

**BANGLADESH UNIVERSITY OF BUSINESS AND TECHNOLOGY (BUBT)**

**REPORT On (“Recycle App**”)

**Course code: CSE 400**

**Course title: Software Development IV**

**Developed By:**

| Name | ID |
| --- | --- |
| Most. Sonia Islam | 21225103101 |
| Lamia Muntaha | 21225103185 |
| Md. Mehedi Hasan | 21225103334 |
| Abdullah Al Hill Baki Anim | 21225103341 |
| Humayra Kabir Hride | 21225103501 |
| Mohammad Shayukh Uddin Lohani | 21225103497 |

**Supervised by:**

Mastura Sadaf

Lecturer

Department of Computer Science and Engineering

Bangladesh University of Business and Technology (BUBT)

**29 June, 2025**

**DECLARATION**

We hereby declare that the project entitled “**Recycle App**” submitted for the completion of **CSE 400: Software Development IV** course in the faculty of Computer Science and Engineering of Bangladesh University of Business and Technology (BUBT), is our original work

**Most. Sonia Islam** **Lamia Muntaha Md. Mehedi Hasan**

**ID: 21225103101 ID: 21225103185 ID: 21225103334**

**Intake: 49 Intake: 49 Intake: 49** 

**Section-07 Section-07**  **Section-07**



**Abdullah Al Hill Baki Humayra Kabir Hride Mohammad Shayukh Uddin**

**Lohani**

**ID: 21225103341 ID: 21225103501 21225103947**

**Intake: 49 Intake: 49 Intake:49**

**Section-07 Section-07 Section-07**

# **ACKNOWLEDGEMENTS**

I would like to thank the following people for their help in the production of this project: **Mastura Sadaf** miss, ***Lecturer***, Dept. of **CSE**, BUBT, & our project supervisor for all of his ongoing assistance with the project, without whose help and support throughout, this project would not have been possible.We should thank our entire associate at **Bangladesh University Of** **Business And**  **Technology**, who took in this inspection while completing the class work. Finally, we ought to perceive with due respect the reliable assistance and patients of our parents.

# **ABSTRACT**

The “Recycle App”, an application, simplifies the process of recyclable waste management with a user-admin approval and rewards system. Users login using their Google credentials and can categorize recyclable items (plastic, paper, battery, glass), submit requests for pickups with image submissions, and enter relevant details involving address, quantity, and phone number. Once an admin approves the user's request, points are awarded for use through bKash when a redemption request is submitted. The admins login on a control panel and can view, approve, or reject pick up and redemptions requests as well as view their history of approval requests. The whole system is coded in using Flutter and Firebase, allowing for real-time data management and a responsive app. The Recycle App is intended to incite eco-friendly behavior by introducing a reward-driven approach and insert the ease of using an application, thus increasing ease of pick up requests for recyclable materials, while streamlining the workflows for users and administrators.

**List of Figures**

| Sl.No. | Figure Name | Page |
| --- | --- | --- |
| 1 | **Use Case Diagram** | **9** |
| 2 | **Context Level DFD** | **9** |
| 3 | **ER Diagram** | **10** |
| 4 | **Flowchart** | **10** |
| 5 | **Login using Google** | **11** |
| 6 | **Dashboard showing recycling categories** | **12** |
| 7 | **Image upload interface using device gallery** | **12** |
| 8 | **Points tracking and redeem option** | **13** |
| 9 | **Profile Page** | **13** |
| 10 | **Admin Manual login using ID and password** | **14** |
| 11 | **Admin Dashboard** | **14** |
| 12 | **View and approve/reject recycling requests** | **15** |
| 13 | **Approve redeem requests** | **15** |
| 14 | **View approve history** | **16** |
| 15 | **View approve history** | **16** |

**TABLE OF CONTENTS**

**DECLARATION****2**

[**ACKNOWLEDGEMENTS**](#_heading=) **3**

[**ABSTRACT**](#_heading=) **4**

**List of Figures****5**

[**CHAPTER 1: INTRODUCTION**](#_heading=) **7**

[1.1. Problem Specification](#_heading=h.636bxt2m3ump) 7

[1.2. Project Objectives](#_heading=h.m3an8fnbnqen) 7

[1.3 Flow of the Project](#_heading=h.yp1ikx4x5v05) 7

[1.4 Organization of Project Report](#_heading=h.4swnipkux4kf) 7

[**CHAPTER 2: BACKGROUND**](#_heading=) **8**

[2.1 Existing System Analysis](#_heading=h.4quictuzpcb9) 8

[2.2 Supporting Literatures](#_heading=h.2of2ew4u0s4m) 8

[**CHAPTER 3: SYSTEM ANALYSIS & DESIGN**](#_heading=) **9**

[3.1 Technology & Tools](#_heading=) 9

[3.2 Model & Diagram](#_heading=h.3g5ui7aifpgd) 9

3[.2.1. Model (AGILE)](#_heading=h.jife8g4mapxo) 9

3[.2.2. Use Case Diagram](#_heading=h.v7jaai6wr5i3) 9

3[.2.3. Context Level Diagram](#_heading=h.h488jn1p4sh5) 9

3[.2.4. Database Schema(ER Diagram)](#_heading=h.i7sppyv7r88e) 10

3.2.5. [Flowchart](#_heading=h.o8xjy9nej8rr) 10

[**CHAPTER 4: IMPLEMENTATION**](#_heading=) **11-32**

[4.1. Interface Design / Front-End](#_heading=h.aoef007c0cqk) 11-16

[4.2. Back-End](#_heading=h.bufck436etvn) 17-32

[4.3. Modules](#_heading=h.1n3c6st0axu5) 32-33

[**CHAPTER 5: USER MANUAL**](#_heading=) **34-35**

[5.1. System Requirements](#_heading=h.w0brtl6e7pzi) 34

[5.1.1. Hardware Requirements](#_heading=h.jife8g4mapxo) 34

[5.1.2. Software Requirements](#_heading=h.v7jaai6wr5i3) 34

[5.2. User Interfaces](#_heading=h.h488jn1p4sh5) 35

[5.2.1. Panel A – User Panel](#_heading=h.i7sppyv7r88e) 35

[5.2.2. Panel B – Admin Panel](#_heading=h.o8xjy9nej8rr) 35

[5.2.3. Login Credentials](#_heading=h.ir31viilzfge) 35

[**CHAPTER 6: CONCLUSION**](#_heading=) **36**

6[.1. Conclusion](https://docs.google.com/document/d/1tZcIHYtudLPMSaq6-aQECVLsQT9sXHUw/edit#heading=h.jife8g4mapxo) 36

6[.2. Limitation](https://docs.google.com/document/d/1tZcIHYtudLPMSaq6-aQECVLsQT9sXHUw/edit#heading=h.v7jaai6wr5i3) 36

6[.3. Future Works 3](https://docs.google.com/document/d/1tZcIHYtudLPMSaq6-aQECVLsQT9sXHUw/edit#heading=h.h488jn1p4sh5)6

[**References**](#_heading=h.3vua49s3asoh)**37**

# CHAPTER 1

# **INTRODUCTION**

### **1.1. Problem Specification**

The Recycle App is a digital platform designed to revolutionize the recyclable waste management procedure in Bangladesh. Conventional methods for dealing with recycled waste materials involve problems in inefficiency, problems in structure, and no useful benefits to motivate the public. In this way, the app has features which allow the user to classify which items should be recycled, submit a request for pickup that includes an image, deliver details, and earn points redeemable when administrative approval is received. It enables the user to log in using Google Authentication, makes tracking requests easy, gives access to redeem points and is environmentally friendly. The feature which allows administrative access to securely return to the app to approve a request and track point redemption, enhances the waste collection process. The Recycle App is built with Firebase, which facilitates real-time updates, provides responsiveness and delivers scalability.

### **1.2. Project Objectives**

* To develop a user-friendly application for users to request the pickup of recyclable items.
* To have users upload photos of items and fill out any necessary information for the pickup.
* To allow admins to verify requests and provide reward points for the requesting user.
* To make it possible for users to credit their points to collect their physical item.
* To engender awareness and awareness for a cleaner environment through digital ingenuity.
* Encourage sustainable waste management and environmental awareness.

### **1.3 Flow of the Project**

### **Requirement Analysis:** Identified user needs, features, and roles for both users and admins.

### **System Design:** Planned UI layout, data flow, and selected Firebase as core technologies.

### **Development:** Built modules like login, request submission, admin approval, and point redemption.

### **Testing:** Checked form validation, database syncing, and admin actions for errors.

### **Deployment:** Android application on an Android Phone.

### **1.4 Organization of Project Report**

This report is structured into six chapters:

* **Chapter 1** introduces the problem, objectives, scope, and report structure.
* **Chapter 2** reviews existing systems and related studies.
* **Chapter 3** details technologies, tools, diagrams, and system design models.
* **Chapter 4** explains the implementation process, front-end and back-end modules.
* **Chapter 5** presents the user manual, hardware/software requirements, and interface.
* **Chapter 6** provides the conclusion, limitations, and future scope.

CHAPTER 2

**BACKGROUND**

## **2.1 Existing System Analysis**

In the realm of digital scrap management, many recycling apps allow users to sort materials, but they often lack reward systems and proper admin controls. For instance, **Recycle Coach** gives recycling schedules and tips, but it does not offer user points or approvals for requests. **Scrapify** is an Android-based platform that connects scrap collectors and sellers but it lacks administrative control and flexibility. Similarly, **iRecycle** provides recycling information but lacks interactive request and redemption features. **Bota,** targets sustainable waste management by helping users classify waste into categories like recyclable, organic, and hazardous. However, it does not support monetary transactions, or rewards for users. These systems, while innovative, operate within narrow scopes.These shortcomings lower user motivation and hinder effective system management. The “**Recycle App**” addresses these issues. It enables users to choose categories, upload images, and submit detailed pickup requests, including their address and quantity. The app has a points system that requires admin approval, encouraging users to recycle more. Users can redeem their points through mobile payment, while admins efficiently manage approvals and redemption history. This approach creates a more engaging recycling solution.

## **2.2 Supporting Literatures**

The design and development of the **Recycle App** follow both technological best practices and academic research.It’s development is based on established research and modern development practices. Studies indicate that combining digital tools with waste management improves efficiency, increases user engagement, and raises environmental awareness. Reward-based systems, such as points and redemptions, effectively motivate behavior in sustainability-focused apps. The app uses Google Sign-In for secure user authentication, making it more accessible. Flutter allows for rapid cross-platform development with a single codebase, which saves time and money. Firebase provides real-time data updates, secure authentication, cloud functions, and analytics, which are crucial for responsive app behavior. The points system linked to admin approval ensures control and transparency, while payment integration allows for local mobile payments when redeeming points. These technologies support user-centered design principles and promote sustainable recycling through digital engagement.

CHAPTER 3

**SYSTEM ANALYSIS & DESIGN**

## **3.1 Technology & Tools**

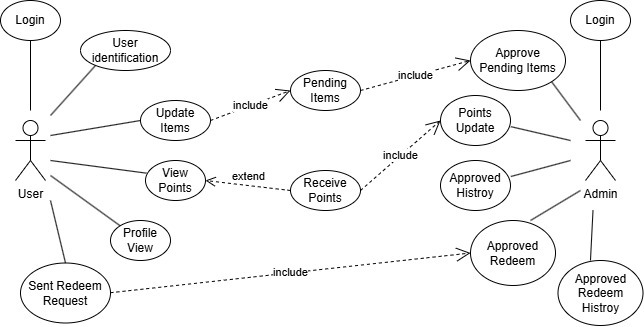
* **Flutter:** For cross-platform front-end development.
* **Firebase:** Backend services, including real-time database and user authentication.
* **Android Studio & VS Code:** Integrated development environments.

## **3.2 Model & Diagram**

**3.2.1. Model (Agile)**

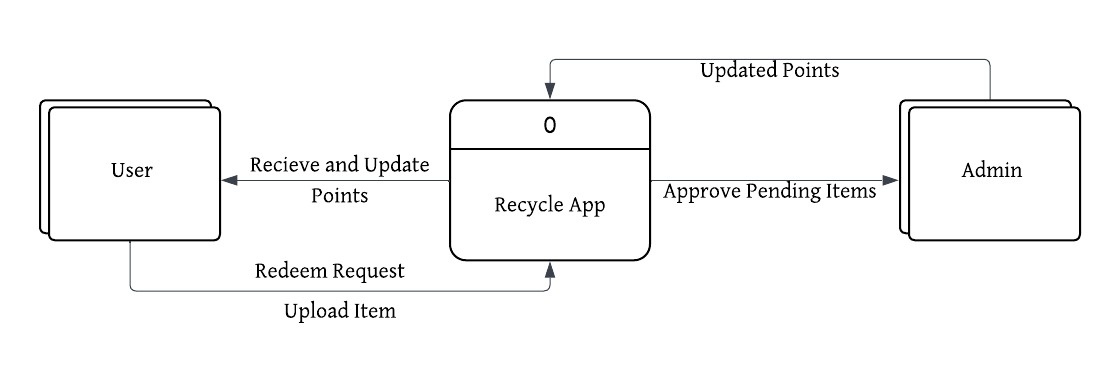
Agile methodology is used to ensure flexibility. It allows iterative releases and fast responses to user feedback and market demands.It is chosen because of its flexibility, relapse method and ability to include smooth feedback from the user for this project. Since the app requires frequent updates based on user interaction and development market needs, Agile ensures rapid growth cycle, risk reduction and compatibility to change requirements. The model allows step -by -step improvement and causes more sophisticated and user -friendly applications over time

**3.2.2. Use Case Diagram**

****

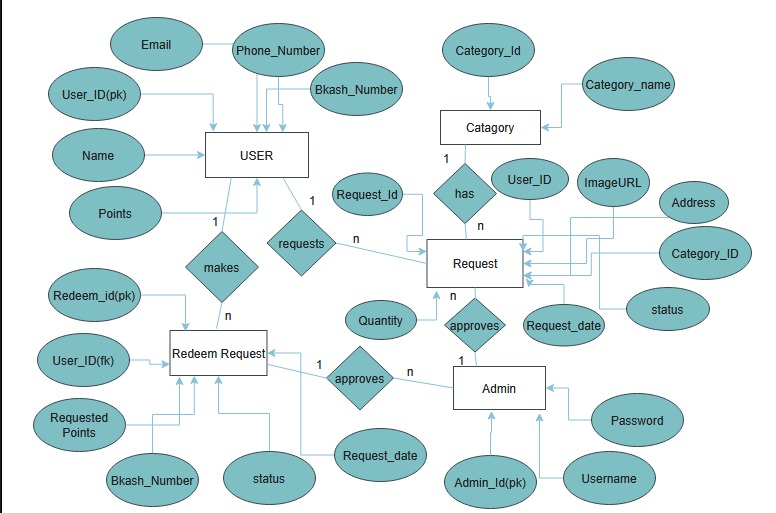
**Figure 1: Use Case Diagram**

**3.2.3. Context Level Diagram**

****

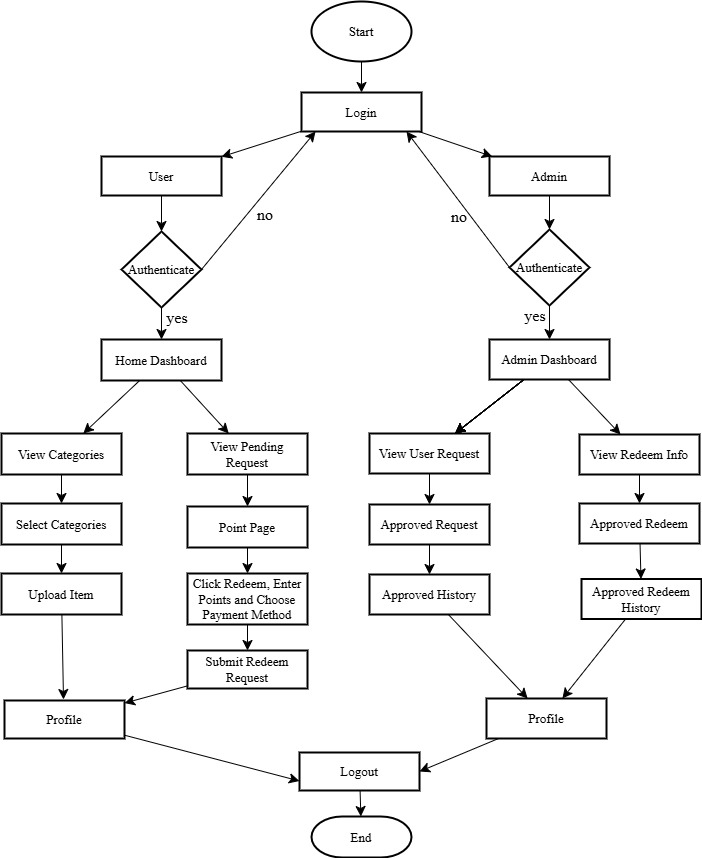
**Figure 2: Context Level DFD**

**3.2.4. ER Diagram**

****

**Figure 3: ER Diagram**

**3.2.5. Flowchart**

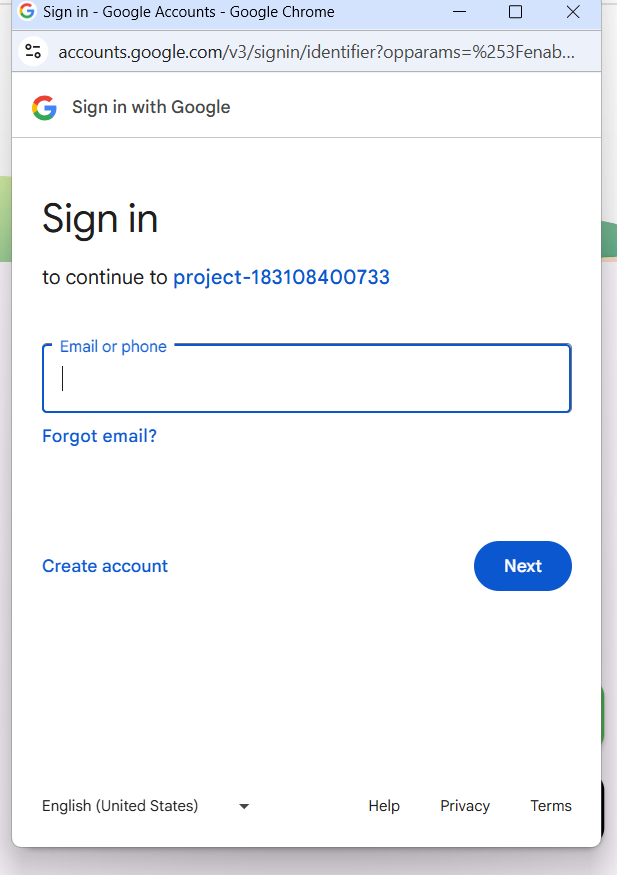
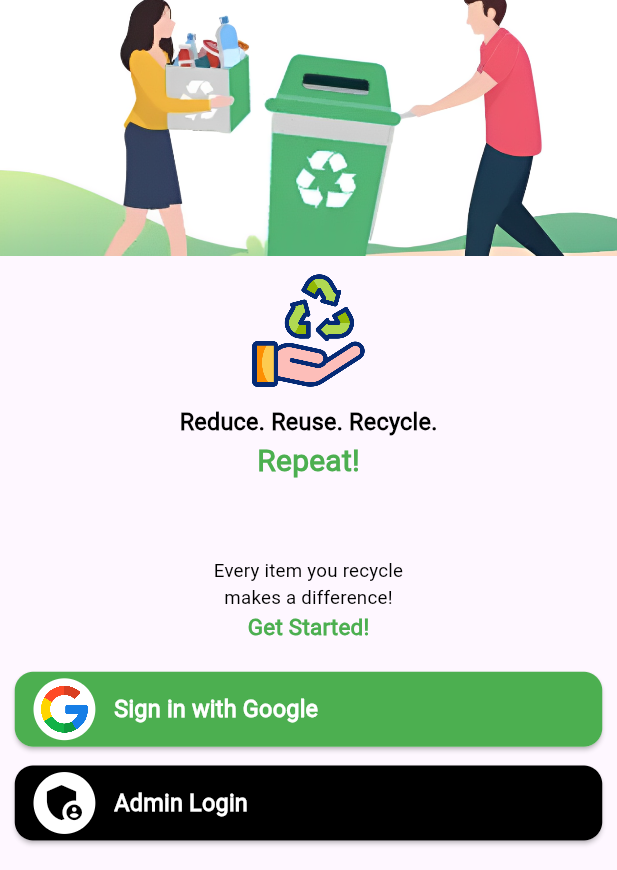
****

**Figure 4: Flowchart**

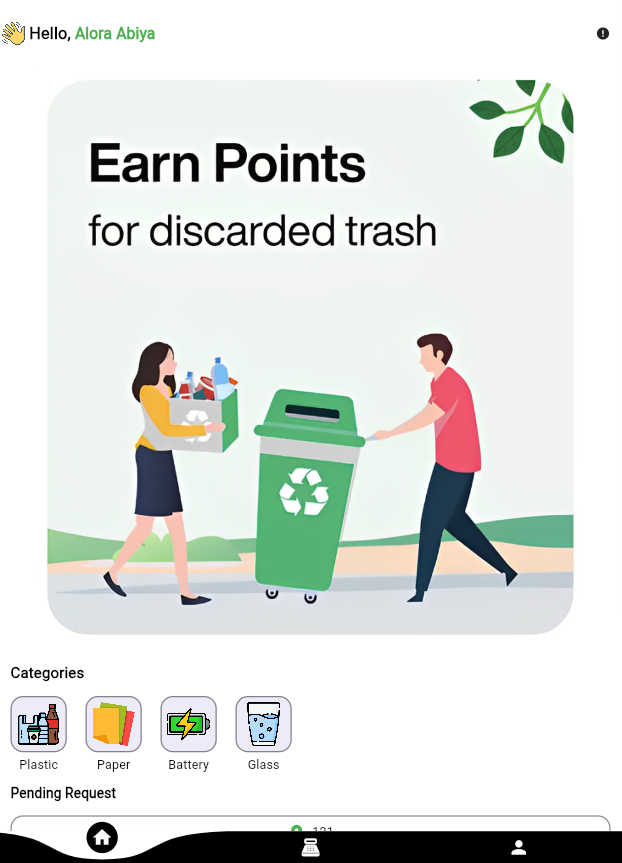
**CHAPTER 4**

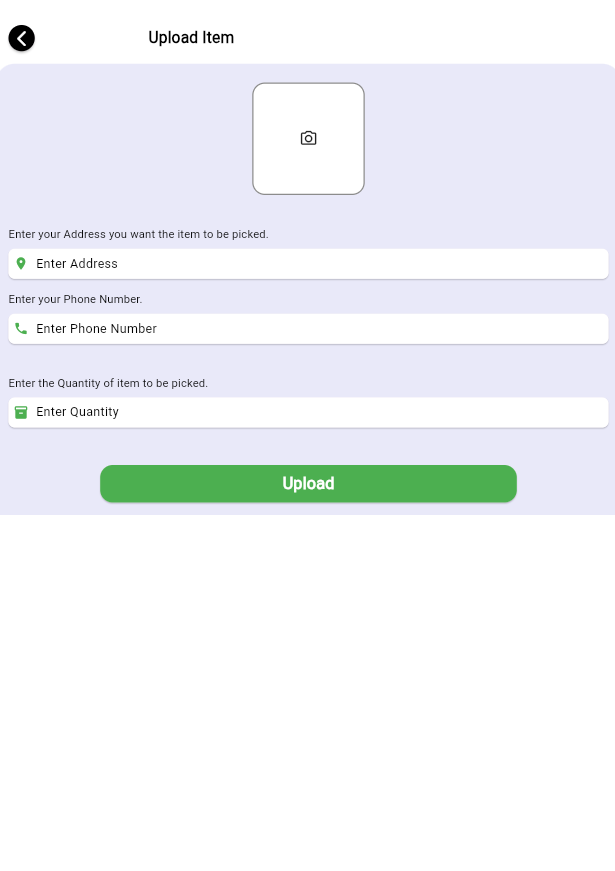
**IMPLEMENTATION**

### **4.1. Interface Design / Front-End**

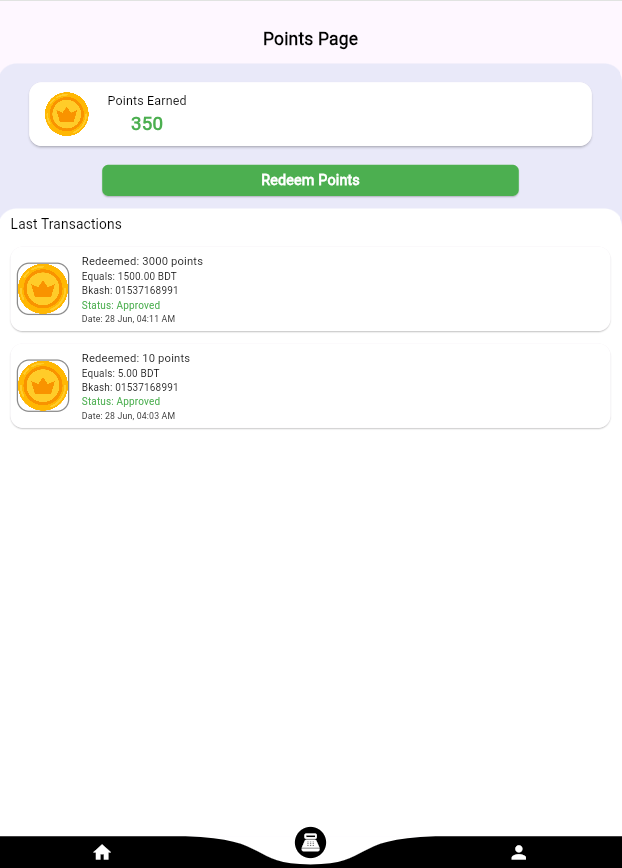
**The front-end of the Recycle App is built using Flutter  
**

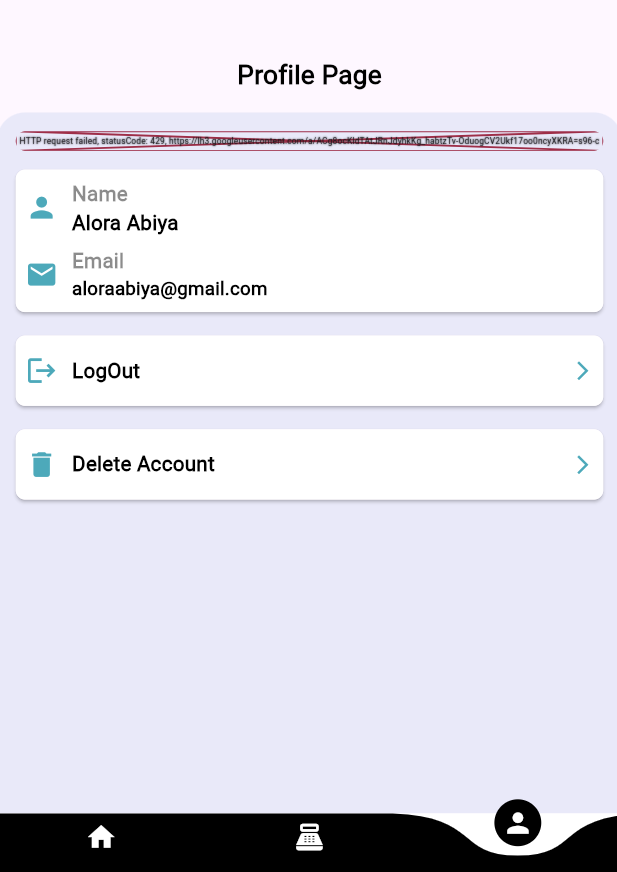
**Figure 5: Login using Google**

****

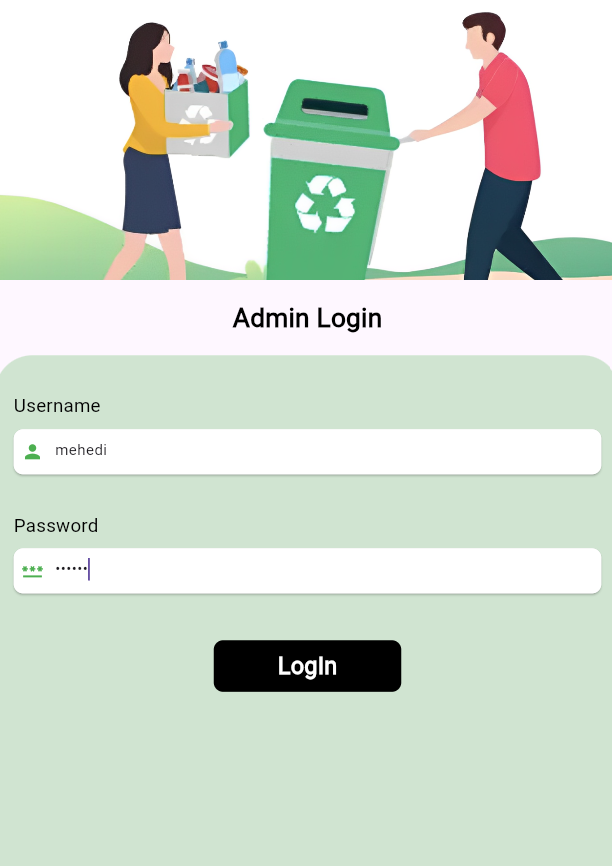
**Figure 6: Dashboard showing recycling categories  
**

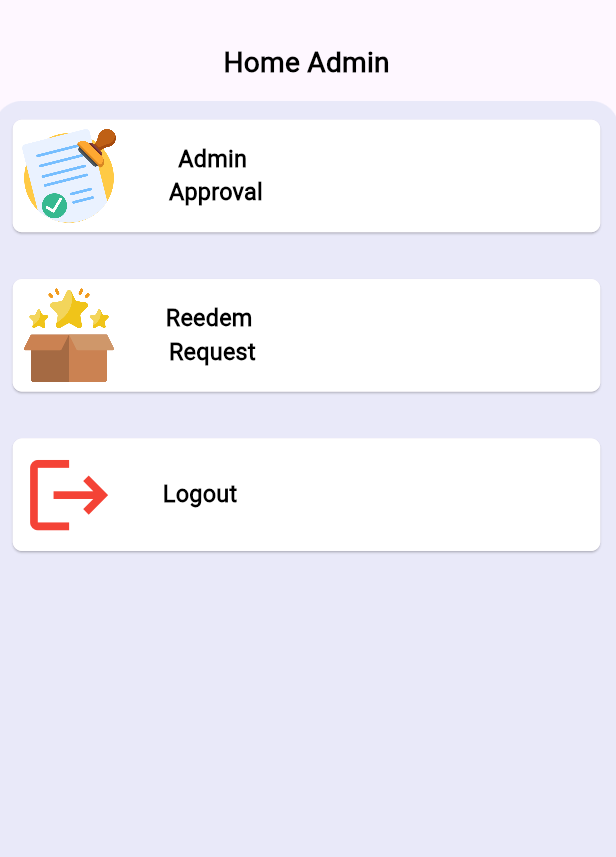
**Figure 7: Image upload interface using device gallery**

****

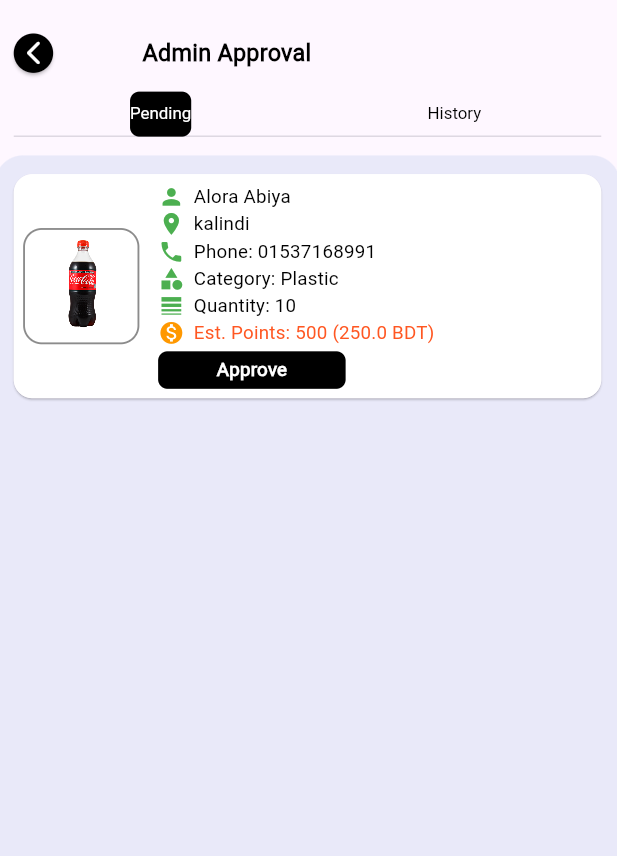
**Figure 8: Points tracking and redeem option  
**

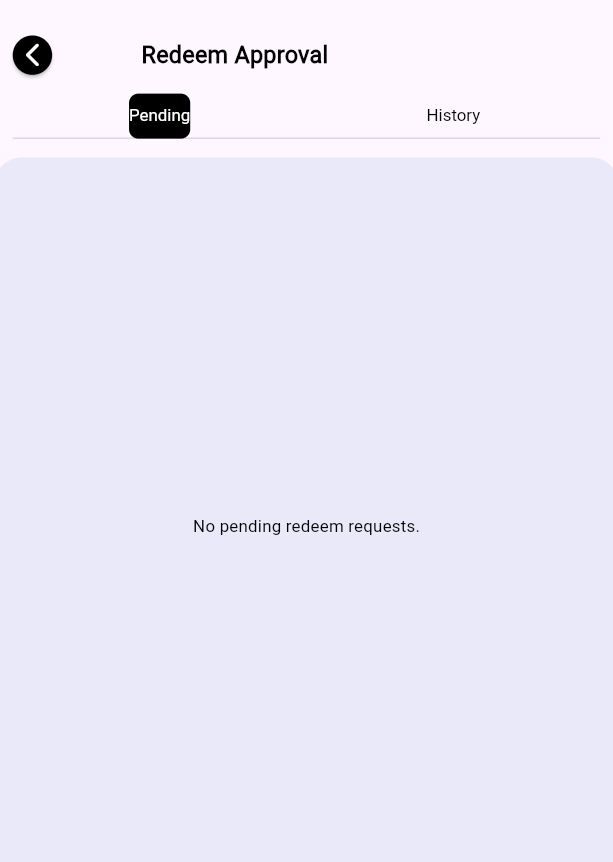
**Figure 9: Profile Page**

****

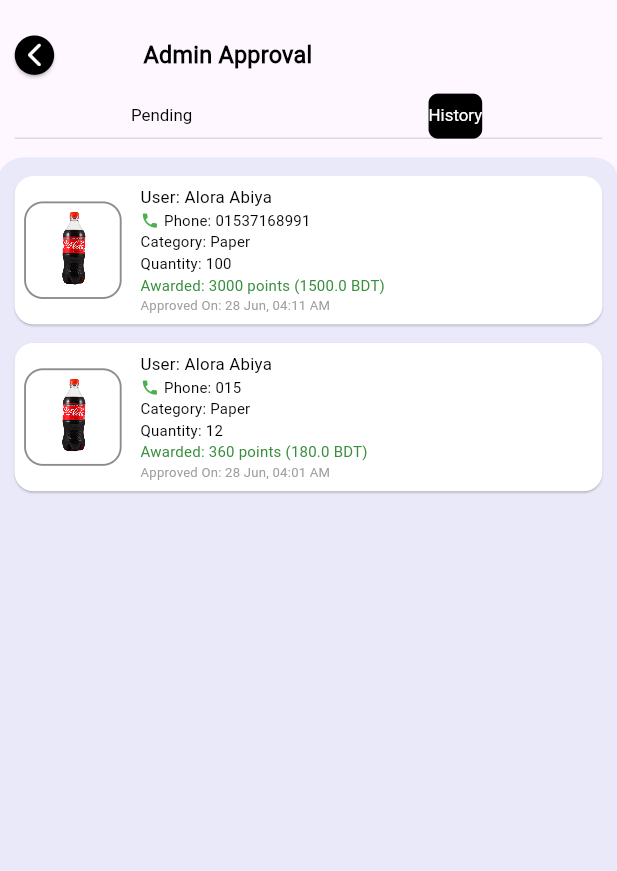
**Figure 10: Admin Manual login using ID and password  
**

**Figure 11: Admin Dashboard**

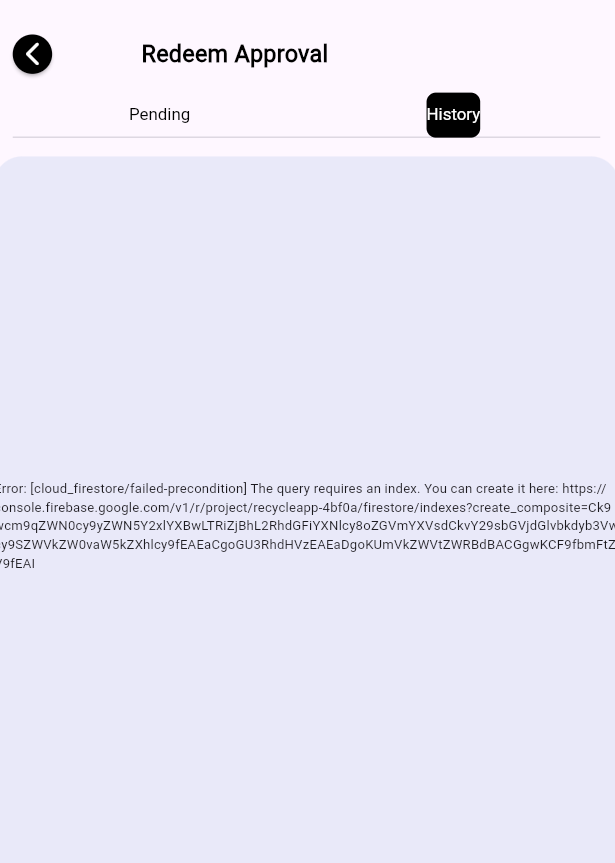
****

**Figure 12: View and approve/reject recycling requests  
**

**Figure 13: Approve redeem requests**

****

**Figure 14: View approve history**

****

**Figure 15: View approve history**

### **4.2. Back-End**

### **Snippet 1: Initializing Google Sign-In and Creating Firebase Credential**

| // Code Snippet X.1: Initiating Google Sign-In and Firebase Credential Creation  class AuthMethods {  signInWithGoogle(BuildContext context) async {  final FirebaseAuth firebaseAuth = FirebaseAuth.instance;  final GoogleSignIn googleSignIn = GoogleSignIn(  clientId: 'YOUR\_CLIENT\_ID', // Replace with a placeholder or mention it's from Google Cloud  scopes: [  'email',  'profile',  'openid',  ],  );  try {  final GoogleSignInAccount? googleSignInAccount = await googleSignIn.signIn();  if (googleSignInAccount == null) {  // Handle cancellation  return;  }  final GoogleSignInAuthentication? googleSignInAuthentication =  await googleSignInAccount.authentication; | if (googleSignInAuthentication == null || googleSignInAuthentication.accessToken == null) {  // Handle missing tokens  return;  }  final AuthCredential credential = GoogleAuthProvider.credential(  idToken: googleSignInAuthentication.idToken,  accessToken: googleSignInAuthentication.accessToken,  );  UserCredential result = await firebaseAuth.signInWithCredential(credential);  User? userDetails = result.user;  if (userDetails == null) {  // Handle user details not found  return;  }  // ... (Further logic for saving user details and navigating)  } catch (e) {  // Error handling  }  }  // ... other methods  } |
| --- | --- |

**Snippet 2: Handling New vs. Existing User Data in Firestore**

| // Code Snippet X.2: Firestore Logic for New vs. Existing User Profiles  // (Continuing from the signInWithGoogle method)  // ...  User? userDetails = result.user; // User object from Firebase Auth  if (userDetails == null) { /\* handle null \*/ return; }  // Save user details to SharedPreferences (important for client-side state)  await SharedpreferenceHelper().saveUserEmail(userDetails.email!);  await SharedpreferenceHelper().saveUserId(userDetails.uid);  await SharedpreferenceHelper().saveUserImage(userDetails.photoURL!);  await SharedpreferenceHelper().saveUserName(userDetails.displayName!);  // Check if user document already exists in Firestore  // DatabaseMethods().getUserDocument fetches the DocumentSnapshot for the user's UID  DocumentSnapshot userDoc = await DatabaseMethods().getUserDocument(userDetails.uid);  if (!userDoc.exists) {  // User is signing in for the first time or document was deleted | Map<String, dynamic> newUserInfoMap = {  "email": userDetails.email,  "name": userDetails.displayName,  "image": userDetails.photoURL,  "Id": userDetails.uid,  "Points": "0", // Initialize points to "0" only for new users  };  // DatabaseMethods().addUserInfo adds the new document  await DatabaseMethods().addUserInfo(newUserInfoMap, userDetails.uid);  } else {  // User document exists, update other profile info but preserve Points  Map<String, dynamic> updateInfoMap = {  "email": userDetails.email,  "name": userDetails.displayName,  "image": userDetails.photoURL,  "Id": userDetails.uid,  };  // Use .update() to modify specific fields without overwriting the whole document  await FirebaseFirestore.instance  .collection("users")  .doc(userDetails.uid)  .update(updateInfoMap);  }  // Navigate to main app  Navigator.pushReplacement(  context,  MaterialPageRoute(builder: (context) => const BottomNav()),  );  // ... (End of signInWithGoogle method) |
| --- | --- |

**Snippet 3: User Sign Out Functionality**

| // Code Snippet X.3: User Sign Out Functionality  class AuthMethods {  // ... signInWithGoogle method ...  Future SignOut() async {  // Sign out from Firebase Authentication  await FirebaseAuth.instance.signOut();  // Sign out from Google Sign-In (important for Google-linked accounts)  await GoogleSignIn().signOut();  // Clear user data from SharedPreferences for local cache invalidation  await SharedpreferenceHelper().saveUserId('');  await SharedpreferenceHelper().saveUserName('');  await SharedpreferenceHelper().saveUserEmail('');  await SharedpreferenceHelper().saveUserImage('');  print('DEBUG: User signed out successfully and SharedPreferences cleared.');  }  // ... deleteuser method ...  } |
| --- |

**Snippet 4: Fetching User Data and Pending Requests**

| import 'package:cloud\_firestore/cloud\_firestore.dart';  import 'package:flutter/material.dart';  import 'package:recycleapp/services/database.dart';  import 'package:recycleapp/services/shared\_pref.dart';  class Home extends StatefulWidget {  const Home({super.key});  @override  State<Home> createState() => \_HomeState();  }  class \_HomeState extends State<Home> {  String? id, name, userImage;  Stream? pendingStream;  @override  void initState() {  ontheload();  super.initState();  }  ontheload() async {  await getthesharedpref();  if (id != null) {  pendingStream = await DatabaseMethods().getUserPendingRequests(id!);  }  setState(() {});  }  getthesharedpref() async {  id = await SharedpreferenceHelper().getUserId();  name = await SharedpreferenceHelper().getUserName();  userImage = await SharedpreferenceHelper().getUserImage();  setState(() {});  } | Widget allPendingRequests() {  return StreamBuilder(  stream: pendingStream,  builder: (context, AsyncSnapshot snapshot) {  return snapshot.hasData  ? ListView.builder(  padding: EdgeInsets.zero,  itemCount: snapshot.data.docs.length,  itemBuilder: (context, index) {  DocumentSnapshot ds = snapshot.data.docs[index];  // UI code to display data...  return Container(  margin: const EdgeInsets.only(  left: 20.0,  right: 20.0,  bottom: 20.0,  ),  // ... (rest of the UI for each pending request item)  child: Column(  children: [  // ... UI elements using ds["Address"], ds["Quantity"] ...  ],  ),  );  },  )  : Container();  },  );  }  // ... rest of the Home widget build method ...  } |
| --- | --- |

**Snippet 5: UploadItem Widget**

| import 'dart:io' show File;  import 'package:flutter/material.dart';  import 'package:image\_picker/image\_picker.h,';  import 'package:random\_string/random\_string.h,';  import 'package:recycleapp/services/database.dart'; // \*\*Backend Interaction\*\*  import 'package:recycleapp/services/shared\_pref.dart'; // \*\*Backend Interaction\*\*  import 'package:flutter/foundation.h,'  show kIsWeb;  import 'dart:typed\_data';  class UploadItem extends StatefulWidget {  final String category;  final String id;  UploadItem({super.key, required this.category, required this.id});  @override  State<UploadItem> createState() => \_UploadItemState();  }  class \_UploadItemState extends State<UploadItem> {  TextEditingController addresscontroller = TextEditingController();  TextEditingController quantitycontroller = TextEditingController();  TextEditingController phonenumbercontroller = TextEditingController();  final ImagePicker \_picker = ImagePicker();  XFile? \_pickedXFile;  Uint8List? \_webImageBytes;  String? userId, userName;  // Function to get user id and name from shared preferences  getTheSharedPref() async {  // \*\*Backend Interaction: Reading from Shared Preferences\*\*  userId = await SharedpreferenceHelper().getUserId();  userName = await SharedpreferenceHelper().getUserName();  setState(() {});  }  @override  void initState() {  super.initState();  getTheSharedPref();  }  Future<void> getImage() async {  try {  final XFile? image = await \_picker.pickImage(source: ImageSource.gallery);  if (image != null) {  setState(() {  \_pickedXFile = image;  });  if (kIsWeb) {  \_webImageBytes = await image.readAsBytes();  }  }  } catch (e) {  print('Error picking image: $e');  ScaffoldMessenger.of(context).showSnackBar(SnackBar(content: Text('Failed to pick image: $e')));  }  }  @override  Widget build(BuildContext context) {  return Scaffold(  backgroundColor: Colors.white,  body: SingleChildScrollView(  child: Container(  margin: const EdgeInsets.only(top: 40.0),  child: Column(  children: [  // ... (UI elements like back button, title, image picker, text fields) ...  GestureDetector(  onTap: () async {  if (addresscontroller.text.isNotEmpty &&  quantitycontroller.text.isNotEmpty &&  phonenumbercontroller.text.isNotEmpty &&  \_pickedXFile != null) {  String itemId = randomAlphaNumeric(10); | // \*\*Backend Interaction: Placeholder for Firebase Storage Upload\*\*  // This section is commented out but indicates where image  // upload to Firebase Storage would occur.  // For web: firebaseStorageRef.putData(\_webImageBytes!)  // For mobile: firebaseStorageRef.putFile(File(\_pickedXFile!.path))  Map<String, dynamic> addItem = {  "Image": "", // This would be the download URL from Firebase Storage  "Address": addresscontroller.text,  "PhoneNumber": phonenumbercontroller.text,  "Quantity": quantitycontroller.text,  "UserId": userId,  "Name": userName,  "Status": "Pending",  "Category": widget.category,  };  // \*\*Backend Interaction: Adding data to Firestore (User's Collection)\*\*  await DatabaseMethods().addUserUploadItem(  addItem,  userId!,  itemId,  );  // \*\*Backend Interaction: Adding data to Firestore (Admin's Collection)\*\*  await DatabaseMethods().addAdminItem(  addItem,  itemId,  );  ScaffoldMessenger.of(context).showSnackBar(  SnackBar(  backgroundColor: Colors.green,  content: Text(  "Item has been uploaded Successfully!",  style: AppWidget.whitetextstyle(22.0),  ),  ),  );  setState(() {  addresscontroller.text = "";  quantitycontroller.text = "";  phonenumbercontroller.text = "";  \_pickedXFile = null;  \_webImageBytes = null;  });  } else {  ScaffoldMessenger.of(context).showSnackBar(  const SnackBar(  backgroundColor: Colors.red,  content: Text(  "Please fill all fields and select an image.",  style: TextStyle(color: Colors.white),  ),  ),  );  }  },  child: Center(  // ... (Upload button UI) ...  ),  ),  ],  ),  ),  ),  );  }  } |
| --- | --- |

**Snippet 6: Point**

| import 'package:cloud\_firestore/cloud\_firestore.dart';  import 'package:random\_string/random\_string.dart';  import 'package:intl/intl.dart';  // Assuming you have these service classes defined elsewhere  import 'package:recycleapp/services/database.dart';  import 'package:recycleapp/services/shared\_pref.dart';  class PointsService { // Renamed from BackendSnippet for better clarity  String? id; // User ID from shared preferences  String? mypoints; // User points, WILL BE UPDATED BY the StreamBuilder  String? name; // User name from shared preferences  Stream? userRedeemTransactionsStream;  // Define the conversion rate  static const double pointsToBdtRate = 0.5; // 1 point = 0.5 BDT  PointsService() {  \_loadUserDataAndStreams();  }  Future<void> \_loadUserDataAndStreams() async {  print("Points Page: Loading user data from shared preferences...");  id = await SharedpreferenceHelper().getUserId();  name = await SharedpreferenceHelper().getUserName();  print("Points Page: Loaded userId: $id, userName: $name");  if (id != null) {  userRedeemTransactionsStream = await DatabaseMethods().getUserTransactions(id!);  print("Points Page: Initialized userRedeemTransactionsStream.");  } else {  print("Points Page: User ID is null, cannot initialize userRedeemTransactionsStream.");  }  }  // This method provides the stream for user redeem transactions  Stream? getUserRedeemTransactionsStream() {  return userRedeemTransactionsStream;  }  // This method handles the redemption logic  Future<String> redeemPoints(String pointsToRedeemStr, String bkashNumber) async {  print("Points Page: Redeem button tapped. Points: $pointsToRedeemStr, Bkash: $bkashNumber");  if (pointsToRedeemStr.isEmpty || bkashNumber.isEmpty) {  return 'Please fill all fields.';  }  int pointsToRedeem = int.tryParse(pointsToRedeemStr) ?? 0;  int currentMyPoints = int.tryParse(mypoints ?? '0') ?? 0;  print("Points Page: Attempting to redeem $pointsToRedeem points. Current points: $currentMyPoints"); | if (pointsToRedeem <= 0) {  return 'Please enter a positive amount of points to redeem.';  }  if (currentMyPoints < pointsToRedeem) {  return 'Insufficient points to redeem.';  }  DateTime now = DateTime.now();  String formattedDate = DateFormat('d MMM, hh:mm a').format(now);  // Calculate the equivalent BDT amount for the redemption request  double bdtEquivalent = pointsToRedeem \* pointsToBdtRate;  Map<String, dynamic> redeemRequestMap = {  "Name": name,  "Points": pointsToRedeemStr,  "BkashNumber": bkashNumber,  "Status": "Pending", // Set status to Pending  "Date": formattedDate,  "UserId": id,  "BDT\_Equivalent": bdtEquivalent.toStringAsFixed(2), // Add BDT equivalent to the request  };  String redeemid = randomAlphaNumeric(10);  // Assuming DatabaseMethods is a class that interacts with Firestore  await DatabaseMethods().addUserReedemPoints(  redeemRequestMap,  id!,  redeemid,  );  print("Points Page: Added user redeem history.");  await DatabaseMethods().addAdminReedemRequests(  redeemRequestMap,  redeemid,  );  print("Points Page: Added admin redeem request.");  return 'Redemption request submitted successfully! Waiting for admin approval.';  }  // Method to update mypoints when the StreamBuilder in Flutter provides new data  void updateMyPoints(String newPoints) {  mypoints = newPoints;  print("PointsService: mypoints updated to $mypoints");  }  // Method to get the current points for display or validation  String getCurrentMyPoints() {  return mypoints ?? '0';  }  } |
| --- | --- |

**Snippet 7: Profile Service Backend**

| import 'package:flutter/material.dart'; // Keep if AuthMethods().deleteuser() uses BuildContext  import 'package:recycleapp/services/auth.dart';  import 'package:recycleapp/services/shared\_pref.dart';  class ProfileService {  String? id;  String? name;  String? email;  String? image;  /// Loads user data (ID, name, email, image) from shared preferences.  ///  /// This method retrieves the stored user information asynchronously  /// and populates the corresponding properties within this service.  Future<void> loadUserDataFromSharedPrefs() async {  id = await SharedpreferenceHelper().getUserId();  name = await SharedpreferenceHelper().getUserName();  email = await SharedpreferenceHelper().getUserEmail();  image = await SharedpreferenceHelper().getUserImage();  } | /// Signs out the current user.  ///  /// This method calls the `SignOut` function from `AuthMethods`  /// to log the user out of the application.  Future<void> signOutUser() async {  await AuthMethods().SignOut();  }  /// Deletes the current user's account.  ///  /// This method calls the `deleteuser` function from `AuthMethods`,  /// which typically handles the account deletion process, potentially  /// requiring a `BuildContext` for UI interactions like showing dialogs.  /// It's crucial to handle re-authentication before calling this in a real app.  Future<void> deleteUserAccount(BuildContext context) async {  await AuthMethods().deleteuser(context);  }  // Optional: Getters to safely access the loaded data from outside the service.  String? get currentUserId => id;  String? get currentUserName => name;  String? get currentUserEmail => email;  String? get currentUserImage => image;  } |
| --- | --- |

**Snippet 8: Admin login**

| import 'package:cloud\_firestore/cloud\_firestore.dart';  import 'package:flutter/material.dart'; // Keep BuildContext for ScaffoldMessenger  // This class encapsulates the backend logic for Admin Login  class AdminLoginService {  /// Authenticates an admin user against a Firestore collection.  ///  /// This method retrieves all documents from the "Admin" collection  /// in Firestore. It then iterates through the documents to check  /// if the provided `username` and `password` match any admin record.  ///  /// \*\*Important Security Note\*\*: This approach of fetching all admin  /// credentials to the client and then comparing them is HIGHLY INSECURE.  /// In a production application, admin authentication should ALWAYS  /// be handled server-side (e.g., using Firebase Authentication with custom claims,  /// Firebase Admin SDK, or a secure backend API).  ///  /// For demonstration purposes within this snippet, it mimics the provided logic.  ///  /// [username] The username entered by the admin.  /// [password] The password entered by the admin.  /// [context] The BuildContext to show UI feedback (SnackBar, navigation).  Future<void> loginAdmin(String username, String password, BuildContext context) async {  try {  // Fetch all documents from the "Admin" collection  QuerySnapshot adminSnapshot = await FirebaseFirestore.instance.collection("Admin").get(); | if (isAuthenticated) {  // Navigate to admin home on successful login  // Assuming HomeAdmin is defined and imported in the calling UI file  // Navigator.pushReplacement(context, MaterialPageRoute(builder: (context) => HomeAdmin()));  ScaffoldMessenger.of(context).showSnackBar(  SnackBar(  backgroundColor: Colors.green,  content: Text(  "Admin Login Successful!",  style: TextStyle(fontSize: 18.0, color: Colors.white, fontWeight: FontWeight.bold),  ),  ),  );  print("Admin login successful for user: $username");  } else {  ScaffoldMessenger.of(context).showSnackBar(  SnackBar(  backgroundColor: Colors.red,  content: Text(  "Invalid Username or Password",  style: TextStyle(fontSize: 18.0, color: Colors.white, fontWeight: FontWeight.bold),  ),  ),  );  print("Admin login failed for user: $username - Invalid credentials.");  }  } catch (e) {  ScaffoldMessenger.of(context).showSnackBar(  SnackBar(  backgroundColor: Colors.red,  content: Text(  "An error occurred: ${e.toString()}",  style: TextStyle(fontSize: 18.0, color: Colors.white, fontWeight: FontWeight.bold),  ),  ),  );  print("Error during admin login: $e");  }  }  } |
| --- | --- |

**Snippet 9:admin dashboard**

| import 'package:firebase\_auth/firebase\_auth.dart';  import 'package:flutter/material.dart'; // Required for BuildContext, SnackBar  class AdminHomeBackend {  Future<void> logout(BuildContext context) async {  try {  await FirebaseAuth.instance.signOut();  // Navigation is a UI concern, but included here as it's directly tied  // to the success of the backend operation.  Navigator.pushAndRemoveUntil(  context,  MaterialPageRoute(builder: (context) => | LogIn()), // Assuming LogIn is your login page  (Route<dynamic> route) => false,  );  } catch (e) {  // Error handling for logout failure  print("Error during logout: $e");  ScaffoldMessenger.of(context).showSnackBar(  SnackBar(content: Text("Failed to logout. Please try again.")),  );  }  }  } |
| --- | --- |

**Snippet 10:admin approval**

| import 'package:cloud\_firestore/cloud\_firestore.dart';  class DatabaseMethods {  // Collection reference for user-submitted item requests  final CollectionReference \_adminApprovalCollection =  FirebaseFirestore.instance.collection('AdminApproval');  // Collection reference for user documents  final CollectionReference \_usersCollection =  FirebaseFirestore.instance.collection('users');  /// Fetches a stream of pending item requests for admin approval.  ///  /// This retrieves documents from the 'AdminApproval' collection  /// where the 'Status' field is 'Pending'.  Stream<QuerySnapshot> getAdminApproval() {  return \_adminApprovalCollection.where('Status', isEqualTo: 'Pending').snapshots();  }  /// Fetches a stream of approved item requests for admin history.  ///  /// This retrieves documents from the 'AdminApproval' collection  /// where the 'Status' field is 'Approved', ordered by 'ApprovedAt'  /// in descending order to show most recent approvals first.  Stream<QuerySnapshot> getApprovedItemsHistory() {  return \_adminApprovalCollection  .where('Status', isEqualTo: 'Approved')  .orderBy('ApprovedAt', descending: true)  .snapshots();  }  /// Retrieves a specific user's document from the 'users' collection.  ///  /// [userId] The ID of the user document to retrieve.  Future<DocumentSnapshot> getUserDocument(String userId) { | return \_usersCollection.doc(userId).get();  }  /// Updates the 'Points' field for a specific user.  ///  /// [userId] The ID of the user whose points are to be updated.  /// [newPoints] The new total points value as a string.  Future<void> updateUserPoints(String userId, String newPoints) async {  return await \_usersCollection.doc(userId).update({'Points': newPoints});  }  /// Updates the status of an item approval request in the 'AdminApproval' collection.  ///  /// Sets the 'Status' to 'Approved' and records the 'ApprovedAt' timestamp.  /// [requestId] The ID of the request document to update.  Future<void> updateAdminRequest(String requestId) async {  return await \_adminApprovalCollection.doc(requestId).update({  'Status': 'Approved',  'ApprovedAt': FieldValue.serverTimestamp(),  });  }  /// Updates the status of a user's specific item request within their 'Items' subcollection.  ///  /// Sets the 'Status' to 'Approved' for the given [requestId] under the specified [userId].  /// This assumes a subcollection 'Items' exists under each user document.  /// [userId] The ID of the user.  /// [requestId] The ID of the specific item request within the user's 'Items' subcollection.  Future<void> updateUserRequest(String userId, String requestId) async {  return await \_usersCollection  .doc(userId)  .collection('Items')  .doc(requestId)  .update({'Status': 'Approved'});  }  } |
| --- | --- |

**Snippet 11: redeem requests**

| import 'package:cloud\_firestore/cloud\_firestore.dart';  import 'package:flutter/material.dart';  import 'package:intl/intl.dart';  import 'package:recycleapp/services/database.dart';  import 'package:recycleapp/services/widget\_support.dart';  class AdminReedem extends StatefulWidget {  const AdminReedem({super.key});  @override  State<AdminReedem> createState() => \_AdminReedemState();  }  class \_AdminReedemState extends State<AdminReedem> with SingleTickerProviderStateMixin {  Stream? redeemStream;  Stream? redeemedHistoryStream;  late TabController \_tabController;  static const double pointsToBdtRate = 0.5;  @override  void initState() {  super.initState();  \_tabController = TabController(  length: 2,  vsync: this,  );  getontheload();  }  @override  void dispose() {  \_tabController.dispose();  super.dispose();  }  getontheload() async {  redeemStream = await DatabaseMethods().getAdminReedemApproval();  redeemedHistoryStream = await DatabaseMethods().getApprovedRedeemHistory();  setState(() {});  }  Future<String> getUserPoints(String docId) async {  try {  DocumentSnapshot docSnapshot = await DatabaseMethods().getUserDocument(  docId,  );  if (docSnapshot.exists) {  var data = docSnapshot.data() as Map<String, dynamic>;  var points = data['Points'];  String pointsStr = points?.toString() ?? '0';  return pointsStr;  } else {  return '0';  }  } catch (e) {  return '0';  }  }  void approveRedeemRequest(  BuildContext context, DocumentSnapshot ds) async {  String userId = ds["UserId"];  String redeemRequestId = ds.id;  String pointsRedeemedStr = ds["Points"];  String currentUserTotalPointsStr = await getUserPoints(userId);  int currentUserTotalPoints = int.tryParse(currentUserTotalPointsStr) ?? 0;  int pointsToSubtract = int.tryParse(pointsRedeemedStr) ?? 0;  if (currentUserTotalPoints >= pointsToSubtract) {  int updatedUserTotalPoints = currentUserTotalPoints - pointsToSubtract;  String updatedUserTotalPointsStr = updatedUserTotalPoints.toString();  try {  await DatabaseMethods().updateUserPoints(  userId,  updatedUserTotalPointsStr,  );  await DatabaseMethods().updateAdminReedemRequest(redeemRequestId);  await DatabaseMethods().updateUserReedemRequest(  userId,  redeemRequestId,  );  double bdtConverted = pointsToSubtract \* pointsToBdtRate;  ScaffoldMessenger.of(context).showSnackBar(  SnackBar(  content: Text(  'Redeem request approved for ${ds["Name"]}! ${pointsToSubtract} points subtracted (Worth ${bdtConverted.toStringAsFixed(2)} BDT).',  ),  ),  );  } catch (e) {  ScaffoldMessenger.of(context).showSnackBar(  SnackBar(  content: Text(  'Error approving redeem request: $e',  ),  ),  );  }  } else { | ScaffoldMessenger.of(context).showSnackBar(  const SnackBar(  content: Text(  'Error: User has insufficient points for this redemption.',  ),  ),  );  }  }  @override  Widget build(BuildContext context) {  // This build method is kept minimal to satisfy the "only backend" request,  // as a StatefulWidget requires a build method.  // In a real application, this would contain your UI.  return Scaffold(  appBar: AppBar(title: const Text("Admin Redeem Backend")),  body: Center(  child: Column(  children: [  ElevatedButton(  onPressed: () => getontheload(),  child: const Text("Refresh Data (Backend Call)"),  ),  // You would typically use StreamBuilders here to display data,  // but for a pure backend snippet, we omit the detailed UI.  const Text("Backend logic is active. Check console for logs (if any)."),  ],  ),  ),  );  }  }  // Assuming DatabaseMethods and AppWidget are defined elsewhere in your project  // Example placeholder for DatabaseMethods and AppWidget to make this snippet runnable  // In your actual project, these would be in 'package:recycleapp/services/database.dart'  // and 'package:recycleapp/services/widget\_support.dart' respectively.  class DatabaseMethods {  final FirebaseFirestore \_firestore = FirebaseFirestore.instance;  Future<Stream<QuerySnapshot>> getAdminReedemApproval() async {  return \_firestore.collection('AdminRedeem').where('Status', isEqualTo: 'Pending').snapshots();  }  Future<Stream<QuerySnapshot>> getApprovedRedeemHistory() async {  return \_firestore.collection('AdminRedeem').where('Status', isEqualTo: 'Approved').snapshots();  }  Future<DocumentSnapshot> getUserDocument(String userId) async {  return await \_firestore.collection('users').doc(userId).get();  }  Future<void> updateUserPoints(String userId, String newPoints) async {  await \_firestore.collection('users').doc(userId).update({'Points': newPoints});  }  Future<void> updateAdminReedemRequest(String docId) async {  await \_firestore.collection('AdminRedeem').doc(docId).update({  'Status': 'Approved',  'RedeemedAt': Timestamp.now(),  });  }  Future<void> updateUserReedemRequest(String userId, String docId) async {  await \_firestore.collection('users').doc(userId).collection('RedeemRequests').doc(docId).update({  'Status': 'Approved',  'RedeemedAt': Timestamp.now(),  });  }  }  class AppWidget {  static TextStyle normaltextstyle(double size) {  return TextStyle(fontSize: size);  }  static TextStyle whitetextstyle(double size) {  return TextStyle(fontSize: size, color: Colors.white);  }  static TextStyle healinetextstyle(double size) {  return TextStyle(fontSize: size, fontWeight: FontWeight.bold);  }  } |
| --- | --- |

**Snippet 12: Database**

| import 'package:cloud\_firestore/cloud\_firestore.dart';  class DatabaseMethods {  Future addUserInfo(Map<String, dynamic> userInfoMap, String id) async {  return await FirebaseFirestore.instance  .collection("users")  .doc(id)  .set(userInfoMap);  }  Future addUserUploadItem(  Map<String, dynamic> userInfoMap,  String id,  String itemid,  ) async {  return await FirebaseFirestore.instance  .collection("users")  .doc(id)  .collection("Items")  .doc(itemid)  .set(userInfoMap);  }  Future addAdminItem(Map<String, dynamic> userInfoMap, String id) async {  return await FirebaseFirestore.instance  .collection("Requests")  .doc(id)  .set(userInfoMap);  }  Future<Stream<QuerySnapshot>> getAdminApproval() async {  return FirebaseFirestore.instance  .collection("Requests")  .where("Status", isEqualTo: "Pending")  .snapshots();  }  Future<Stream<QuerySnapshot>> getUserPendingRequests(String id) async {  return FirebaseFirestore.instance  .collection("users")  .doc(id)  .collection("Items")  .where("Status", isEqualTo: "Pending")  .snapshots();  }  Future<Stream<QuerySnapshot>> getAdminReedemApproval() async {  return FirebaseFirestore.instance  .collection("Reedem")  .where("Status", isEqualTo: "Pending")  .snapshots();  }  Future<Stream<QuerySnapshot>> getUserTransactions(String id) async {  return FirebaseFirestore.instance  .collection("users")  .doc(id)  .collection("Reedem")  .snapshots();  }  Future updateAdminRequest(String id) async {  return await FirebaseFirestore.instance  .collection("Requests")  .doc(id)  .update({"Status": "Approved", "ApprovedAt": Timestamp.now()});  }  Future updateUserRequest(String id, String itemid) async {  return await FirebaseFirestore.instance  .collection("users")  .doc(id)  .collection("Items")  .doc(itemid)  .update({"Status": "Approved", "ApprovedAt": Timestamp.now()});  } | Future updateAdminReedemRequest(String id) async {  return await FirebaseFirestore.instance.collection("Reedem").doc(id).update(  {"Status": "Approved", "RedeemedAt": Timestamp.now()},  );  }  Future updateUserReedemRequest(String id, String itemid) async {  return await FirebaseFirestore.instance  .collection("users")  .doc(id)  .collection("Reedem")  .doc(itemid)  .update({"Status": "Approved", "RedeemedAt": Timestamp.now()});  }  Future addUserReedemPoints(  Map<String, dynamic> userInfoMap,  String id,  String reedemid,  ) async {  return await FirebaseFirestore.instance  .collection("users")  .doc(id)  .collection("Reedem")  .doc(reedemid)  .set(userInfoMap);  }  Future addAdminReedemRequests(  Map<String, dynamic> userInfoMap,  String reedemid,  ) async {  return await FirebaseFirestore.instance  .collection("Reedem")  .doc(reedemid)  .set(userInfoMap);  }  Future<void> updateUserPoints(String userId, String newPoints) async {  try {  await FirebaseFirestore.instance.collection('users').doc(userId).update({  'Points': newPoints,  });  } catch (e) {  // Handle error appropriately  }  }  Future<DocumentSnapshot> getUserDocument(String userId) async {  return await FirebaseFirestore.instance  .collection('users')  .doc(userId)  .get();  }  Future<Stream<QuerySnapshot>> getApprovedItemsHistory() async {  return FirebaseFirestore.instance  .collection("Requests")  .where("Status", isEqualTo: "Approved")  .orderBy("ApprovedAt", descending: true)  .snapshots();  }  Future<Stream<QuerySnapshot>> getApprovedRedeemHistory() async {  return FirebaseFirestore.instance  .collection(  "Reedem",  )  .where("Status", isEqualTo: "Approved")  .orderBy("RedeemedAt", descending: true)  .snapshots();  }  } |
| --- | --- |

**Snipper 13: Shared Preference**

| import 'package:shared\_preferences/shared\_preferences.dart';  class SharedpreferenceHelper {  static String userIdKey = "USERKEY";  static String userNameKey = "USERNAMEKEY";  static String userEmailKey = "USEREMAILKEY";  static String userImageKey = "USERIMAGEKEY";  Future<bool> saveUserId(String getUserId) async {  SharedPreferences prefs = await SharedPreferences.getInstance();  return prefs.setString(userIdKey, getUserId);  }  Future<bool> saveUserName(String getUserName) async {  SharedPreferences prefs = await SharedPreferences.getInstance();  return prefs.setString(userNameKey, getUserName);  }  Future<bool> saveUserEmail(String getUserEmail) async {  SharedPreferences prefs = await SharedPreferences.getInstance();  return prefs.setString(userEmailKey, getUserEmail);  } | Future<bool> saveUserImage(String getUserImage) async {  SharedPreferences prefs = await SharedPreferences.getInstance();  return prefs.setString(userImageKey, getUserImage);  }  Future<String?> getUserId() async {  SharedPreferences prefs = await SharedPreferences.getInstance();  return prefs.getString(userIdKey);  }  Future<String?> getUserName() async {  SharedPreferences prefs = await SharedPreferences.getInstance();  return prefs.getString(userNameKey);  }  Future<String?> getUserEmail() async {  SharedPreferences prefs = await SharedPreferences.getInstance();  return prefs.getString(userEmailKey);  }  Future<String?> getUserImage() async {  SharedPreferences prefs = await SharedPreferences.getInstance();  return prefs.getString(userImageKey);  }  } |
| --- | --- |

### **4.3. Modules**

The system is divided into several modules:

* **User Module**
  + Sign in via Google
  + Submit recycling request
  + View request status and points
  + Redeem points
* **Admin Module**
  + Manual login
  + View, approve/reject requests
  + View and process redeem requests
* **Database Module**
  + Firestore collections: users, adminItems, redeem, admin
  + Subcollections for user-uploaded items
* **Security Module**
  + Firestore security rules based on user roles (admin/user)

## 

## 

## 

## 

## 

## **CHAPTER 5**

## **USER MANUAL**

### **5.1. System Requirements**

#### **5.1.1. Hardware Requirements**

| **Component** | **Minimum Requirement** |
| --- | --- |
| Processor | Intel Core i5 / ARM Equivalent |
| RAM | 8 GB (16 GB Recommended) |
| Storage | At least 500 GB free space |
| Device | Android/iOS device or modern browser |

#### **5.1.2. Software Requirements**

| **Software** | **Requirement** |
| --- | --- |
| Flutter SDK | 3.7.0 or higher |
| Dart SDK | Latest stable |
| Firebase CLI | Latest |
| Android Studio | Latest version |
| VS Code | Latest version |

### 

### **5.2. User Interfaces**

### **5.2.1. Panel A – User Panel**

* Login using Google
* Dashboard showing recycling categories
* Image picker for uploads
* Form to submit recycling requests
* Points tracking and redeem option
* Request status tracking (pending/approved)

#### **5.2.2. Panel B – Admin Panel**

* Manual login using ID and password
* View and approve/reject recycling requests
* Approve/reject redeem requests
* View user histories
* Logout option

#### 

#### **5.2.3. Login Credentials**

* **User**: Google Sign-In (via Firebase Authentication)
* **Admin**: Static ID and password stored in the Firestore admin collection or managed via Firebase Authentication. Admin:mehedi & Password:123456

CHAPTER 6

**CONCLUSION**

**6.1. Conclusion**

The **Recycle App** is an integrated solution aimed at making recycling easier and more accessible to everyone. The application efficiently brings users and administrators of the recycling system together using Flutter’s cross-platform capabilities in conjunction with Firebase’s powerful backend services. Engagement in the system is kept high through efficient Google authentication, categorized recycling request submissions, image uploads as well as a point reward system that incentivises engagements whilst streamlining waste management.Reporting tools allow administrators to effortlessly process requests and redemption activities. Thus, in addition to providing users entitlements for engagement, the application supports proactive environmental conservation by ensuring structured waste management frameworks are followed during expenditures.

## **6.2. Limitation**

## The **Recycle App** tackles key parts of recycling, but it has some drawbacks:

## The app's Role-Based Access Control (RBAC) has limits. It splits users into basic groups and admins, without detailed permission options.

## You need the internet to use the app. If you're offline, it won't work at all.

## The app handles requests and rewards by hand. This could cause hold-ups when lots of people use it.

**6.3. Future Works**

We can make the **Recycle App** better, faster, and able to handle more users in future versions.

* **Add Push Alerts:** To tell users right away about changes in recycling and reward requests.
* **Use Smart Learning:** Put in AI tech to sort recyclable items on its own from pictures users upload.
* **Build a Special Admin App:** A phone-friendly screen just for admins to do their jobs more easily.
* **Stronger Security:** Put in strong login and permission systems to keep access safe based on user roles.
* **Use Location Services:** Add address auto-fill and better pickup planning by detecting where users are.
* **More Reward Choices:** Let users cash in their points through other ways, not just mobile money.
* **Charts and Number Crunching:** Give admins tools to see how users act recycling patterns, and what types of items people turn in most, so they can make smart choices.

### 

### 

### **References**

1. Tamirat, N., Ambhaikar, S., Patil, A., Bisen, N., & Joshi, A. M. (2023). Scrapify: Digital Solution for Scrap Collection. *International Journal of Advanced Research in Science, Communication, and Technology (IJARSCT)*, 3(1), 299–307.<https://doi.org/10.48175/IJARSCT-7879>
2. Suruliraj, B., Olagunju, T., Nkwo, M., & Orji, R. (2020). Bota: A Personalized Persuasive Mobile App for Sustainable Waste Management. *Research paper on sustainable waste categorization and disposal facilitation through digital means.*
3. Lad, D. K., & Jaybhaye, K. R. (n.d.). E-Commerce Platform for Selling and Buying Industrial Scrap as Raw Material. *Proteus Journal.*<https://doi.org/10.37896/pj>Kahhat, R., Williams, E., Zhang, H., Allenby, B., Kavazanjian, E., & Kimball, D. (2008). Exploring e-waste management systems in the United States. *Resources, Conservation and Recycling*, 52(7), 955–964.
4. Zimring, C. A. (2005). Rags and Old Iron. In *Cash for Your Trash: Scrap Recycling in America* (pp. 12–36). Rutgers University Press.<http://www.jstor.org/stable/j.ctt5hj417.5>
5. Google. (n.d.). *Flutter – Build apps for any screen.* [https://flutter.dev](https://flutter.dev/)
6. Google Firebase. (n.d.). *Firebase – Build and run successful apps.* [https://firebase.google.com](https://firebase.google.com/)
7. Wagle, M. (2021). *Flutter Firebase Authentication Tutorial: Login & Signup.* Towards Data Science.<https://towardsdatascience.com/flutter-firebase-authentication-tutorial-login-signup-1e3c7ec34a5f>
8. Kothari, H. (2020). *HTTP Requests in Flutter using http Package.* Medium.<https://medium.flutterdevs.com/http-requests-in-flutter-using-http-package-4c3944d8b390>