DOCUMENT TYPE:

TECHNICAL DOCUMENT

DOCUMENT TITLE:

COHERENTPLUS – E-Commerce - Internet Payment Gateway -IPG FILES

TARGET AUDIENCE:

Project Management Team, Software, and Hardware Development Team, Internal use only.

(PRIVATE AND CONFIDENTIAL)

2025

Table of Contents

DOCUMENT VERSION

Version	Date	Name of Author	Description
1.0.0	3/11/2025	Liew Yee Shian	Document Creation

1 Abstract

This document outlines the steps and overview on how the internet payment gateway is integrated into the flutter application system.

SCOPE:

1) Overview of flutter versions and dependencies
2) Overview of internet payment gateway
3) How internet payment gateway is integrated with flutter
4) Outputs

1.1 Abbreviations

Code	Description
IPG	Internet Payment Gateway
tx	Transaction
&Pay	AmpersandPay SDN BHD

2 Overview

2.1 Overview of flutter version and dependencies

This is version of flutter 2025 October. And these are the dependencies that we will be using in this flutter ecommerce project.

The words in blue are the name of the dependencies, and the green numbers are the latest version used. The versions will change overtime depending on when flutter is updated.

You can get their latest by searching it here:

https://pub.dev/ or just add whatever dependencies name at the back to get it

eg. https://pub.dev/packages > https://pub.dev/packages/crypto

```
dependencies:
  flutter:
    sdk: flutter

  # The following adds the Cupertino Icons font to your application.
  # Use with the CupertinoIcons class for iOS style icons.
  cupertino_icons: ^1.0.8
  carousel_slider: ^5.1.1
  url_launcher: ^6.3.2
  firebase_core: ^4.1.1
  cloud_firestore: ^6.0.2
  http: ^1.5.0
  crypto: ^3.0.6
  flutter_inappwebview: ^6.1.5
  webview_flutter: ^4.13.0
  firebase_auth: ^6.1.1
  shared_preferences: ^2.5.3
```

Figure 2.1 flutter dependencies and version

2.1.1 Dependencies Definition

These are the purpose of each dependency that is used in the files.

Dependency	Version	Purpose
flutter	SDK	Core Flutter SDK – provides the framework for building cross-platform mobile apps
cupertino_icons	^1.0.8	Provides iOS-style icons for Cupertino widgets
carousel_slider	^5.1.1	Enables creating image or widget carousels/sliders in the app
url_launcher	^6.3.2	Allows opening external URLs, email, phone calls, or other apps from Flutter
firebase_core	^4.1.1	Core Firebase SDK – required to initialize Firebase in Flutter
cloud_firestore	^6.0.2	Enables reading/writing data to Firebase Cloud Firestore (NoSQL database)
http	^1.5.0	Provides HTTP client functionality for REST API calls
crypto	^3.0.6	Provides cryptographic functions like hashing (MD5, SHA) and HMAC
flutter_inappwebview	^6.1.5	Provides advanced WebView functionality with more control over web content
webview_flutter	^4.13.0	Official Flutter plugin to embed simple WebViews in the app
firebase_auth	^6.1.1	Firebase Authentication SDK – supports email/password, social login, and more

Table 2.1.1 flutter dependencies 2025 version and purpose

2.1.2 File Dependencies

Each files have their own specific dependencies that is used to make it work. So, in this section we will be discussing more the important files that is needed to make the IPG work.

The main files that are involved in the IPG are:

1) order_checkout.dart
2) call_api.dart
3) payment_webview.dart
4) query_transaction.dart
5) finish_checkout.dart

File Name	Functionality	Key Points
order_checkout.dart	Handles the initial checkout process: collecting order details, user inputs, and preparing order data	Typically interacts with the UI for order summary and user confirmation before payment
call_api.dart	Manages API requests related to orders and payments	Sends order data to backend, receives tx info or status, and returns responses
payment_webview.dart	Opens a WebView for the payment gateway	Displays the payment provider's interface inside the app; monitors completion or failure events
query_transaction.dart	Queries the backend or payment gateway for tx status	Checks if payment succeeded, pending, or failed; used to update UI and trigger next steps
finish_checkout.dart	Handles post-payment logic: finalizing order, updating order status, showing success or failure page	Confirms tx completion, updates database/UI, and optionally redirects user to receipt

Table 2.1.2 File name and its functionalities

3 ORDER_CHECKOUT.DART

This order_checkout.dart implements the checkout page. It allows users to enter delivery information, select a payment method, review their cart summary, save the order to Firestore, and proceed with payment via an online payment gateway (call_api.dart).

3.1 Key Functionalities

The order_checkout.dart file implements the **checkout and payment initiation interface** of the Flutter SDK.
It connects the user's frontend actions with backend operations such as data storage, payment gateway communication, and tx handling.

No.	Functionality	Description
1	**User Input Collection**	Provides text fields for name, address, email, and phone number, along with a dropdown to select payment method (Online Banking, Credit Card, or E-Wallet).
2	**Form Validation**	Uses a GlobalKey<FormState> (_formKey) to validate all user inputs before proceeding with the checkout process.
3	**Order Data Preparation**	Constructs an order object containing order ID, user ID, payment method, subtotal, product list, and customer details.
4	**Firestore Integration**	Saves the order data into the Firestore orders collection with status set to "pending_payment" and links it to the logged-in user.
5	**Payment Readiness Control**	Displays the "Pay Now" button only after a valid order has been successfully stored in Firestore, ensuring that incomplete orders cannot proceed to payment.
6	**Payment Gateway Integration**	Initiates the payment process by calling CallApi.processPayment() (from call_api.dart), passing the order details, total amount, and user information.
7	**Secure tx Handling**	Uses the SDK's built-in signing process (SHA-512) through call_api.dart to ensure that all payment requests are securely verified.

8	**Payment WebView Redirection**	Redirects the user to the PaymentWebView page using the returned checkoutUrl to complete the payment within a secure in-app browser.
9	**User Feedback and Error Handling**	Displays contextual messages using SnackBar widgets to inform users of success, validation errors, or failed payment attempts.
10	**Dynamic UI Components**	Builds responsive widgets such as _buildTextField() and _buildCartSummary() to show cart contents, totals, and interactively manage form data.

3.2 Dependencies Used

Dependency	Purpose / How It's Used
flutter/material.dart	Provides UI widgets, layouts, buttons, forms, styling, and input validation.
cloud_firestore/cloud_firestore.dart	Enables reading and writing order data to Firestore database. Used to save orders and retrieve user information.
call_api.dart	Handles online payment API integration. Used to process payments with the selected method and order data.
payment_webview.dart	Opens a WebView to show the payment gateway page after initiating payment.

Table 3.2 dependencies in order_checkout.dart and its purpose

3.2.1 Flutter/material.dart

The flutter/material.dart package provides the core Material Design UI components for Flutter applications. In order_checkout.dart, it is used to build the visual structure of the checkout page, including the Scaffold, AppBar, TextFormField for user input, DropdownButtonFormField for payment selection, ElevatedButton for actions like "Proceed" and "Pay Now," and layout widgets such as Column and Row. It ensures a consistent, responsive, and visually appealing user interface following Material Design principles.

Component / Widget	Purpose in OrderCheckout.dart
Scaffold	Provides the basic page layout structure with AppBar, body, and background color.
AppBar	Displays the top navigation bar with the title “Checkout” and a red accent background.
SingleChildScrollView	Makes the page scrollable to accommodate all form fields and the cart summary.
Column / Row	Arranges widgets vertically (Column) and horizontally (Row) for layout of buttons and form fields.
Text	Shows labels, order ID, and summary information with different font styles and colors.
TextFormField	Creates form fields for user inputs: Full Name, Address, Email, Phone Number.
DropdownButtonFormField	Provides a dropdown for selecting the payment method.
ElevatedButton	Buttons for “Cancel”, “Proceed”, and “Pay Now” actions with custom styling.
Padding / SizedBox	Controls spacing between widgets for proper layout and readability.
SnackBar	Displays messages to the user for errors or confirmation (e.g., “Order saved”).
Card / ListTile	Displays the items in the cart with product image, name, quantity, and price.

TextStyle / Colors	Controls the appearance of text and UI elements to match the app's dark theme.

Table 3.2.1 widgets inside flutter material dependency

3.2.2 Cloud_firestore/cloud_firestore.dart

The cloud_firestore dependency is used to store and manage order data in a Firebase Firestore database. When a user fills in their delivery details and clicks **"Proceed"**, the app validates the form and saves all order information into the orders collection. Each order is associated with a unique orderId and the userId of the logged-in Firestore user, allowing tracking of order status and linking orders to specific users.

Dependency	**Usage in File**	**Purpose**
FirebaseFirestore	FirebaseFirestore.instance.collection('orders').doc(widget.orderId).set(orderData)	Saves order details to Firestore
DocumentReference	doc(widget.orderId)	References a specific order document
CollectionReference	collection('orders')	Points to the orders collection
set()	.set(orderData)	Writes order data into Firestore

Table 3.2.2 Firestore dependency

Functionality	Explanation	Code Reference
Save order data	Saves customer info and cart items to Firestore	_handleProceed() method, FirebaseFirestore.instance.collection ('orders').doc(widget.orderId).set(orderData)
Link order to user	Associates order with current user	orderData["userId"] = widget.userId
Track order status	Marks order as pending payment and delivery not started	orderData["status"] = "pending_payment" & orderData["deliveryStatus"] = "not_started"
Store product list	Stores cart items with id, name, qty, price, image	orderData["productList"] = widget.cartItems.map(...).toList()

Table 3.2.2 Functionality of firestore dependency

3.2.3 call_api.dart

The call_api.dart file handles sending the checkout information to your payment gateway and retrieves the URL and tx ID needed to redirect the user to the payment page. It acts as the bridge between your Flutter app and the external payment system.

Component / Function	Usage in File	Purpose
CallApi class	_handlePay() in OrderCheckout	Provides static methods to call external payment API
processPayment()	_handlePay()	Sends order details, payment method, user info, and items to payment API and returns checkout URL and tx ID
http.post()	Inside processPayment()	Makes a POST request to the payment API with JSON payload
payload	Inside processPayment()	Contains order ID, user ID, total amount, items, payment method, and customer info
jsonEncode(payload)	Inside processPayment()	Converts Dart map into JSON for API request
jsonDecode(response.body)	Inside processPayment()	Parses API response into a Dart map for use in app
return data	Inside processPayment()	Returns API response to OrderCheckout for redirecting to payment page

Table 3.2.3 functionality of call_api.dart link file

3.2.4 payment_webview.dart

Payment_webview.dart provides a WebView screen that loads the external payment page for the user to complete their tx. It tracks the order and tx ID and can detect payment completion to update the app accordingly.

Component / Function	Usage in File	Purpose
PaymentWebView class	_handlePay() in OrderCheckout	Opens a WebView to display the payment gateway checkout page
checkoutUrl	Constructor parameter	URL provided by the payment API to complete the tx
orderId	Constructor parameter	Tracks the order associated with the payment
txId	Constructor parameter	Tx ID returned by the payment API
userId	Constructor parameter	Firestore user ID for linking payment status to user
WebView widget	Inside PaymentWebView	Renders the payment gateway page within the app
navigationDelegate	Optional WebView property	Monitors URL changes to detect payment success or failure
onPageFinished / onWebResourceError	Optional WebView callbacks	Handle events like page load complete or errors in the WebView

Table 3.2.4 functionality of payment_webview.dart link file

3.3 How Flutter Use This Order_checkout.dart File

The order_checkout.dart file functions as the checkout interface in the Flutter application. It serves as the main bridge between the user, the app's Firestore database, and the payment gateway integration handled by call_api.dart.

It collects user input, validates order details, saves the order to Firebase Firestore, and then triggers the payment process through the API SDK.

Step	Action	Explanation
1	User fills out the checkout form	The user enters delivery details such as name, address, email, and phone number, and selects a payment method from the dropdown list.
2	App validates input fields	The form is validated through _formKey to ensure all required information is correctly entered before proceeding.
3	Order data prepared	The app constructs an order object containing the customer information, product list, total amount, and order ID.
4	Order saved in Firestore	The _handleProceed() method uploads the order data into the orders collection in Firebase Firestore with a status of "pending_payment."
5	"Pay Now" button activated	Once the order is saved, the interface enables the Pay Now button, allowing the user to proceed with payment.
6	Payment process initiated	When the user taps "Pay Now," the _handlePay() function calls the CallApi.processPayment() method from call_api.dart.
7	Payment request sent to &Pay	The CallApi class builds the payment payload, signs it using SHA-512, and communicates with the &Pay API.
8	Checkout URL received	If the API call succeeds, it returns a checkoutUrl and txId, which are used to redirect the user to the payment gateway.
9	User redirected to payment page	Flutter navigates to PaymentWebView, which loads the &Pay checkout interface, allowing the user to complete the payment.

10	Tx tracking maintained	Tx details including order ID, txId, user ID, and payment status remain stored in Firestore for monitoring and history.

Table 3.3 steps on order_checkout.dart

4 CALL_API.DART

4.1 Key Functionalities

The call_api.dart file serves as the core SDK module responsible for establishing secure communication between the Flutter application and the &Pay IPG.
It performs request preparation, cryptographic signing, data transmission, response handling, and tx recording.

No.	Functionality	Description
1	**Merchant Configuration**	Defines essential merchant credentials including merchantId, integrationKey, and API endpoint (apiUrl) for connecting to &Pay sandbox or live environment.
2	**Payment Channel Mapping**	Uses the _getChannelCode() method to map Flutter payment options (e.g., Online Banking, Credit Card, E-Wallet) to &Pay required tx channel codes (DD, CC, EW).
3	**SHA-512 Signature Generation**	Implements _generateSignature() to combine the JSON payload with the integrationKey and apply **SHA-512 hashing**, ensuring message integrity and authentication.
4	**Payload Construction**	Builds a structured JSON object containing merchant details, tx information, customer data, and product list, following &Pay API format.
5	**Secure HTTP Request Handling**	Uses the http.post() method to send the signed payload via HTTPS with required headers (Content-Type, signature) to the &Pay API.
6	**API Response Processing**	Decodes and validates the server's response. If successful (ret == 0), it extracts the txId and checkout URL (checkoutUrl).

7	**Firestore Tx Logging**	Automatically saves tx details (order ID, txId, status, userId, timestamps) into the Firestore orders collection for tracking and reconciliation.
8	**Error Handling and Logging**	Uses try–catch blocks and Dart's log() function to record API responses, errors, or exceptions for debugging and reliability analysis.
9	**Frontend Integration Support**	Returns structured response data (e.g., checkoutUrl, txId) back to Flutter's UI layer (e.g., order_checkout.dart) for redirection to the payment WebView.

Table 4.1 functionalities of call_api.dart

4.2 Dependencies Used

Dependency	Import Statement	Purpose / Usage
dart:convert	import 'dart:convert';	Used for encoding and decoding JSON (jsonEncode, jsonDecode) when preparing the request payload and parsing the API response.
dart:developer	import 'dart:developer';	Provides the log() function to print structured debugging information to the console.
crypto	import 'package:crypto/crypto.dart';	Used to generate a SHA-512 hash signature for securely signing payment requests before sending to &Pay.
http	import 'package:http/http.dart' as http;	Handles HTTPS POST requests to communicate with &Pay's payment API endpoint.
cloud_firestore	import 'package:cloud_firestore/ cloud_firestore.dart';	Connects to Firebase Firestore to store and update tx details (txId, status, timestamps, userId) after initiating payment.

Table 4.2 Dependencies used in call_api.dart

4.2.1 Dart Convert

The dart:convert library in Dart is used for encoding and decoding data — mainly converting between JSON strings and Dart objects like Map and List.

It allows your app to:

- Encode (serialize) Dart data into JSON before sending it to APIs or saving it.
- Decode (deserialize) JSON responses from APIs back into usable Dart data structures.

In short, dart:convert acts as the bridge between Dart objects and JSON data used in web APIs.

```dart
// Encode to JSON and generate signature for security
final jsonString = jsonEncode(payload); // Encode Dart Map (payload) into JSON string
final signature = _generateSignature(jsonString);
```

Figure 4.2.1 Encode data into JSON

```dart
// ✅ Handle successful response
if (response.statusCode == 200) {
  final Map<String, dynamic> data = jsonDecode(response.body);//Decode JSON response from API back into a Dart Map
```

Figure 4.2.1 Decode JSON into String

4.2.2 Dart Developer

The dart:developer library provides built-in tools for debugging and performance monitoring in Dart.

It lets developers:

- Log diagnostic messages using log() (shown in the Debug Console or Dart DevTools).
- Record custom events and performance metrics.
- Interact with Dart DevTools for profiling and inspecting runtime behavior.

In short, it's mainly used for **debugging and logging** during development.

```
// Log payload and signature for debugging
log("=== JSON Payload ===\n$jsonString");
log("=== Signature === $signature");
```

Figure 4.2.2 debugging

```
// Log response details
log("=== Response Code: ${response.statusCode} ===");
log("=== Response Body: ${response.body} ===");
```

Figure 4.2.2 error code response

```
} catch (e) {
  // Catch and log any runtime errors (network or parsing issues)
  log("❌ processPayment Error: $e");
  return {"error": true, "message": e.toString()};
}
```

Figure 4.2.2 runtime error response

****These error responses will be displayed on the UI if there is any error****

4.2.3 Crypto

The crypto package in Dart provides cryptographic hashing and message authentication tools. It allows developers to generate secure hashes (like SHA-1, SHA-256, SHA-512, MD5) and verify data integrity — essential for secure API communication, password storage, and digital signatures.

In this file, it's used to generate a SHA-512 signature that authenticates and verifies your payment request before sending it to &Pay.

Component	Description	Purpose in call_api.dart
import 'package:crypto/crypto.dart';	Imports the crypto library	Gives access to hashing algorithms like SHA-512
import 'dart:convert';	Imports Dart's encoding utilities	Provides utf8.encode() used before hashing
utf8.encode(jsonString + integrationKey)	Converts the payment data and key into bytes	Prepares data for hashing
sha512.convert(bytes)	Runs the SHA-512 hashing function	Generates a secure, 512-bit hash digest
digest.toString()	Converts hashed bytes into a human-readable hex string	Produces the final signature string
signature (in header)	Used as an HTTP header in API request	Authenticates the request between Flutter and &Pay

Table 4.2.3 Crypto dependency components

These are the steps that is used with this crypto dependency in call_api.dart file.

Step	Action	Purpose	Code
1	Build JSON payload	Collect payment data	final jsonString = jsonEncode(payload);
2	Add integration key	Include secret for verification	jsonString + integrationKey
3	Encode to bytes	Prepare data for hashing	final bytes = utf8.encode(jsonString + integrationKey);
4	Hash using SHA-512	Create secure digital fingerprint	final digest = sha512.convert(bytes);
5	Convert to hex string	Make it readable and transmittable	final signature = digest.toString();

6	Attach to HTTP header	Authenticate API request	headers: {
			"Content-Type": "application/json",
			"signature": signature,
			}

7	Server verifies	Ensure message integrity and authenticity	

Table 4.2.3 steps in call_api.dart

4.2.4 HTTP

The http package in Dart provides tools for sending and receiving data over the web using the HTTP protocol.
In this file, it's used to send payment requests to the &Pay API using a secure HTTPS POST method and handle the API's response.

It acts as the communication bridge between the Flutter app and the external payment server.

Component	Description	Purpose in call_api.dart
import 'package:http/http.dart' as http;	Imports the Dart HTTP client library	Enables network communication (GET, POST, etc.)
Uri.parse(apiUrl)	Converts a string URL into a Uri object	Required by the HTTP client to know where to send the request
http.post(...)	Sends an HTTP POST request	Transmits the payment data securely to the &Pay API
headers	A map of key-value pairs attached to the request	Includes metadata like Content-Type and the generated signature
body: jsonString	Contains the JSON-encoded payment payload	Sends customer and order data to the payment gateway
response.statusCode	HTTP status code returned by the server	Indicates whether the request succeeded (200) or failed
response.body	The content returned by the server	Contains the API response (e.g., checkout URL, tx status)
jsonDecode(response.body)	Decodes JSON string from the server back into a Dart map	Allows your app to process and extract API data easily

Table 4.2.4 http dependency usage

4.2.5 Cloud Firestore

The cloud_firestore package allows your Flutter app to interact with Google Cloud Firestore, a NoSQL cloud database.
In call_api.dart, it's used to store and track payment tx records such as txId, status, timestamps, and the userId of the person making the order.

It ensures that every tx processed through &Pay is also logged and retrievable in your own app's database.

Component	Description	Purpose in call_api.dart
Import 'package:cloud_firestore/cloud_firestore.dart';	Imports Firestore library	Enables connection to Firebase's Firestore database
FirebaseFirestore.instance	Access point for the Firestore database	Provides methods to read/write data
.collection('orders')	Refers to the "orders" collection in Firestore	Groups all order documents for organized storage
.doc(orderId)	Creates or references a document identified by orderId	Uniquely identifies each tx record
.set({...}, SetOptions(merge: true))	Writes or updates data in the document	Saves tx info (txId, status, timestamps, etc.)
"userId": userId	Stores the user's Firestore ID	Links the order record to a specific logged-in user
"status": "pending"	Tracks current tx status	Useful for checking payment updates later (section 6)
"createdAt" / "updatedAt"	Timestamps	Record when the tx was initiated and last updated

Table 4.2.5 firestore usage

4.3 How Flutter Use This call_api.dart File

The call_api.dart file acts as the backend communication layer between the Flutter application and the &Pay IPG.
It is used whenever a user confirms an order and proceeds to make a payment.

In the app's checkout or order confirmation page, the CallApi.processPayment() method is called with the user's details, selected payment method, item list, total amount, and unique order ID.

Step	Action	Explanation
1	User clicks "Proceed to Payment"	In the Flutter checkout page, a button triggers a function that calls CallApi.processPayment()
2	App prepares payment info	The app collects all the details like total amount, order ID, payment method, and customer info
3	CallApi builds and signs the request	The call_api.dart file creates a JSON payload and generates a SHA-512 signature for security
4	Request sent to &Pay	It uses http.post() to send the data to the &Pay API endpoint
5	&Pay returns checkout URL	If successful, the API sends back a link (checkoutUrl) where the user can make the payment
6	App redirects user to payment page	Flutter uses that checkout URL to open the payment interface (in browser or WebView)
7	**Tx** saved in Firestore	The call_api.dart SDK also saves the tx data (txId, status, userId) into Firebase Firestore for tracking

Table 4.3 overall steps of call_api.dart

5 PAYMENT_WEBVIEW.DART

5.1 Functionality of Payment_webview.dart

The payment_webview.dart file provides the Flutter SDK's payment execution and tracking interface, enabling users to complete payments directly within the app using an embedded webview. It continuously monitors the tx status through periodic API polling and updates Firestore records accordingly.

No.	Functionality	Description
1	WebView Integration	Embeds the &Pay checkout page within Flutter using the InAppWebView widget, allowing seamless in-app payment without leaving the application.
2	Tx Polling Mechanism	Implements _startPolling() using a Timer.periodic loop that checks the tx status every 2 seconds through the &PayQuery.queryTransaction() API call.
3	Real-Time Status Detection	Continuously monitors tx responses and webview URL redirects to detect payment outcomes such as success, failed, or cancelled.
4	Firestore Order Update	Updates the corresponding order document in Firestore with real-time payment status and timestamps once the tx is finalized.
5	Cart Synchronization	Automatically clears purchased items from the user's Firestore cart after successful payment, ensuring the user's cart remains up to date.
6	Error Handling and Time-outs	Cancels polling after exceeding the maximum allowed attempts or upon result handling, preventing infinite loops and ensuring robust error management.
7	Automatic Navigation	Redirects the user to payment result upon tx completion.

Table 5.1 functionalities of payment_webview.dart

5.2 Dependencies Used

No.	Dependency	Description / Purpose
1	dart:async	Provides asynchronous programming support such as Timer, Future, and Stream, used for polling and delayed tasks during payment verification.
2	package:flutter/ material.dart	Supplies Flutter's Material Design widgets used to build and style the payment interface**.**
3	package:flutter_ inappwebview/ flutter_inappweb view.dart	Enables the app to embed a web browser inside the Flutter app, allowing users to complete payments within a secure WebView without leaving the app.
4	package:cloud_fire store/ cloud_firestore.dart	Connects the app to Cloud Firestore for reading and writing data. It is used to update order status**,** record timestamps, and clear purchased items from the user's cart.
5	query_transaction. dart	Contains logic to query the &Pay tx API, allowing the app to check payment status periodically using the &Pay Query.queryTransaction() method.
6	finish_checkout. dart	Displays the final payment result screen after the tx is completed or failed, confirming the checkout status to the user.

Table 5.2 dependencies used in payment_webview.dart

5.2.1 Dart Async

The dart:async library in payment_webview.dart enables asynchronous operations such as waiting, polling, and handling delays. It ensures smooth payment processing without freezing the UI, allowing the app to periodically check the tx status, delay actions when necessary, and manage background tasks efficiently.

Component	Description	Purpose in payment_webview.dart
import 'dart:async';	Imports Dart's asynchronous programming library	Enables background operations like timers and delayed tasks
Timer.periodic()	Runs a function repeatedly at fixed intervals	Polls the &Pay API every 2 seconds to check payment status
Future.delayed()	Executes a task after a specified delay	Waits 1 minute before starting tx polling
Future<void>	Represents async functions with no return value	Used for Firestore updates and cart clearing
await	Pauses execution until an async task completes	Ensures operations like Firestore writes finish before continuing

Table 5.2.1 functionalities of dart async dependency

5.2.2 Package:flutter/material.dart

The package:flutter/material.dart dependency provides the core Flutter Material Design components used to build the user interface of payment_webview.dart. It handles UI layout, navigation, and visual styling such as the app bar, progress indicator, and scaffold structure.

Component	Description	Purpose in payment_webview.dart
import 'package:flutter/material.dart';	Imports Flutter's Material Design library	Enables use of widgets for building the UI
Scaffold	Provides a basic visual layout structure	Wraps the entire payment page with app bar and body
AppBar	Displays a top navigation bar	Shows the page title and back button during payment
IconButton	A clickable icon widget	Used for the back button to cancel payment
Text	Displays text content	Used for the page title "Complete Payment"
LinearProgressIndicator	Shows loading progress visually	Indicates webview loading progress
Navigator.pushReplacement()	Navigates between pages	Redirects to the finish checkout page after payment completion
setState()	Updates the widget's state dynamically	Refreshes progress bar as the webview loads

Table 5.2.2 functionalities of package flutter dependency

5.2.3 Flutter_inappweb view.dart

This Flutter_inappweb view.dart enables the Flutter application to embed and control a directly within the app.

In the payment_webview.dart file, it is used to display the &Pay payment page, track loading progress, and detect redirection URLs (success/failure) to determine payment status.

Component	Description	Purpose in payment_webview.dart
import 'package:flutter_inappwebview/flutter_inappwebview.dart';	Imports the InAppWebView package	Allows Flutter to render a web browser view inside the app
InAppWebView	Core webview widget	Displays the payment checkout page inside the app rather than redirecting to an external browser
InAppWebViewController	Controller class for webview instance	Enables controlling or monitoring the webview (e.g., checking URLs, progress updates)
InAppWebViewSettings	Configuration class for webview behavior	Enables JavaScript, inline media playback, and secure browsing settings
URLRequest(url: WebUri(widget.checkoutUrl))	Defines the webview's initial page to load	Opens the payment checkout URL when the user reaches this screen
onProgressChanged	Event listener for load progress	Updates the progress bar (LinearProgressIndicator) as the payment page loads
onLoadStop	Event triggered when page finishes loading	Detects when the webview redirects to "success" or "failed" URLs to handle tx result

Table 5.2.3 functionality of flutter in app view dependency

5.2.4 Firestore

The Firestore dependency in this file is used to store, update, and synchronize payment tx data in real-time. It tracks the user's order status (paid / failed), updates timestamps, and clears the purchased items from the user's cart upon successful payment.

Component	Description	Purpose in payment_webview.dart
import 'package:cloud_firestore/cloud_firestore.dart';	Imports the Cloud Firestore library	Allows communication between the Flutter app and Firebase Firestore database
FirebaseFirestore.instance	Singleton instance of the Firestore service	Provides access to Firestore's read/write and update methods
.collection('orders')	References the "orders" collection	Stores and updates each order document related to payment tx
.doc(widget.orderId)	Retrieves or updates a specific order document	Identifies and updates the correct tx record by order ID
.update({...})	Updates existing document fields	Changes order status to "paid" or "failed" after tx verification
.collection('users').doc(widget.userId).collection('cart')	Accesses the user's cart subcollection	Finds and deletes purchased items after successful payment
.get()	Fetches all documents from the collection	Reads the cart items to determine which ones to remove
.delete()	Deletes specific documents	Removes items that were part of the completed order from the user's cart
DateTime.now()	Generates current timestamp	Records the exact time the payment was updated in Firestore

Table 5.2.4 functionalities of firestore dependency

5.2.5 Query_transaction.dart

The query_transaction.dart file is responsible for communicating with the &Pay API to verify the status of a tx. It is called periodically by payment_webview.dart during payment polling to check whether the tx has succeeded, failed, or is still pending.

Component	Description	Purpose in payment_webview.dart
import 'query_transaction.dart';	Imports the tx query module	Enables API-based tx verification with &Pay
AmpersandPayQuery. query Transaction(widget.txId)	Static method call to query the API using the tx ID	Sends HTTP requests to check tx status from &Pay
data["txStatus"]	Extracted response field from the API	Used to determine if the payment is “SUCCESS”, “PAID”, or still pending
data["ret"] == 1201	Return code from &Pay API	Indicates that the tx is not yet available or still processing
try...catch (e)	Error handling block	Prevents the app from crashing if the API request fails during polling

Table 5.2.5 functionalities of query_transaction.dart import

5.2.6 Finish_checkout.dart

The finish_checkout.dart file handles what happens after payment completion. It is responsible for confirming the tx, updating the order record in Firestore, and displaying a success or failure message to the user. This ensures that the checkout process ends cleanly and that tx data remains synchronized between the app and backend.

Component	**Description**	**Purpose in payment_webview.dart**
import 'finish_checkout.dart';	Imports the post-payment handler	Allows the app to execute final checkout logic after payment confirmation
FinishCheckoutPage(orderId: widget.orderId, txId: widget.txId)	Navigates to the final checkout page	Displays the result of the payment and order confirmation
updateTransactionStatus()	Method inside finish_checkout.dart	Updates the order's status (e.g., "paid", "failed") in Firestore
showSuccessMessage() / showErrorMessage()	UI feedback functions	Provides visual confirmation of tx outcome to the user
Navigator.pushReplacement()	Replaces the current WebView screen with FinishCheckout page	Cleans up the navigation stack and redirects user after completion

Table 5.2.6 functionalities of finish_checkout.dart import

5.3 How Flutter Use This payment_webview.dart File

The payment_webview.dart file is used by Flutter to display and monitor the payment process in real-time. It integrates a web-based payment gateway directly inside the Flutter app using a WebView and continuously checks the tx status through API polling. This file bridges the gap between the user interface and backend payment verification, ensuring a seamless checkout experience.

Step	Action	Explanation
1	**User proceeds to payment**	When the user clicks the "Proceed to Payment" button on the checkout page, the app navigates to the PaymentWebView widget.
2	**Checkout page opens in WebView**	The widget loads the checkoutUrl using the InAppWebView component, allowing the user to complete payment without leaving the app.
3	**Tx polling starts**	After a short delay, the _startPolling() method automatically queries the tx status from the &Pay API (via query_transaction.dart).
4	**Firestore updates order status**	Once the API confirms the payment result, _handlePaymentResult() updates the Firestore record (e.g., marking order as "paid" or "failed").
5	**Cart items cleared**	If the payment succeeds, _clearCart() removes the purchased products from the user's Firestore cart collection.
6	**Redirect to final confirmation**	Finally, Flutter navigates to FinishCheckoutPage (from finish_checkout.dart) to show the success or failure message.

Table 5.3 steps in payment_webview.dart

6 QUERY_TRANSACTION.DART

6.1 Functionalities of Query_transaction.dart

The query_transaction.dart file functions as a backend communication and status tracking module within the Flutter payment SDK. It connects directly to the &Pay API to verify and monitor tx statuses. It can perform both one-time status checks and automated polling, and it updates Firestore records when payments are confirmed.

No.	Functionality	Description
1	**Tx Status Query**	Sends a secure POST request to the &Pay /tx/query API endpoint using the tx ID and merchant credentials to retrieve the current payment status.
2	**Secure Signature Generation**	Generates a SHA-512 signature combining the request payload and integration key to authenticate and validate all outgoing API requests.
3	**Automated Polling Mechanism**	Implements a Timer.periodic function in pollTransaction() to repeatedly query the tx status every few seconds until a result is obtained or timeout occurs.
4	**Firestore Integration**	Automatically updates the corresponding order document in Firestore with the latest tx status (paid, failed, etc.) and timestamps once payment confirmation is received.
5	**Timeout and Retry Control**	Limits the number of polling attempts (default: 60) to prevent endless API calls and adds retry logic for delayed or missing tx responses.
6	**Error Logging and Exception Handling**	Captures and logs all HTTP errors, exceptions, and invalid responses using the dart:developer log utility to aid debugging and API monitoring.
7	**Data Integrity Assurance**	Ensures accurate synchronization between &Pay's tx records and Firestore by validating tx results before database updates.

Table 6.1 functionalities of query_transaction.dart file

6.2 Dependencies Used

No.	Dependency	Description / Purpose
1	dart:async	Provides asynchronous operations like Timer and Future, enabling timed polling and repeated tx status checks.
2	dart:convert	Handles JSON encoding and decoding, allowing the SDK to send properly formatted data and interpret API responses from &Pay.
3	dart:developer	Offers logging functionality via log(), used for debugging, error tracing, and tracking API responses during tx polling.
4	package:crypto/ crypto.dart	Generates cryptographic SHA-512 hashes for securely signing API requests before they are sent to the &Pay server.
5	package:http/http.dart	Facilitates HTTP communication by sending POST requests to the &Pay query endpoint and receiving tx status data.
6	package:cloud_ firestore/cloud_ firestore.dart	Connects to Firebase Firestore, allowing automatic updates to order documents when a tx status changes to “paid.”

Table 6.2 dependencies used in query_transaction.dart

6.2.1 Dart Async

The dart:async dependency in query_transaction.dart enables asynchronous operations essential for real-time payment verification. It powers features like delayed execution, periodic polling with Timer.periodic(), and non-blocking HTTP requests using Future and await, ensuring the app continuously checks tx status from &Pay without freezing the user interface.

Component	Description	Purpose in query_transaction.dart
import 'dart:async';	Imports Dart's asynchronous programming library	Enables use of Future, Timer, and Stream for non-blocking tasks such as periodic tx polling.
Timer.periodic()	Creates a timer that runs a callback function repeatedly at fixed intervals	Used to poll the &Pay API every few seconds until the payment status is confirmed.
Future.delayed()	Delays the execution of code for a specific duration	Ensures that the first polling request only starts after one minute, preventing premature queries.
Future<Map<String, dynamic>?>	Defines an asynchronous function that returns a future Map result	Represents the asynchronous HTTP request and response handling for querying tx data.
async / await keywords	Dart syntax for managing asynchronous code execution	Allows the app to wait for HTTP responses and Firestore updates without blocking the main UI thread.

Table 6.2.1 functionality of dart async in query_transaction.dart file

6.2.2 Dart Convert

The dart:convert dependency provides JSON encoding and decoding functionality in Dart. In this file, it is used to convert Dart objects (maps) into JSON strings for API requests (jsonEncode) and to parse JSON responses from the &Pay API (jsonDecode). It ensures that data can be safely sent to and received from the payment API in a format both Dart and the server understand.

Component	**Description**	**Purpose in query_transaction.dart**
import 'dart:convert';	Imports Dart's built-in JSON library	Enables conversion between Dart objects and JSON for API communication
jsonEncode(body)	Converts a Dart map into a JSON-formatted string	Prepares the request payload to send to the &Pay API
jsonDecode(response.body)	Converts a JSON-formatted string response into a Dart map	Extracts tx details like txStatus and ret from the API response
utf8.encode()	Converts a string to UTF-8 bytes	Used in combination with sha512 to generate a secure request signature for API authentication

Table 6.2.2 functionality of dart convert dependency

6.2.3 Dart Developer

The dart:developer dependency provides tools for logging and debugging within Dart applications. In this file, it is used to record real-time events such as API calls, tx statuses, and error messages during the polling and querying process, helping developers monitor and troubleshoot payment-related operations efficiently.

Component	Description	Purpose in query_transaction.dart
import 'dart:developer';	Imports Dart's developer tools	Allows the use of logging utilities for debugging purposes
log()	Built-in logging function from dart:developer	Prints diagnostic messages, tx status updates, and API responses to the console
log("⚠️ Polling error: $e");	Logs caught exceptions during polling	Helps identify and troubleshoot polling or connection issues
log("✅ Query Response: $data");	Logs successful responses from the API	Confirms valid communication and data retrieval from &Pay
`log("🌐 Polling txId: $txId	Status: $status");`	Displays tx progress during polling
log("⚠️ Max polling attempts reached...");	Tracks timeout conditions	Provides a debug alert when polling exceeds its retry limit

Table 6.2.3 functionality of dart developer dependency

6.2.4 Crypto

The package:crypto/crypto.dart dependency provides cryptographic hashing functions like SHA-512, which are essential for ensuring data integrity and secure communication between the Flutter app and the &Pay API. In this file, it is used to generate a secure signature that authenticates API requests.

Component	**Description**	**Purpose in query_transaction.dart**
import 'package:crypto/ crypto.dart';	Imports the Dart crypto package	Grants access to industry-standard hashing algorithms such as SHA-512 for secure signing
utf8.encode(jsonBod y + integrationKey)	Converts the combined JSON payload and integration key into bytes	Prepares raw data for hashing; ensures encoding consistency
sha512.convert(...)	Applies the SHA-512 algorithm to the encoded data	Generates a unique, irreversible 512-bit hash to verify authenticity
toString()	Converts the hash output into a human-readable hexadecimal format	Produces a clean signature string usable in API headers
final signature = ...	Stores the computed hash value	Serves as the digital signature required by the &Pay API
"signature": signature (inside HTTP headers)	Attaches the generated signature to each outgoing POST request	Allows the &Pay server to verify the request's origin and integrity

Table 6.2.4 functionality of crypto dependency

6.2.5 HTTP

The package:http/http.dart dependency provides the tools for making HTTP requests to external APIs. In query_transaction.dart, it is responsible for sending signed POST requests to the &Pay endpoint, receiving responses, and handling communication errors during tx status queries.

Component	**Description**	**Purpose in query_transaction.dart**
Import 'package: http/http.dart' as http;	Imports the HTTP client library	Enables sending and receiving HTTP requests and responses
http.post()	Sends an HTTP POST request to a specified URL	Used to communicate with the &Pay API to query tx statuses
Uri.parse(queryUrl)	Converts a URL string into a URI object	Ensures the URL is properly formatted for the HTTP request
headers: { ... }	Defines metadata like content type, charset, and signature	Ensures the request is securely formatted and authenticated
body: jsonBody	Contains the encoded JSON payload sent to the server	Sends merchant and tx data for status verification
response.statusCode	Stores the HTTP response code (e.g., 200, 400, 500)	Determines if the API request succeeded or failed
response.body	Contains the response content from the server	Used to decode and process the tx status data
jsonDecode (response.body)	Parses the JSON response into a Dart object	Converts API response data into a usable format for further processing
log("✅ Query Response: $data");	Logs successful HTTP responses for debugging	Helps monitor communication results and API reliability

Table 6.2.5 functionality of http dependency

6.2.6 FireStore Cloud

The package:cloud_firestore/cloud_firestore.dart dependency connects the Flutter app to Google Cloud Firestore, allowing it to read, write, and update real-time data. In query_transaction.dart, it is used to automatically update the order status in the Firestore database once a payment is confirmed by the &Pay API.

Component	Description	Purpose in query_transaction.dart
import 'package:cloud_firestore/ cloud_firestore.dart';	Imports Firestore functionality	Enables communication with the Firestore database
FirebaseFirestore.instance	Provides the singleton Firestore instance	Grants access to the app's Firestore database for read/write operations
.collection('orders')	Refers to the orders collection in Firestore	Specifies where order documents are stored
.doc(orderId)	References a specific document by its ID	Identifies which order record to update after payment verification
.update({...})	Updates fields in an existing Firestore document	Automatically marks an order as "paid" and records a timestamp when payment is confirmed
"status": "paid"	Field value written to Firestore	Reflects the successful payment completion in the order record
"updatedAt": DateTime.now()	Writes the current timestamp	Logs when the order status was last updated for tracking and auditing
await FirebaseFirestore.instance. collection(...).update(...)	Executes the update asynchronously	Ensures real-time synchronization between the SDK and Firestore

Table 6.2.6 functionality of firestore dependency

6.3 How Flutter Use This Query_transaction.dart File

The Flutter application integrates the query_transaction.dart file to handle payment status verification after a tx is initiated. This file functions as the backend service layer that communicates with the &Pay API to check the real-time status of a tx and update the Firestore database accordingly.

Step	Action	Explanation
1	**Payment initiated in Flutter app**	When the user completes payment in the in-app WebView, the Flutter app triggers the verification process.
2	**App calls AmpersandPayQuery. pollTransaction()**	This method starts a background polling loop that periodically queries the &Pay server for the tx status.
3	**queryTransaction() sends HTTP request**	The method builds a JSON payload with the tx ID and merchant credentials, signs it with SHA-512, and sends it to the &Pay API.
4	&Pay **responds with status data**	The API returns a response containing fields such as txStatus and ret, which indicate whether the payment was successful, pending, or failed.
5	**App updates Firestore automatically**	Once a successful status (SUCCESS or PAID) is detected, the function updates the order record in the Firestore orders collection.
6	**Real-time feedback to user**	The Flutter app reflects the updated status on the checkout or confirmation screen, allowing users to see their payment result instantly.

Table 6.3 steps on query_transaction.dart file

7 FINISH_CHECKOUT.DART

7.1 Functionalities

The finish_checkout.dart file functions as the final order confirmation and data synchronization module within the Flutter payment workflow. It executes cleanup and post-payment operations after a tx attempt, ensuring that order data, user carts, and Firestore records remain consistent. Depending on payment success, it either deletes failed orders or archives successful ones into the paidProducts collection for record-keeping.

No.	Functionality	Description
1	**Order Retrieval from Firestore**	Fetches the order document from the orders collection using the provided orderId to display order and customer details.
2	**Failed Payment Handling**	Deletes the Firestore order record if the payment was unsuccessful, preventing incomplete tx from being stored.
3	**Successful Payment Processing**	Transfers each purchased product from the order into the paidProducts collection, recording tx ID, user details, and timestamp.
4	**Cart Cleanup**	Removes successfully purchased products from the user's cart subcollection in Firestore to maintain an accurate cart state.
5	**Payment Method Conversion**	Converts raw payment codes (e.g., "EW", "CC", "DD") into readable names like *E-Wallet*, *Credit Card*, or *Online Banking*.
6	**User Feedback and Navigation**	Displays a detailed receipt showing payment result, customer info, and order summary, and provides a "Return to Home" button for navigation.
7	**Error Handling and Logging**	Uses try...catch and debugPrint() to handle Firestore access errors, preventing crashes during data loading or deletion.

Table 7.1 functionalities of finish_checkout.dart file

7.2 Dependencies Used

These dependencies are similarly used in other files.

No.	Dependency	Description / Purpose
1	**package:flutter/material.dart**	Provides Flutter's core Material Design widgets and layout components used to build the UI for the payment receipt screen (e.g., Scaffold, AppBar, Text, ElevatedButton).
2	**package:cloud_firestore/cloud_firestore.dart**	Connects the app to **Google Cloud Firestore**, allowing it to retrieve order data, delete failed orders, move successful orders to the paidProducts collection, and clear purchased items from the user's cart.

Table 7.2 dependencies used in finish_checkout.dart

7.3 How Flutter Use This Finish_checkout.dart File

The finish_checkout.dart file is executed after a payment process concludes, either successfully or unsuccessfully. Flutter uses this file as the final stage of the checkout workflow, handling both visual feedback to the user and backend updates in Firestore.

When the payment process (from payment_webview.dart) finishes, this page is opened through a Navigator route transition, passing the required parameters such as orderId, txId, success, and userId.

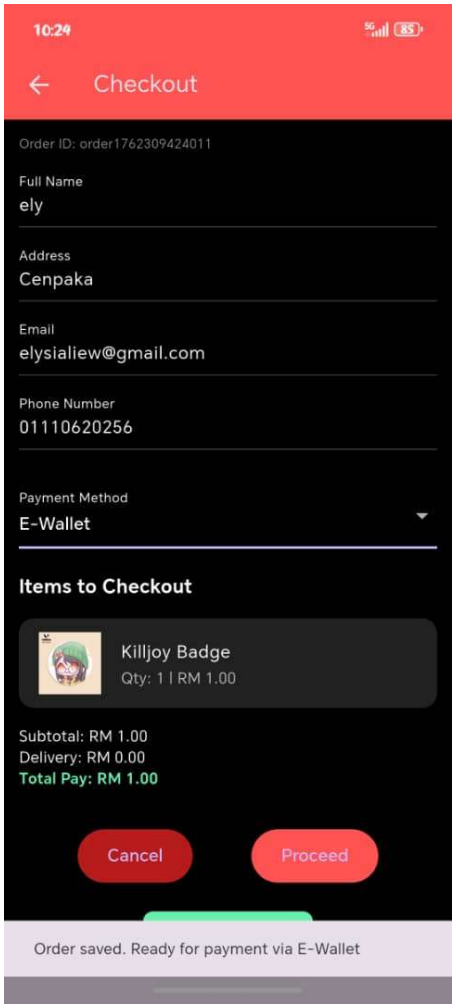
Step	Action	Explanation
1	**FinishCheckoutPage is opened after payment result**	When the user returns from the payment WebView (successful or failed), the app navigates to the FinishCheckoutPage, passing along the orderId, txId, success, and userId.
2	**_loadOrder() runs automatically on initState()**	Upon page initialization, _loadOrder() retrieves the corresponding order document from the Firestore orders collection using the provided orderId.

3	**Checks payment result (success/failure)**	If success is false, the function deletes the order document to remove failed tx. If true, the order details are stored in memory for display.
4	**Displays receipt details to user**	The UI shows whether the tx succeeded or failed, including order details, user info, payment method, and itemized list of purchased products.
5	**User clicks “Return to Home”**	When tapped, _handleHome() runs — if the payment succeeded, it calls _savePaidProducts() before navigating back to the home page.
6	**_savePaidProducts() stores completed purchases**	This method saves each purchased product as a new document in the Firestore paidProducts collection, tagging them with order and tx info.
7	**_clearPurchasedItemsFromCart() cleans up user cart**	After recording purchases, it removes the bought items from the user's Firestore cart collection to prevent duplicates.
8	**Navigation completes**	Finally, Flutter pops all intermediate pages until reaching the home screen, completing the checkout process.

Table 7.3 steps on finish_checkout.dart file

8 Outputs

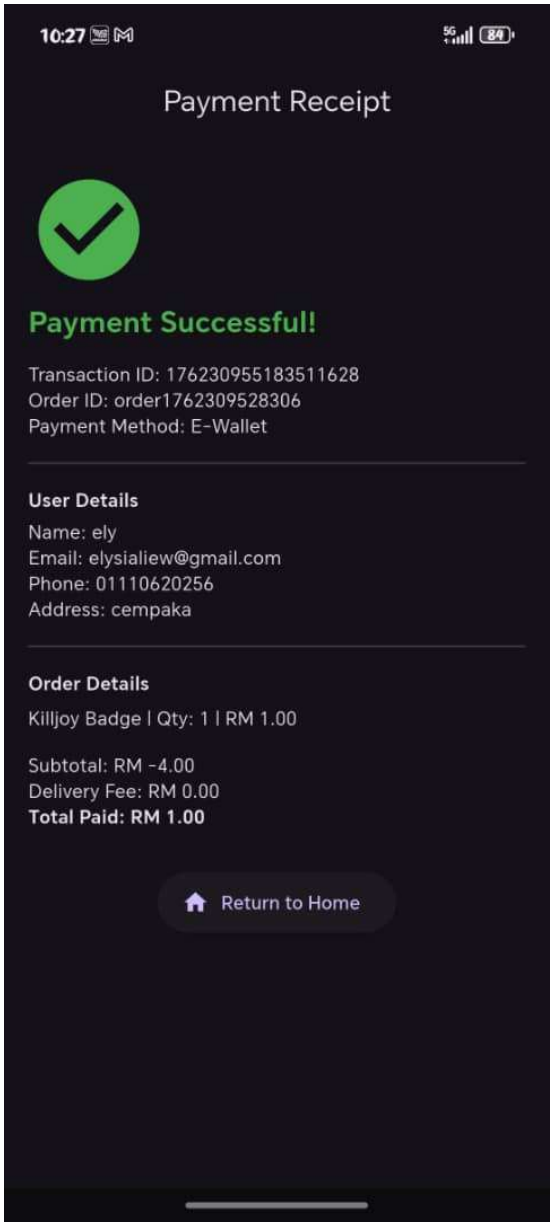
When details are saved successfully in the order_checkout.dart page. It will show a successful message at the bottom. Then user can proceed to pay.



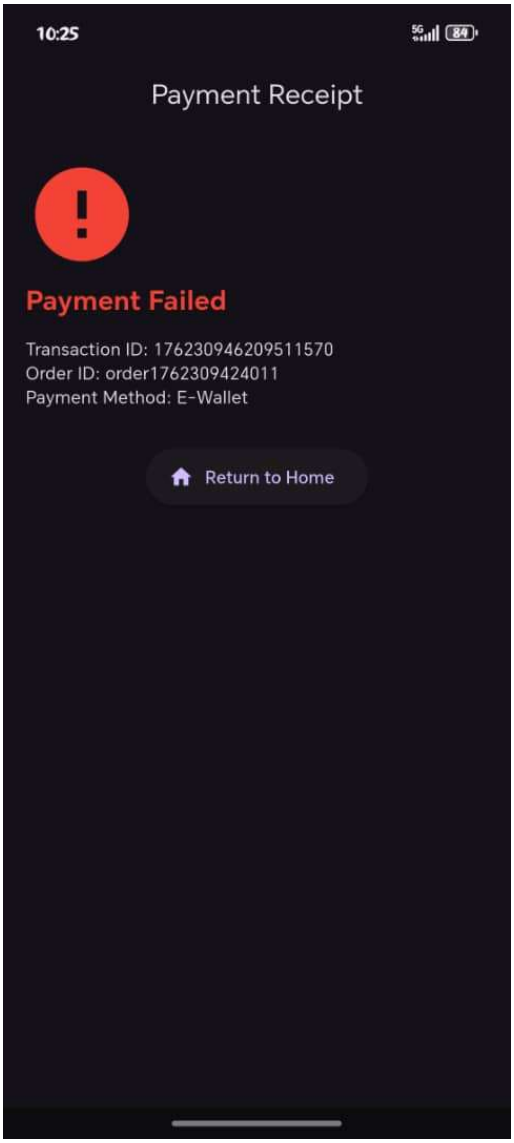
OTHER ERROR MESSAGE WILL PROMPT IN THIS PAGE WHEN YOU CLICK THE PAY BUTTON.

in this case there are no errors, hence no error message will appear when u click pay

When payment is made, and the query_transaction.dart poll that they payment is SUCCESSFUL and update database to successful. It will then redirect user to finish_checkout.dart page and prompt a successful payment receipt.



Hence otherwise, when payment is not made and user click back and the query_transaction.dart poll did not reach successful. It will then redirect user to finish_checkout.dart page and prompt a failed payment receipt.



< END OF DOCUMENT >