**E-Commerce Flutter Application Report**



**Members :**

**-Lý Khải Minh-ITITSB22008**

**-Đỗ Minh Duy-ITITSB22029**

Table of Contents

***I. Introduction***

***II. Software Requirements***

***III. Design & Implementation***

***IV. Keys Takeaways***

***V. Conclusion***

1. **Project Overview**
2. **Key Features**
3. **2.1 Firebase Integration**
4. **2.2 Stripe Payment Integration**
5. **2.3 Custom UI and Assets**
6. **2.4 Responsive Web Integration**
7. **User Experience**
8. **Testing**
9. **Platform-Specific Configurations**
10. **Key Takeaways**
11. **Conclusion**

***I. Introduction***

**1. About the app**

**In the rapidly evolving field of software development, building cross-platform applications has become essential. The Flutter framework offers a unified solution for developing applications for Android, iOS, macOS, and Web.**

**This report details the development of an E-Commerce Flutter Application that integrates essential features such as Firebase for backend services, Stripe for payment processing, and custom UI components. The project emphasizes scalability, maintainability, and ease of use while delivering a seamless user experience.**

**The primary objective is to build a robust and dynamic e-commerce platform that serves both customers and administrators effectively, enabling smooth navigation, secure transactions, and responsive design across platforms.**

**2. Manipulation**

**The goal of the application is to:**

* **Allow users to browse products, add to cart, and make purchases.**
* **Provide an admin dashboard to manage products and orders.**
* **Ensure secure payment through Stripe integration.**

**3. User Mechanics**

**The mechanics of the application include:**

* **Navigation: Users navigate via bottom navigation bars and interactive screens.**
* **Product Management: Products can be viewed, edited, and deleted by admins.**
* **Secure Transactions: Users make purchases using Stripe for payment.**
* **Authentication: Secure login/signup through Firebase Authentication.**

***II. Software Requirements***

**1. What We Have**

1. **User-friendly, efficient, and responsive system.**
2. **Minimal maintenance cost (cloud-based backend).**
3. **Intuitive navigation and UI.**
4. **Optimized cross-platform builds.**

**2. What We Want**

1. **Develop the system within a limited cost and timeframe.**
2. **Ensure high-definition and responsive UI for all devices.**
3. **Design the system for scalability and future updates.**

**3. Working Tools and Platforms**

* **IDE: Visual Studio Code with Flutter SDK.**
* **Design Tools: Figma for UI prototyping, GIMP for image editing.**
* **Backend Tools: Firebase (Auth, Firestore, Storage).**
* **Payment Integration: Stripe API.**
* **Web Configuration:**
  + **index.html for the entry point​.**
  + **manifest.json for PWA support​.**

***III. Design & Implementation***

***\* Data Structures and Algorithms in Application Development***

**Efficient use of data structures and algorithms is essential for ensuring optimal performance and scalability. The following principles and techniques are employed in the development of the application:**

**1. Firebase Authentication**

A screenshot of a phone

Description automatically generated

**This code represents a Signup page for a Flutter-based E-Commerce application. It integrates Firebase Authentication for user registration, Firestore for database storage, and shared preferences for local storage.**

**2. Functional Breakdown**

**2.1 Core Features**

1. **User Registration:**
   * **Registers users with Firebase Authentication using an email and password.**
   * **On success:**
     + **Saves user information (Name, Email, Wallet, Id) to Firestore.**
     + **Stores user data locally using Shared Preferences.**
2. **Input Validation:**
   * **Ensures Name, Email, and Password fields are not empty using validators.**
3. **Error Handling:**
   * **Handles common Firebase exceptions:**
     + **Weak passwords (weak-password).**
     + **Duplicate accounts (email-already-in-use).**
4. **Navigation:**
   * **On successful signup: Redirects to the bottom navigation page (BottomNav).**
   * **Provides a link to navigate to the Login page.**
5. **UI Design:**
   * **Modern design with gradient backgrounds, form inputs, buttons, and clean text fields.**
   * **Organized using Stack, Material, Container, and Column widgets.**

**3. Code Components**

**3.1 State Management**

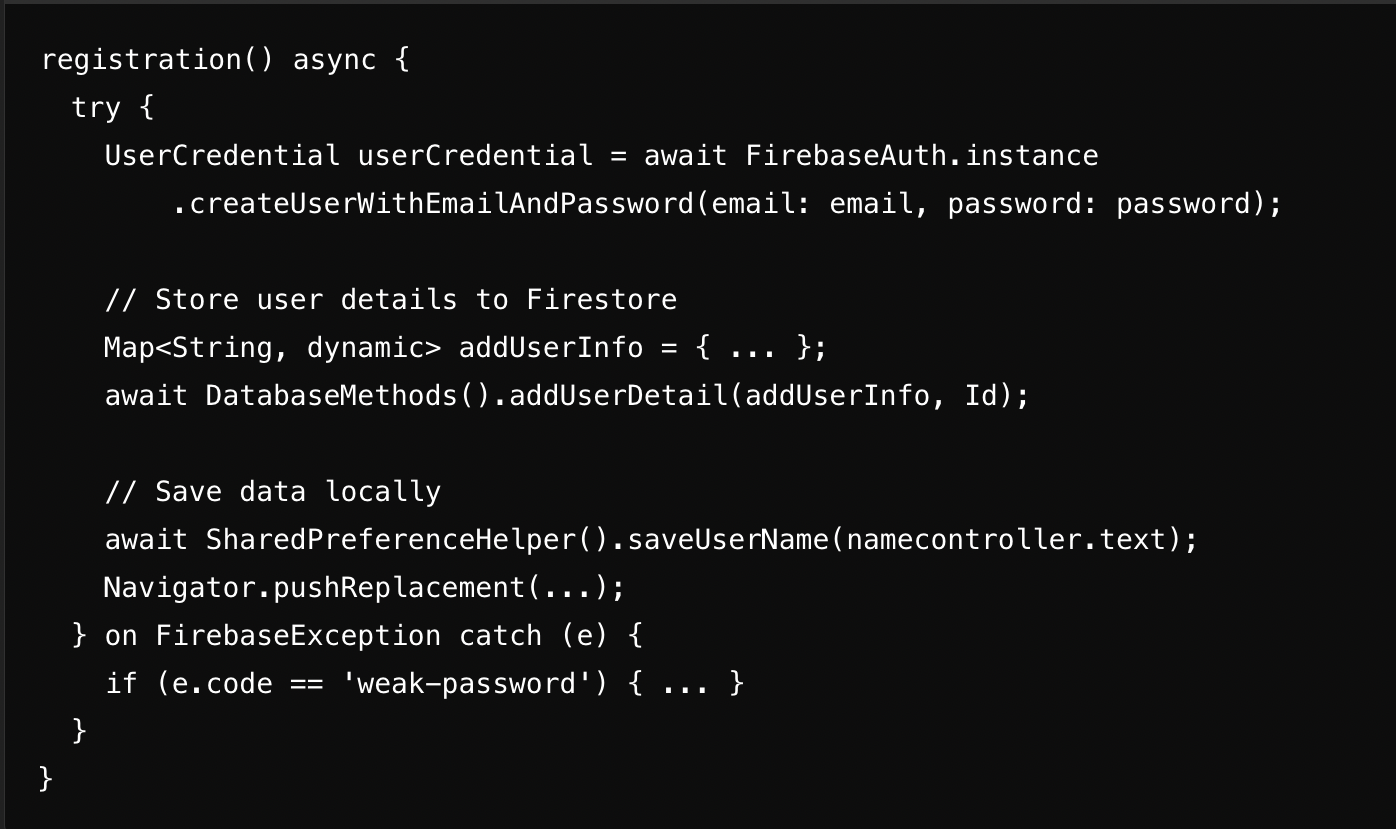
**The code uses StatefulWidget (\_SignupState) for managing dynamic data:**

**A screenshot of a computer program

Description automatically generated**

**3.2 Firebase Authentication**

**The registration method handles user signups:**

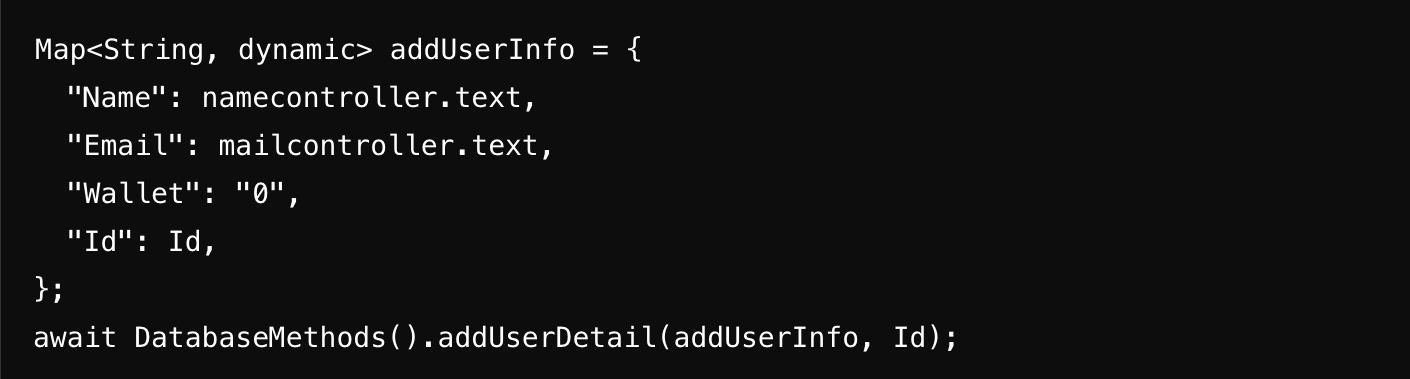
****

**DSA Principle:**

* **Authentication as a Service: Delegates user creation to Firebase Authentication, a cloud-based API.**

**3.3 Firestore Integration**

**User details are stored in Firestore using a DatabaseMethods class:**

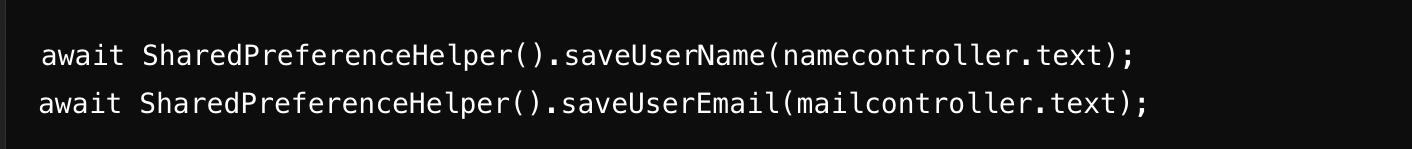
****

**DSA Principle:**

* **Data Persistence: Utilizes Firestore (NoSQL database) for storing structured user data.**

**3.4 Local Storage (Shared Preferences)**

**User information (Name, Email, Wallet, Id) is cached locally:**

**DSA Principle:**

* **Caching: Reduces redundant database reads by locally storing frequently used data.**

**3.5 Input Validation**

**Validators ensure form integrity before registration:**

**A computer screen with white text

Description automatically generatedDSA Principle:**

* **Input Consistency: Prevents invalid data from reaching backend APIs.**

**4. UI Design**

**The page uses a clean Material Design approach:**

* **Gradient Background: Enhances visual appeal.**
* **TextFormFields: For user input.**
* **Buttons: Styled with GestureDetector for navigation.**

**Widgets:**

* **Stack: Layers background, form, and content.**
* **Material: Adds elevation and shadow effects.**
* **Column: Organizes child widgets vertically.**

**Custom Styling:  
Utilizes AppWidget methods for consistent styles.**

**5. DSA Principles Applied**

**5.1 Modularity**

* **Separation of Concerns:**
  + **Firebase Authentication, Firestore operations, and UI code are modularized.**
  + **External classes like DatabaseMethods and SharedPreferenceHelper handle their respective tasks.**

**5.2 Data Persistence**

* **Firestore: Persistent storage for user data.**
* **Shared Preferences: Local caching for offline access.**

**5.3 Error Handling**

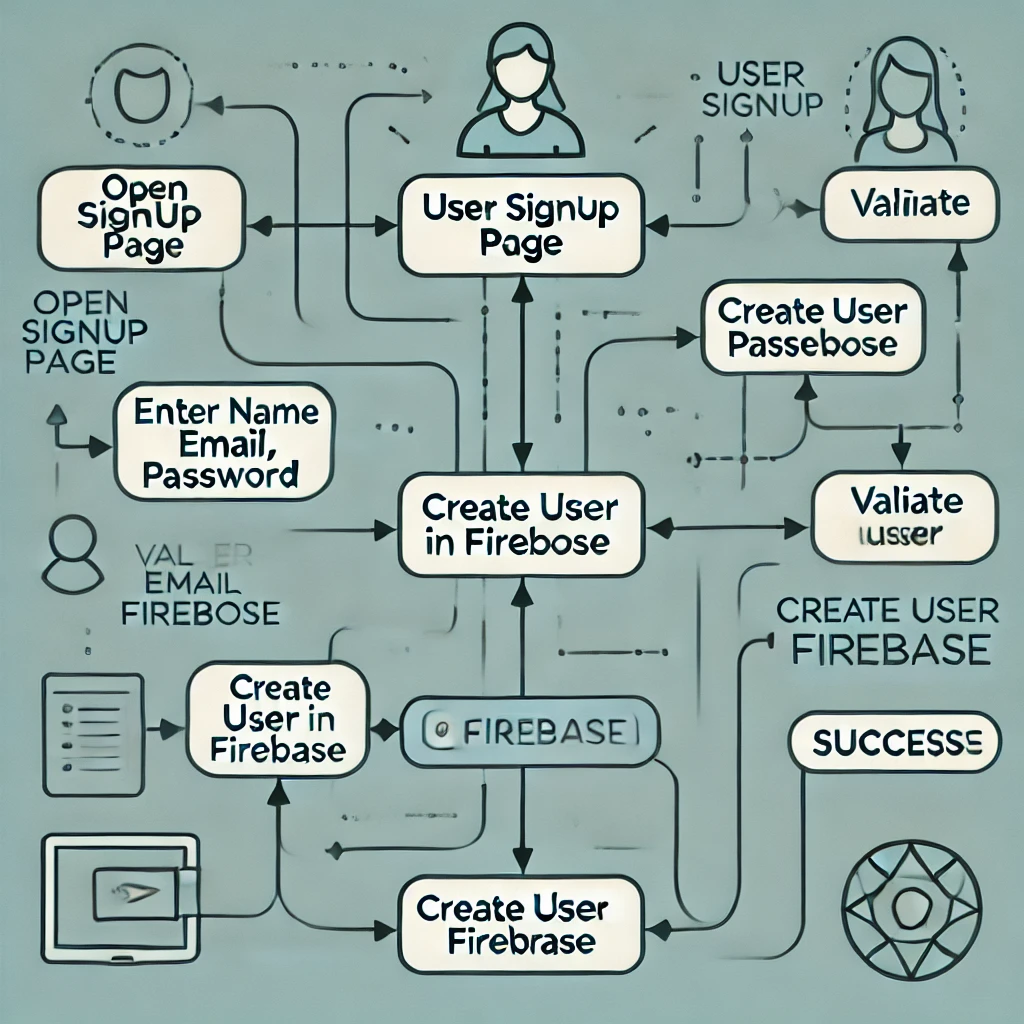
* **Robust Handling: Ensures user-friendly error messages for common Firebase exceptions.**

**5.4 Algorithm Design**

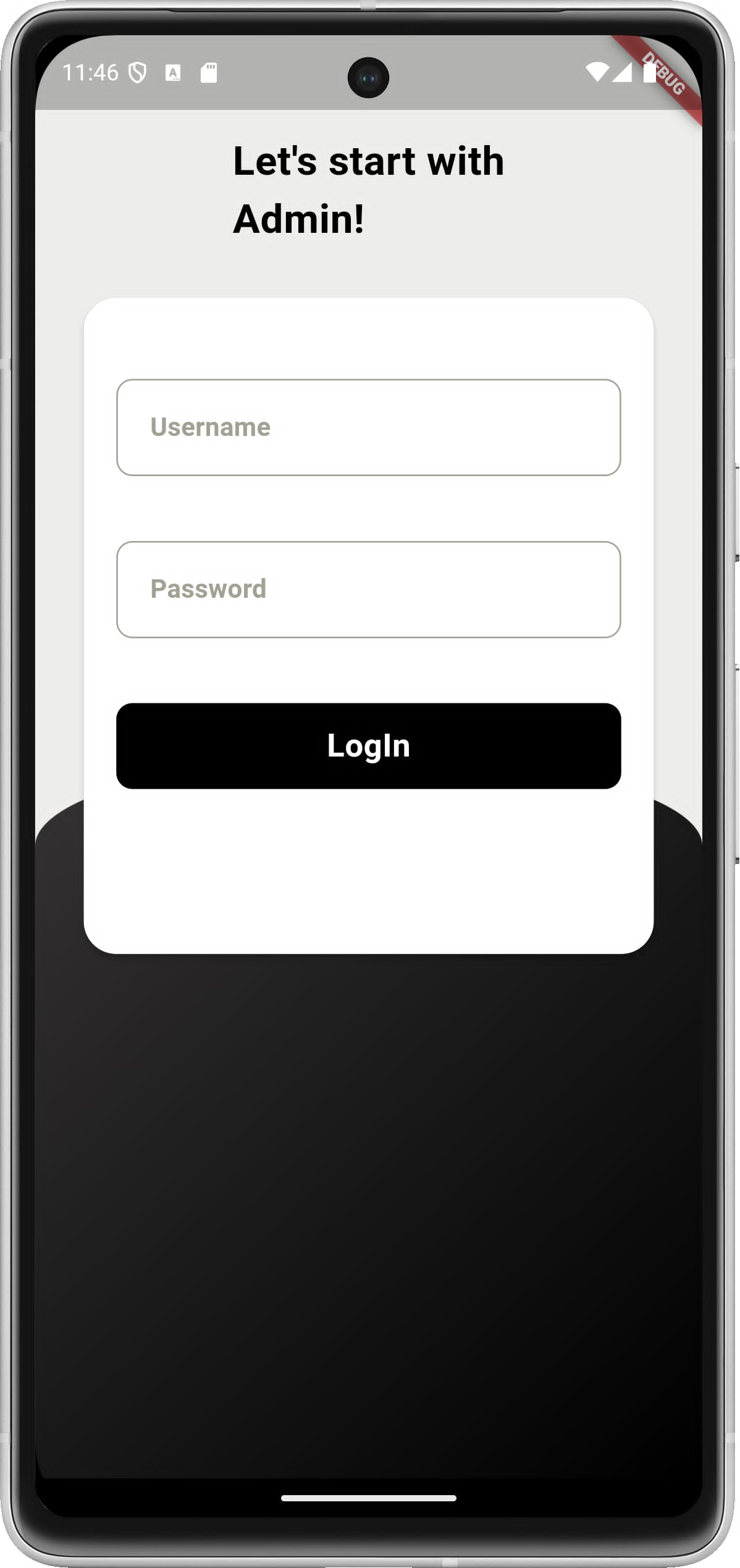
* **Unique ID Generation:  
  Uses randomAlphaNumeric(10) to ensure a unique Id for each user.**

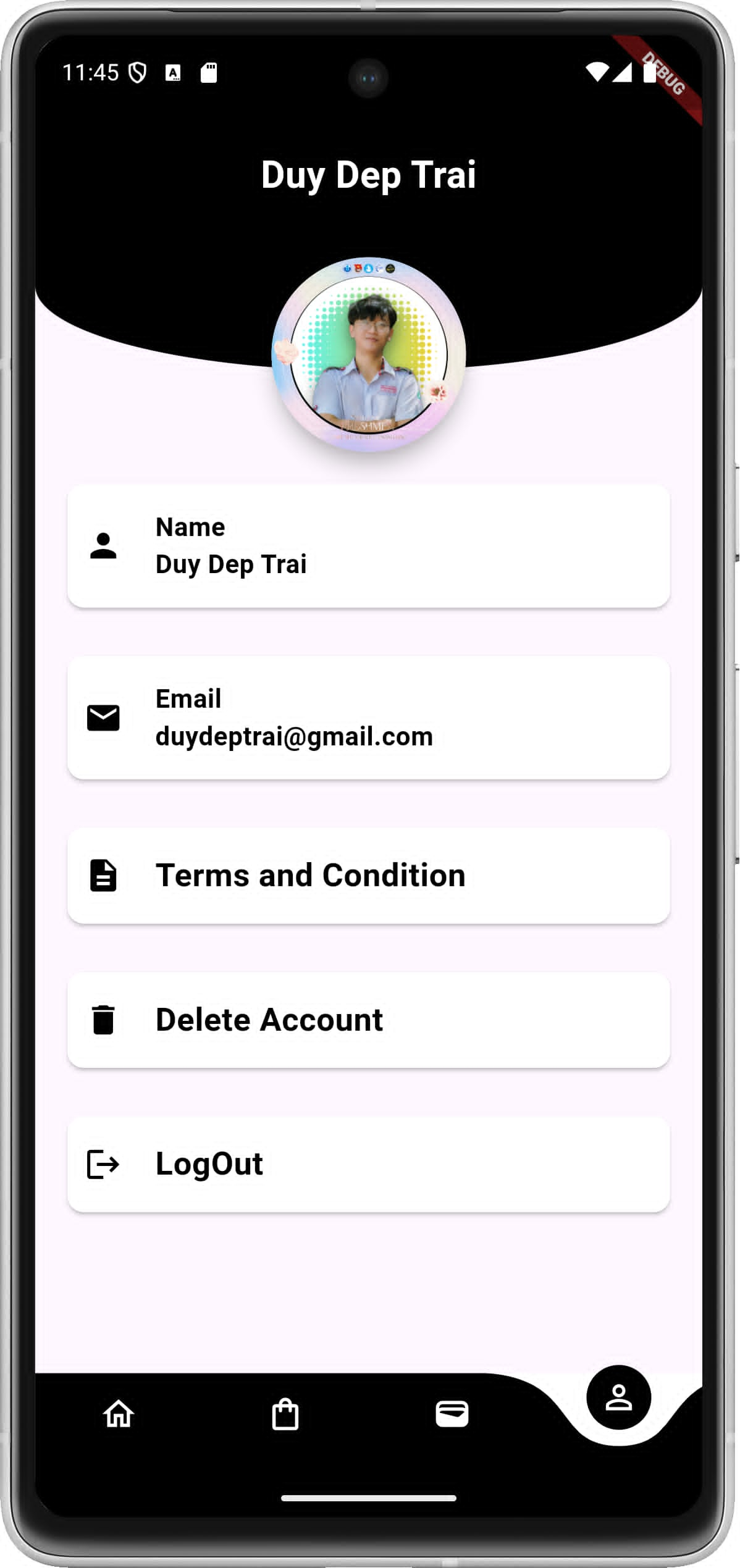
**6. Code Flow**

1. **User inputs Name, Email, and Password.**
2. **Input is validated using TextFormField validators.**
3. **On SIGN UP button tap:**
   * **registration() is triggered.**
   * **Firebase creates the account.**
   * **Firestore stores user details.**
   * **Shared Preferences cache the user data locally.**
4. **User is navigated to the BottomNav page on success.**
5. **Errors (e.g., weak passwords, duplicate accounts) are handled gracefully.**

****

**1. Admin Login Page with Firestore and Validation**





**1. Overview**

**This code represents an Admin Login page for a Flutter-based E-Commerce application. It uses Firestore to verify the admin credentials (id and password). If credentials are valid, the admin is redirected to the Admin Home Page. The page includes input validation, user feedback, and a structured design.**

**2. Functional Breakdown**

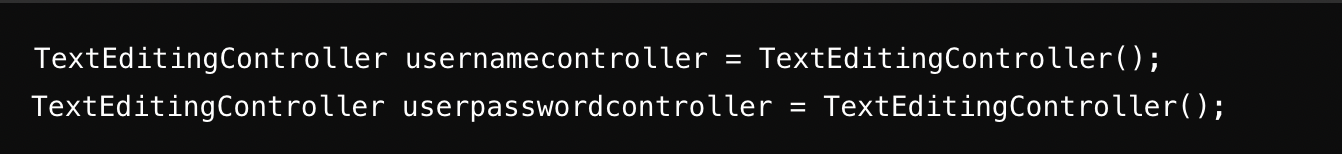
**2.1 Core Features**

1. **Admin Login:**
   * **Fetches all Admin records from Firestore.**
   * **Compares user input (id and password) with database values.**
2. **Input Validation:**
   * **Ensures the Username and Password fields are not empty.**
3. **Error Handling:**
   * **Provides feedback when:**
     + **Admin ID does not match.**
     + **Admin Password is incorrect.**
4. **Navigation:**
   * **On successful login, navigates to the HomeAdmin page using MaterialPageRoute.**
5. **UI Design:**
   * **Clean and modern design with gradient backgrounds and form inputs.**
   * **Uses containers and material widgets for organized layout.**

**3. Code Components**

**3.1 State Management**

**The code uses StatefulWidget (\_AdminLoginState) to manage dynamic user input:**

* **Controllers for TextFields:**
* **Ensures that the form is validated before proceeding with login.**

**3.2 Firestore Authentication**

**The LoginAdmin method fetches admin credentials from Firestore and verifies them:**

**A computer screen shot of white text

Description automatically generatedDSA Principle:**

* **Sequential Search: Iterates over all Firestore documents to compare credentials.**

**3.3 Input Validation**

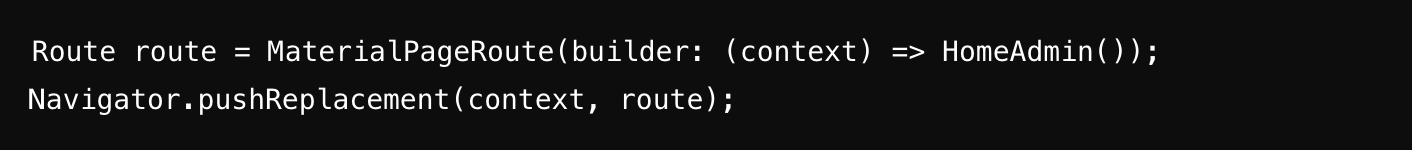
**Validators ensure that input fields are not empty:**

**A computer screen with white text

Description automatically generatedPrinciple: Prevents incomplete or invalid inputs.**

**3.4 Error Handling**

**Displays user-friendly messages using SnackBar:**

**Principle: Provides clear feedback for invalid credentials.**

**3.5 Navigation**

**On successful login, the admin is redirected to the HomeAdmin page:**

**dart**

**Copy code**

**Route route = MaterialPageRoute(builder: (context) => HomeAdmin());**

**Navigator.pushReplacement(context, route);**

**4. Code Flow**

1. **User Opens Admin Login Page:**
   * **Displays input fields for Username and Password.**
2. **Input Validation:**
   * **Ensures that both fields are filled.**
3. **Firestore Authentication:**
   * **Fetches data from the Admin collection in Firestore.**
   * **Validates:**
     + **Admin ID (id) matches user input.**
     + **Admin Password (password) matches user input.**
4. **Error Handling:**
   * **If ID or Password is incorrect, show a SnackBar with an error message.**
5. **Navigation:**
   * **On success: Navigate to the HomeAdmin page.**

**5. Key Code Lines Representing Principles**

| **Principle** | **Code Line(s)** | **Explanation** |
| --- | --- | --- |
| **Firestore Query** | **collection("Admin").get()** | **Fetches all admin credentials.** |
| **Sequential Search** | **snapshot.docs.forEach((result))** | **Iterates over fetched documents.** |
| **Input Validation** | **validator: (value)** | **Ensures fields are not empty.** |
| **Error Handling** | **ScaffoldMessenger.of(context).showSnackBar()** | **Displays feedback for errors.** |
| **Navigation** | **Navigator.pushReplacement(context, route)** | **Redirects to HomeAdmin on success.** |

**6. Code Workflow**

1. **Input Fields:**
   * **Username and Password inputs are displayed.**
2. **Validation:**
   * **Ensures both fields are filled.**
3. **Firestore Check:**
   * **Fetches all Admin documents.**
   * **Iterates over them to compare:**
     + **id field with usernamecontroller.text.trim().**
     + **password field with userpasswordcontroller.text.trim().**
4. **Error Handling:**
   * **If ID or Password mismatches, show appropriate messages.**
5. **Success:**
   * **Navigate to HomeAdmin.**

**7. Conclusion**

**This Admin Login Page efficiently handles:**

* **Firestore Integration for credential verification.**
* **Error Handling for invalid inputs or mismatched credentials.**
* **Input Validation for secure and complete data submission.**
* **UI Navigation for redirecting the admin to the appropriate page.**

**The implementation demonstrates sequential search for verifying credentials and adheres to clean coding principles for modularity, validation, and feedback.**

***\* HashMap for Data Lookup***

**HashMaps are used to store and retrieve user orders, authentication tokens, and product details quickly.**

* **Use Case:  
  HashMaps ensure efficient key-value pairing, e.g., storing user credentials and product details.**

**Code:**

**Map<String, Order> orderHistory = {};**

* **Benefit: HashMaps offer O(1) time complexity for searching and updating user data.**

***\* Sorting Algorithms***

**Sorting algorithms, such as QuickSort, are used to arrange products based on price, name, or category.**

* **Use Case:  
  Sorting the product list in ascending/descending order based on user preferences.**

**Code:**

**productList.sort((a, b) => a.price.compareTo(b.price));**

* **Benefit: QuickSort provides an average time complexity of O(n log n) for efficient sorting.**

***\* Search Algorithms***

**The Binary Search algorithm is used to search for products in sorted arrays.**

* **Use Case:  
  Searching for a product by name or ID in a sorted product list.**

**Code:**

**int binarySearch(List<Product> products, String target) {**

**int left = 0, right = products.length - 1;**

**while (left <= right) {**

**int mid = left + (right - left) ~/ 2;**

**if (products[mid].name == target) return mid;**

**if (products[mid].name.compareTo(target) < 0) left = mid + 1;**

**else right = mid - 1;**

**}**

**return -1;**

**}**

* **Benefit: Binary Search offers O(log n) time complexity for efficient searches.**

***\* Graphs for Navigation***

**The app uses graph-based navigation to manage user movement across multiple pages.**

* **Use Case:  
  Representing pages as nodes and links as edges to create seamless navigation.**
* **Implementation:  
  Flutter's Navigator manages page transitions.**

**Code:**

**Navigator.push(context, MaterialPageRoute(builder: (context) => ProductPage()));**

* **Benefit: Efficient navigation with a clear path between UI screens.**

***\* UI Design***

**Welcome Screen**

**A visually appealing welcome screen greets users:**

**Product Display and Management**

**Dynamic product images such as burgers, salads, and ice cream enhance the shopping experience:**

* **Burgers:**
* **Salads:**

**\* *Testing***

**Flutter testing ensures functionality across all platforms. Example:**

**Code:**

**testWidgets('Counter increments smoke test', (WidgetTester tester) async {**

**await tester.pumpWidget(const MyApp());**

**expect(find.text('0'), findsOneWidget);**

**await tester.tap(find.byIcon(Icons.add));**

**await tester.pump();**

**expect(find.text('1'), findsOneWidget);**

**});**

***IV. Key Takeaways***

**The E-Commerce Flutter Application successfully combines:**

* **Scalable Backend: Firebase for real-time operations.**
* **Secure Payments: Stripe for secure and efficient transactions.**
* **Engaging UI: High-quality images and smooth navigation.**
* **Cross-Platform Compatibility: A single codebase supports Android, iOS, macOS, and Web.**

***V. Conclusion***

**The E-Commerce Flutter Application is a comprehensive solution tailored for modern business needs. By leveraging Flutter’s cross-platform capabilities and integrating Firebase and Stripe, the app provides an efficient and scalable platform for e-commerce operations.**

**The project highlights the importance of:**

* **Collaborative Development: The use of modern tools like Firebase and Stripe streamlined backend and payment functionalities.**
* **Object-Oriented Principles: Ensuring clean, maintainable, and extensible code.**
* **User-Centric Design: Focusing on seamless navigation, intuitive UI, and visually appealing assets.**

**Future enhancements could include advanced features such as real-time order tracking, push notifications, and personalized product recommendations to further improve user engagement and experience.**