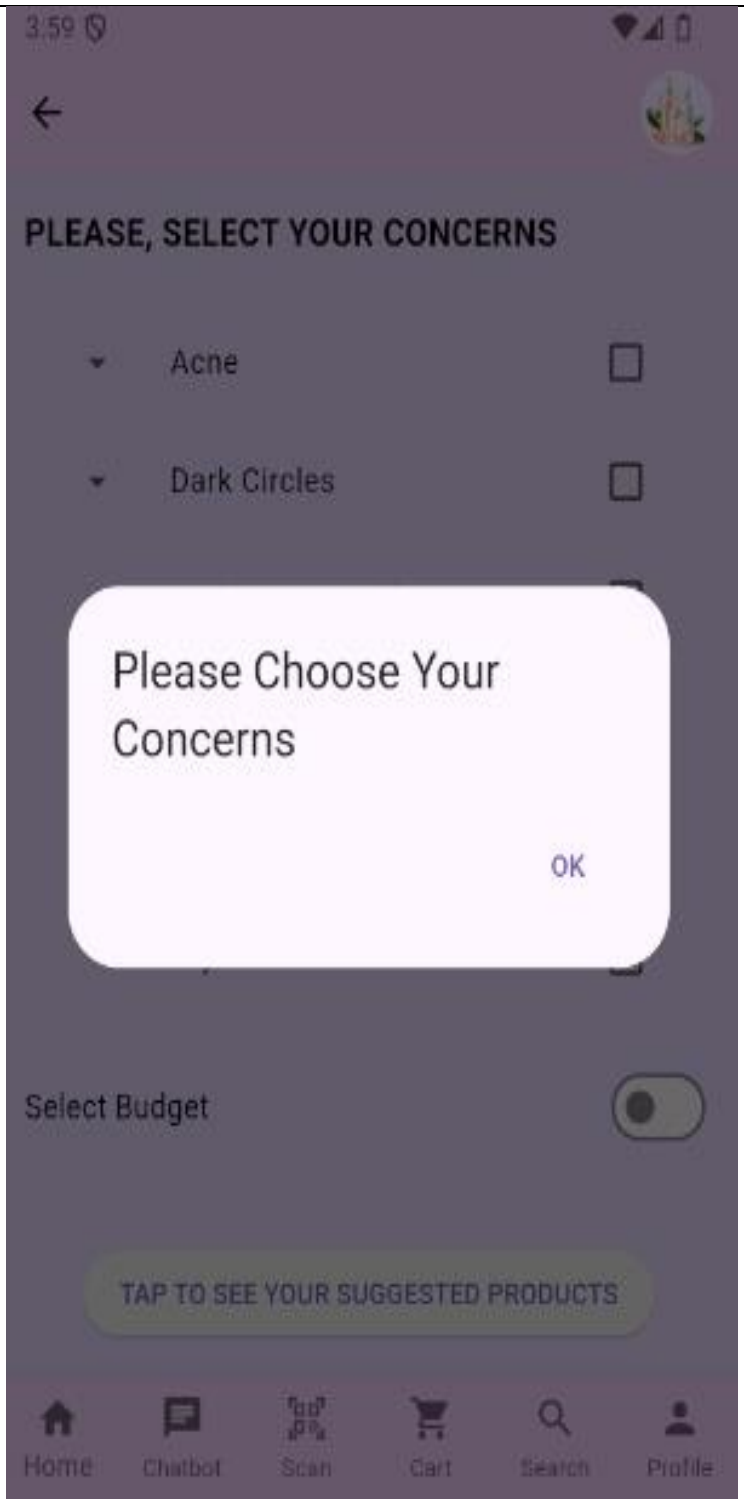
Test case	screenshot
1	

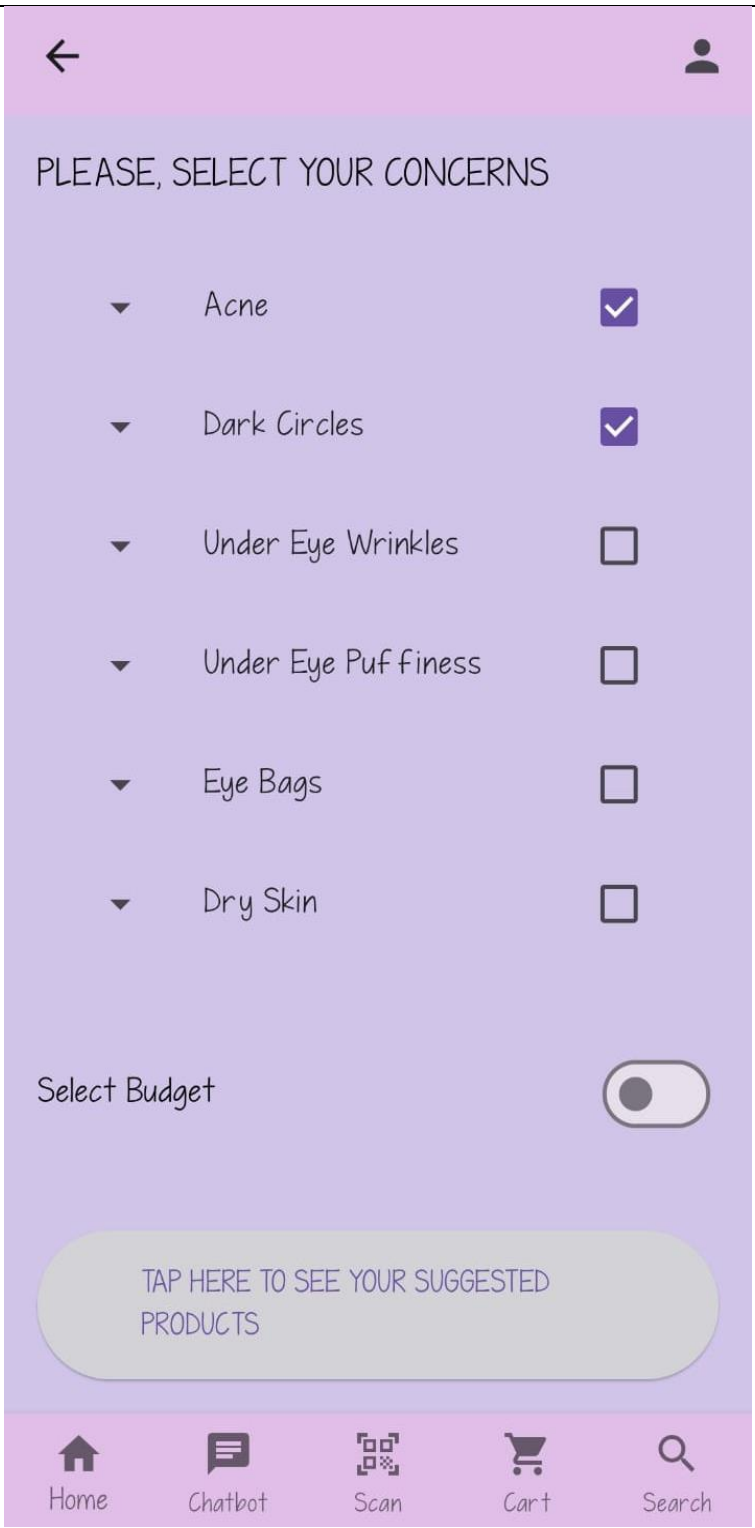
2	 Scan product's BarCode here for more info Scanned Barcode: 6224008073390 Go to Product Details Page Home Chatbot Scan Cart Search

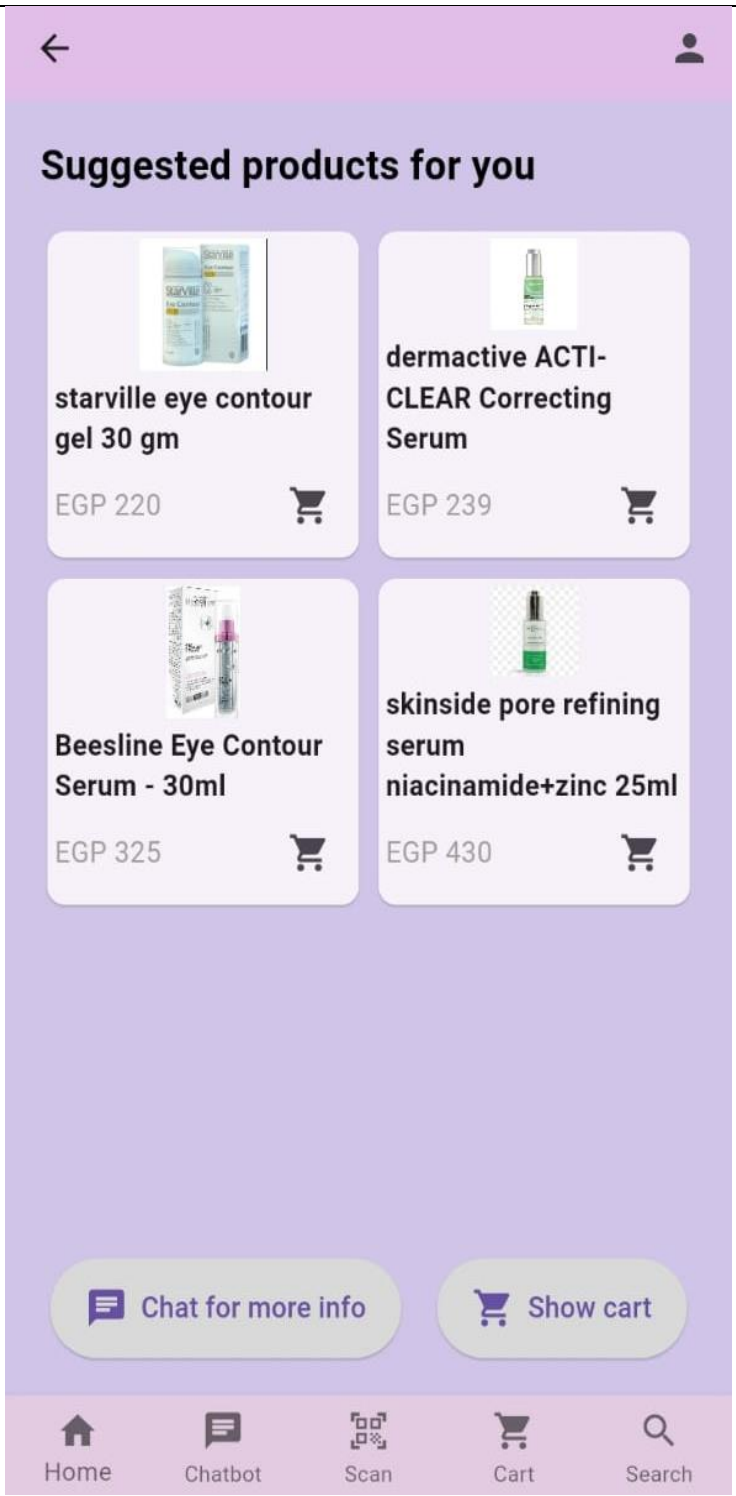
3		 Upload Your Image Here Select Image Predict Skin Type Predicted Skin Type: dry Specify Your Skin Problem Home Chatbot Scan Cart Search Profile

4	 Upload Your Image Here Select Image Predict Skin Type Upload Image Required Please upload an image before predicting. OK Home Chatbot Scan Cart Search Profile

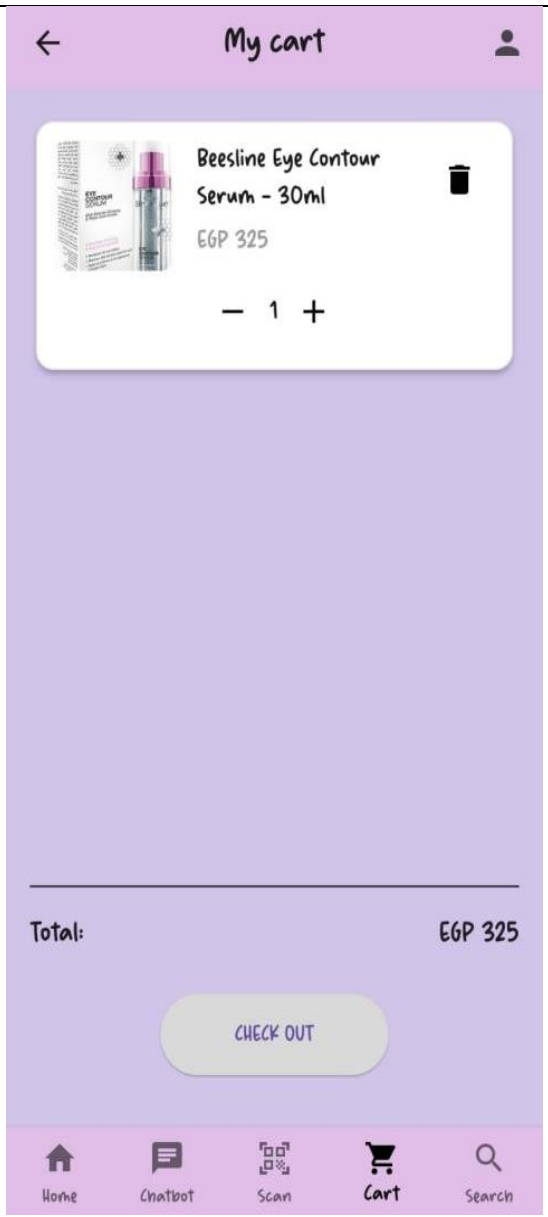
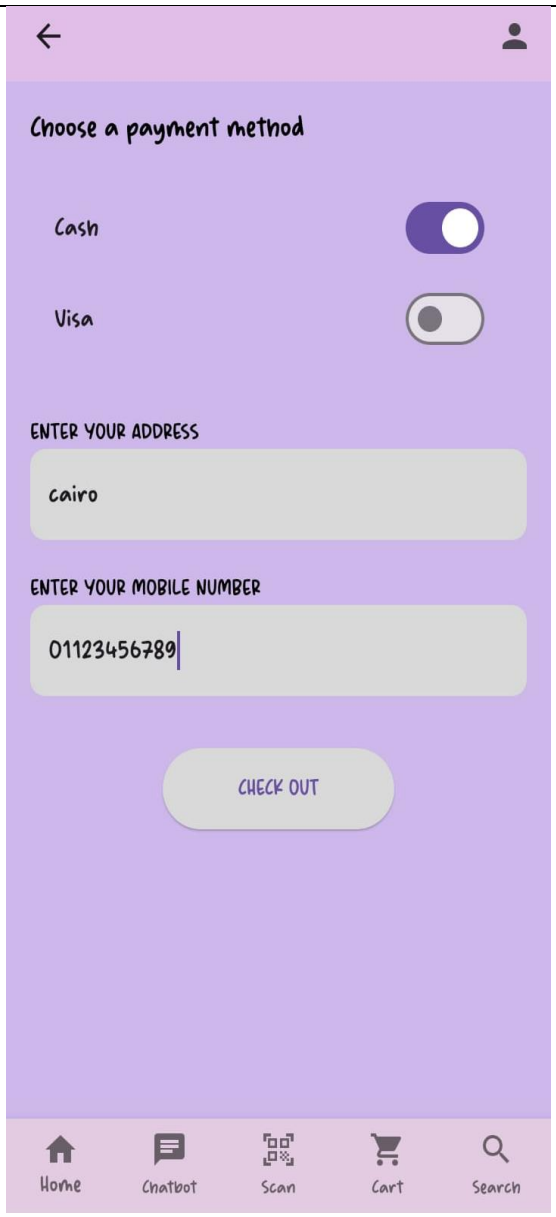
5		 11:38 PLEASE, SELECT YOUR CONCERNS Dry Skin Dry skin is skin that doesn't have enough moisture in it to keep it feeling soft. The medical term for dry skin Select Budget Under EGP 500 EGP 500 - EGP 1000 (500 - 1000) EGP 1000 - EGP 2000 (1000 - 2000) Above EGP 2000 TAP TO SEE YOUR SUGGESTED PRODUCTS Home Chatbot Scan Cart Search Profile

6		 3:59 PLEASE, SELECT YOUR CONCERNS Acne Dark Circles Please Choose Your Concerns OK Select Budget TAP TO SEE YOUR SUGGESTED PRODUCTS Home Chatbot Scan Cart Search Profile

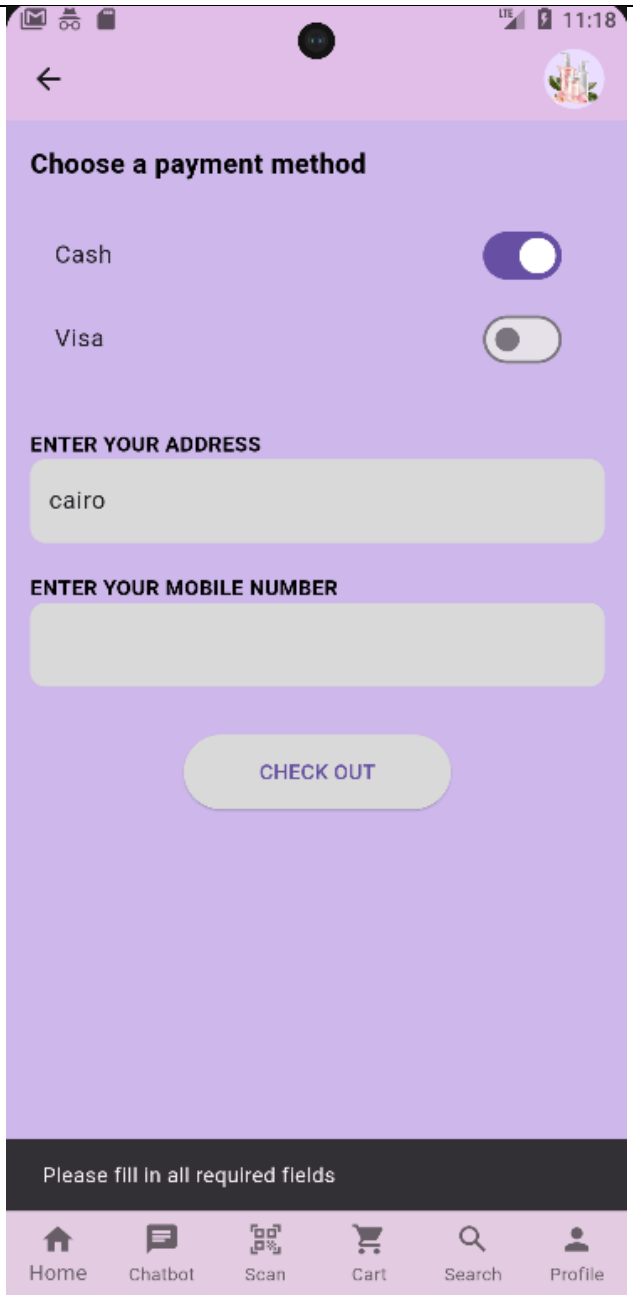
7		 PLEASE, SELECT YOUR CONCERNS Acne Dark Circles Under Eye Wrinkles Under Eye Puffiness Eye Bags Dry Skin Select Budget TAP HERE TO SEE YOUR SUGGESTED PRODUCTS Home Chatbot Scan Cart Search

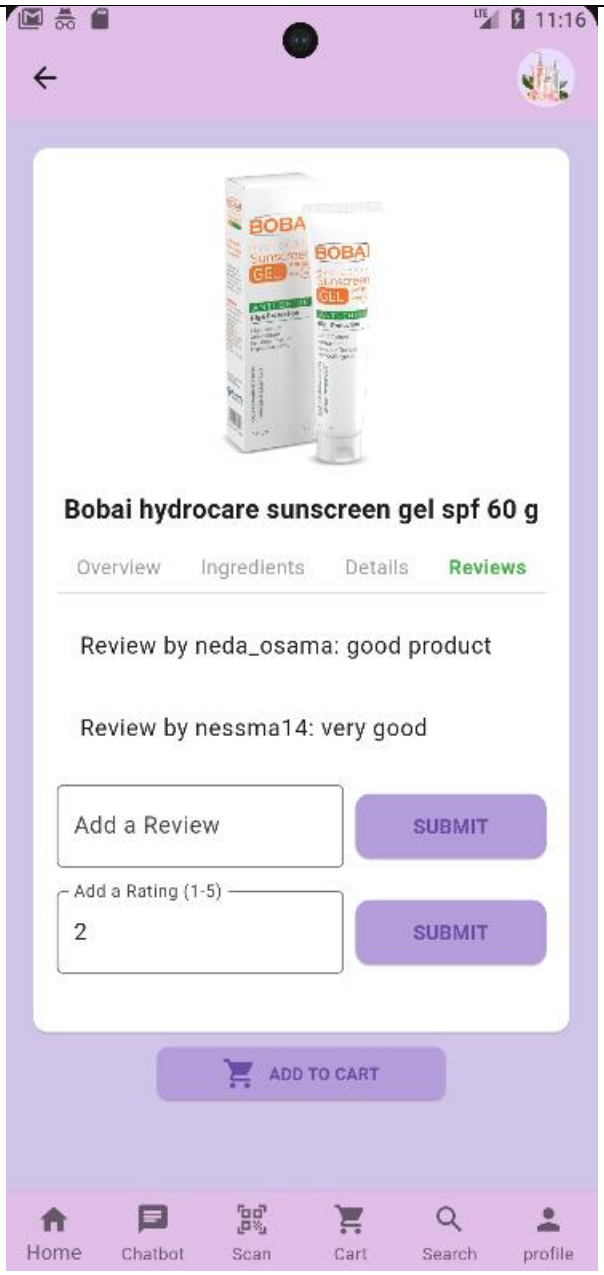
8		 Suggested products for you starville eye contour gel 30 gm EGP 220 dermactive ACTI-CLEAR Correcting Serum EGP 239 Beesline Eye Contour Serum - 30ml EGP 325 skinside pore refining serum niacinamide+zinc 25ml EGP 430 Chat for more info Show cart Home Chatbot Scan Cart Search

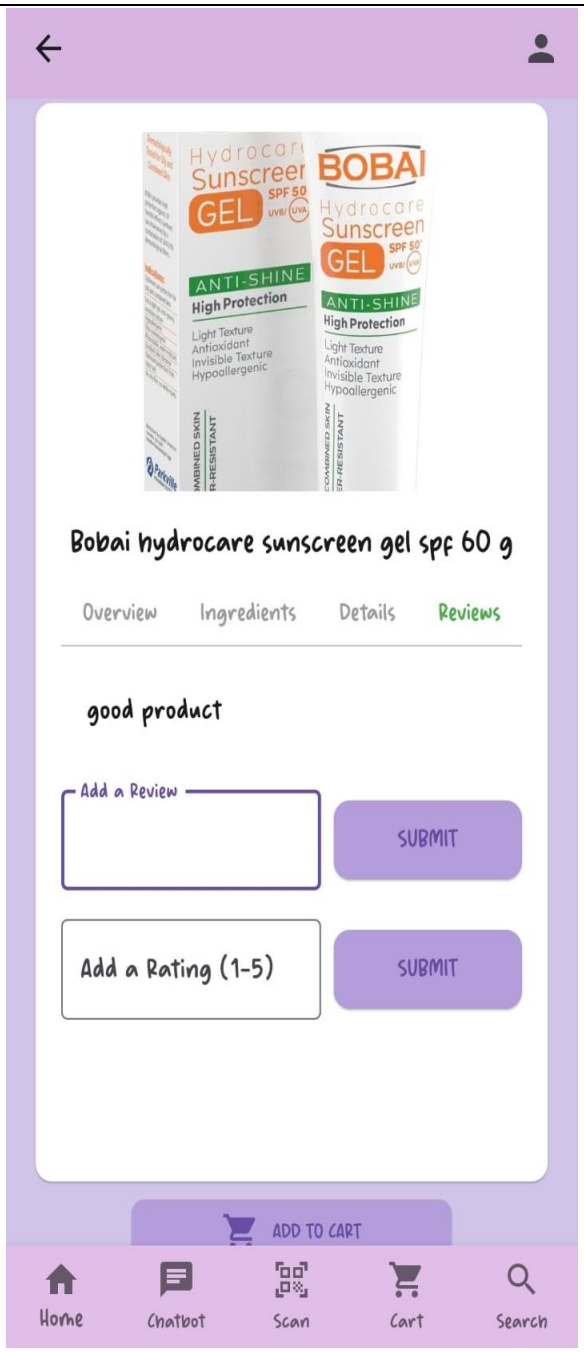
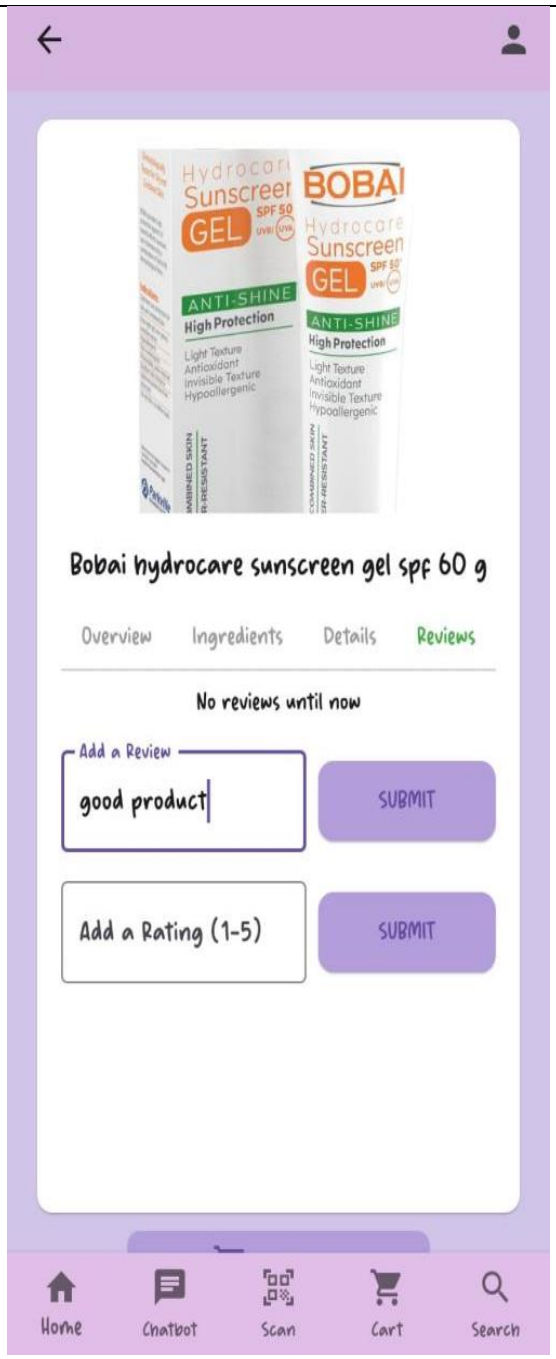
9	 11:54 CeraVe Hydrating Mineral Sunscreen 50 cerave hydrating mineral sunscreen spf 50 Overview Ingredients Details Reviews CeraVe Hydrating Mineral Sunscreen Face Lotion with Ceramides,Zinc Oxide,Titanium Dioxide,Niacinamide,Hyaluronic Acid Broad Spectrum SPF 50 is a 100%-mineral, oil-free sunscreen with titanium dioxide and zinc oxide that forms a protective barrier on your skin's surface to help reflect the sun's UVA and UVB rays without irritating sensitive skin Home Chatbot Scan Cart Search profile

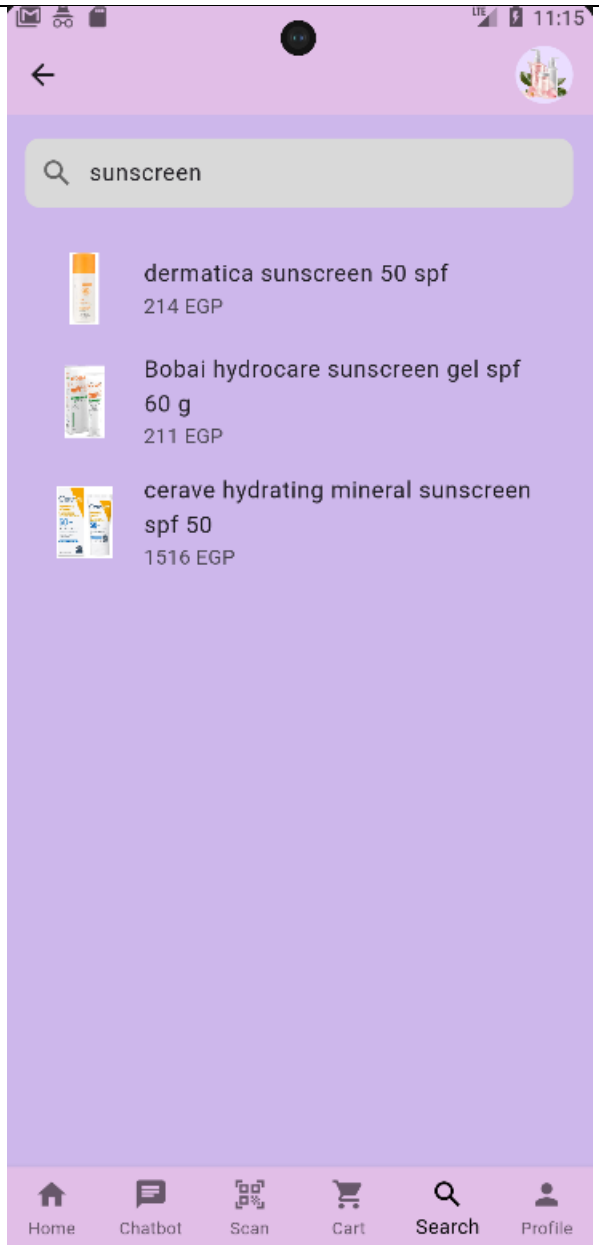
10	 My cart Beesline Eye Contour Serum - 30ml EGP 325 1 Total: EGP 325 CHECK OUT Home Chatbot Scan Cart Search  Choose a payment method Cash Visa ENTER YOUR ADDRESS cairo ENTER YOUR MOBILE NUMBER 01123456789 CHECK OUT Home Chatbot Scan Cart Search	

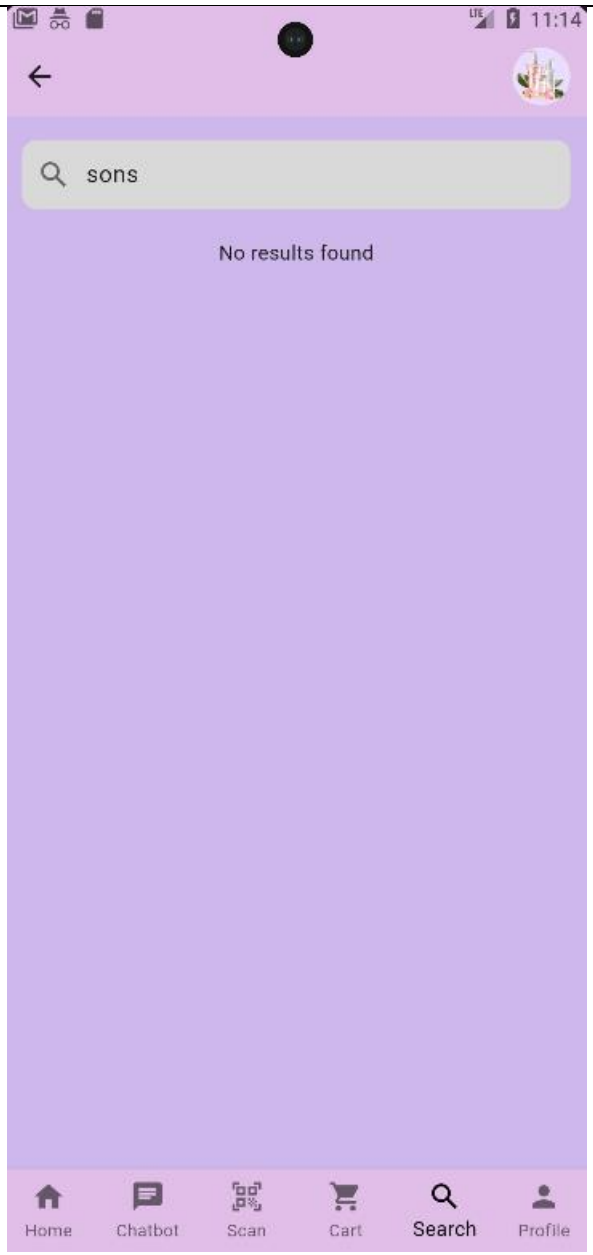

12:19
Your order is confirmed
Total Price: $2061.00
Name: mirna hatem
Phone: 1115586321
Address: haram
IF you like to go to home page again please
CLICK HERE
Home
Chatbot
Scan
Cart
Search
Profile

11		 11:18 Choose a payment method Cash Visa ENTER YOUR ADDRESS cairo ENTER YOUR MOBILE NUMBER CHECK OUT Please fill in all required fields Home Chatbot Scan Cart Search Profile

12	 11:16 Bobai hydrocare sunscreen gel spf 60 g Overview Ingredients Details Reviews Review by neda_osama: good product Review by nessma14: very good Add a Review SUBMIT Add a Rating (1-5) 2 SUBMIT ADD TO CART Home Chatbot Scan Cart Search profile

13	 BOBAI Hydrocare Sunscreen GEL SPF 50 ANTI-SHINE High Protection Light Texture Antioxidant Invisible Texture Hypoallergenic Bobai hydrocare sunscreen gel spf 60 g Overview Ingredients Details Reviews good product Add a Review SUBMIT Add a Rating (1-5) SUBMIT ADD TO CART Home Chatbot Scan Cart Search  BOBAI Hydrocare Sunscreen GEL SPF 50 ANTI-SHINE High Protection Light Texture Antioxidant Invisible Texture Hypoallergenic Bobai hydrocare sunscreen gel spf 60 g Overview Ingredients Details Reviews No reviews until now Add a Review good product SUBMIT Add a Rating (1-5) SUBMIT Home Chatbot Scan Cart Search

14	 11:15 sunscreen dermatica sunscreen 50 spf 214 EGP Bobai hydrocare sunscreen gel spf 60 g 211 EGP cerave hydrating mineral sunscreen spf 50 1516 EGP Home Chatbot Scan Cart Search Profile

15	 11:14 sons No results found Home Chatbot Scan Cart Search Profile

References

- https://www.sciencedirect.com/science/article/pii/S2405844023083846
- Install | Flutter
- Dart basics | Dart
- Official Figma YouTube Channel
- https://www.amazon.eg/s?k=face+skin+care&crid=2H24LGHQUS71R&sprefix=face+skin+care%2Caps%2C242&ref=nb_sb_noss_1
- https://www.cerave.eg/en/skincare?utm_source=google&utm_medium=paid_search&utm_content=45356&utm_campaign=&utm_term=brandeden&gad_source=1&gclid=Cj0KCQjws560BhCuARIsAHMqE0HmAZurTYXLYNRPGApMamn8BEb3S9rkZ0OaA8Fx-mYB7UReKV1oIcsaAqTwEALw_wcB
- https://eparkville.com/collections/shaan
- https://beesline.com/