

VERSION 1.0
DATE – 08/07/2025



PRESENTED BY: AIRPAY PAYMENT SERVICES

AIRPAY PAYMENT SERVICES PVT LTD

EMPRESSA 14, WESTERN EXPRESS HIGHWAY, SAHAR AIRPORT RD, ANDHERI EAST, MUMBAI

Ionic Capacitor Plugin

ABSTRACT

This document specifies the technical aspect of integrating Ionic Capacitor Plugin.

DISCLAIMER

This documentation shall only be used for evaluating the planned services designated herein and may contain information that is privileged, confidential, Proprietary, or otherwise protected from disclosure. As a result, this document or content thereof shall not be disclosed, used, or duplicated, in whole or in part, for any purpose other than the Scope of Work assigned by airpay Payment Services to your company ("Recipient"). Upon completion of service or termination of service, the Recipient shall return all materials, including, without limiting the generality of the foregoing, all originals, copies, reproductions, and summaries of confidential information. Any unauthorized use or disclosure by the directors, officers, or employees of the Recipient shall be deemed to be unauthorized use or disclosure by the Recipient and the Recipient shall indemnify and hold harmless the airpay Payment Services from and against all damages, losses, costs, and expenses incurred because of such breach. airpay payment services may seek injunctive relief restraining the unauthorized disclosure or use of confidential information in addition to any other legal or equitable remedy otherwise available.

VERSION HISTORY

VERSION #	IMPLEMENTED BY	REVISION DATE	APPROVED BY	APPROVAL DATE	REASONS
1.0	Tushar Khandekar	[08/07/2025]	Kunal Shah	[08/07/2025]	First version Document

Table of contents

ABSTRACT.....	2
DISCLAIMER.....	3
VERSION HISTORY	4
AIRPAY IONIC CAPACITOR PLUGIN.....	6
SUPPORTED PLATFORMS	6
REQUIREMENTS	6
INSTALLATION	6
Install Ionic CLI and Capacitor CLI	6
CREATE a New Ionic Project.....	6
ENABLE Capacitor and Add Android Platform	7
ENABLE Capacitor and Add iOS Platform.....	7
INSTALL the Airpay Plugin - (Ionic Capacitor v8).....	7
ANDROID Integration	7
ANDROIDMANIFEST.xml	7
MAINACTIVITY.java	7
build.gradle (app-level)	17
build.gradle (project-level)	17
gradle.properties	18
AirpayPlugin.java	18
Angular Code - home.page.ts	18
iOS Integration	18
Adding Framework.....	18
Info.plist - Adding below code	19
AirpayDemoViewModel.swift file changes -.....	19
Response handling will be managed by the finishPayment() method –	21
AppDelegate.swift file changes -	22
HELP	25
SUPPORT	25
VERSION HISTORY	25
LICENSE	25

AIRPAY IONIC CAPACITOR PLUGIN

This is the official Ionic Capacitor plugin for integrating the Airpay payment gateway into Ionic applications.

SUPPORTED PLATFORMS

- Android (version compatibility details below)
Up to Gradle version 8.7
- iOS
Xcode 16.2_swiftUI

REQUIREMENTS

- Node.js Version: 22.11.0
- Ionic Version: 8.5.0
- Ionic CLI: 7.2.0
- Capacitor CLI: 7.1.0
- @capacitor/android: 7.1.0
- @capacitor/core: 7.1.0
- npm: 8.19.4

INSTALLATION

Note: For Windows users, please run the following commands in Git Bash instead of Command Prompt. You can download Git for Windows [here](#).

Install Ionic CLI and Capacitor CLI

```
npm install -g @ionic/cli  
npm install -g @capacitor/cli
```

CREATE a New Ionic Project

```
ionic start ionic_sample_app blank --type=angular  
cd ionic_sample_app/
```

ENABLE Capacitor and Add Android Platform

```
ionic integrations enable capacitor  
npm install @capacitor/android  
npx cap add android  
npx cap sync android
```

ENABLE Capacitor and Add iOS Platform

```
npm install @capacitor/ios  
npx cap add ios  
npx cap sync ios
```

INSTALL the Airpay Plugin - (Ionic Capacitor v8)

```
npm install https://github.com/Airpay2014/airpay-capacitor-V4-India.git  
ionic build  
ionic cap sync android  
ionic cap sync ios
```

ANDROID Integration

The following Android files are crucial for integrating Airpay with Capacitor:

ANDROIDMANIFEST.xml

Add the following inside the <application> tag:

```
<uses-library  
  android:name="org.apache.http.legacy"  
  android:required="false" />
```

Ensure internet permission is granted:

```
<uses-permission android:name="android.permission.INTERNET" />
```

MAINACTIVITY.java

Modify MainActivity.java to include Airpay imports:

```
import com.airpay.airpaysdk_simplifiedotp.AirpayConfig;  
import com.airpay.airpaysdk_simplifiedotp.constants.ConfigConstants;  
import com.airpay.airpaysdk_simplifiedotp.utils.Transaction;  
import com.airpay.airpaysdk_simplifiedotp.utils.Utils;  
import com.airpay.airpaysdk_simplifiedotp.view.ActionResultListener;  
import com.airpay.plugins.mycustomplugin.AirpayPlugin;
```

```
import com.getcapacitor.BridgeActivity;
import com.getcapacitor.JSObject;

import android.annotation.SuppressLint;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Bundle;
import android.os.Handler;
import android.text.TextUtils;
import android.util.Log;
import android.view.WindowManager;
import android.widget.Toast;

import androidx.activity.result.ActivityResultLauncher;
import androidx.activity.result.contract.ActivityResultContracts;
import androidx.appcompat.app.AppCompatActivity;
import androidx.localbroadcastmanager.content.LocalBroadcastManager;

import org.json.JSONObject;

import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.zip.CRC32;

public class MainActivity extends BridgeActivity implements ActionListener {
    public ActivityResultLauncher<Intent> airpayLauncher;

    private boolean doubleBackToExitPressedOnce = false;

    private String sAllSubscriptionData;
    private BroadcastReceiver receiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            if (AirpayPlugin.LOCAL_AIRPAY_BROADCAST_EVENT.equals(intent.getAction())) {
                if (intent != null) {
                    // if (intent.getStringExtra("flag").equalsIgnoreCase("y")) {

                        String requestData = intent.getStringExtra("request_data");
                        Log.d("requestData", requestData);
                        try {
                            // Convert jsonString to JSONObject
                            JSONObject outerJson = new JSONObject(requestData);
                            // Get the "value" field which contains the actual JSON string
                            String innerJsonString = outerJson.getString("value");
                            // Replace escaped quotes \" with actual quotes "

```



```

innerJsonString = innerJsonString.replace("\\\\\"", "\\");
// Convert to JSONObject
JSONObject innerJson = new JSONObject(innerJsonString);

// Extract values
String firstName = innerJson.getString("firstName");
String lastName = innerJson.getString("lastName");
String email = innerJson.isNull("email") ? "" : innerJson.getString("email");
String address = innerJson.isNull("fullAddress") ? "" : innerJson.getString("fullAddress");
String city = innerJson.isNull("city") ? "" : innerJson.getString("city");
String state = innerJson.isNull("state") ? "" : innerJson.getString("state");
String phone = innerJson.isNull("phone") ? "" : innerJson.getString("phone");
String country = innerJson.isNull("country") ? "" : innerJson.getString("country");
String pincode = innerJson.isNull("pincode") ? "" : innerJson.getString("pincode");
String orderId = innerJson.getString("orderId");
String amount = innerJson.getString("amount");
String txnSubtype = innerJson.getString("txnSubtype");
String nextRunDate = innerJson.getString("nextRunDate");
String period = innerJson.getString("period");
String frequency = innerJson.getString("frequency");
String maxAmount = innerJson.getString("maxAmount");
String subscriptionAmount = innerJson.getString("subscriptionAmount");
String recurringCount = innerJson.getString("recurringCount");
String retryAttempts = innerJson.getString("retryAttempts");

String sAllData1;
DateFormat df1 = new SimpleDateFormat("yyyy-MM-dd");
String sCurDate1 = df1.format(new Date());

if (!TextUtils.isEmpty(txnSubtype) && txnSubtype.equals("12")) {

    if ("A".equalsIgnoreCase(period)) {
        sAllSubscriptionData = period +
            appendDecimal(subscriptionAmount) +
            "1" +
            retryAttempts;
    } else {
        sAllSubscriptionData = nextRunDate +
            frequency +
            period +
            appendDecimal(subscriptionAmount) +
            "1" +
            recurringCount +
            retryAttempts;
    }

    sAllData1 = email + firstName

```

```
+ lastName + address
+ city + state
+ country + appendDecimal(amount)
+ orderId + sAllSubscriptionData + sCurDate1;

} else {
  sAllData1 = email + firstName
    + lastName + address
    + city + state
    + country + appendDecimal(amount)
    + orderId + sCurDate1;
}

// Merchant details
Log.d("sAllData1", "" + sAllData1);

// Merchant details
String sMid = ""; //please enter merchant id
String sSecret = ""; //please enter secret key
String sUserName = ""; //please enter username
String sPassword = ""; //please enter password

String merdom = ""; //please enter merdom
String successUrl = ""; //please enter successUrl
String failureUrl = ""; //please enter failedUrl

String client_id = ""; //please enter client id
String client_secret = ""; //please enter client secret
// private key
String sTemp = sSecret + "@" + sUserName + ":@" + sPassword;
String sPrivateKey = Utils.sha256(sTemp);

// key for checksum
String sTemp3 = sUserName + "~::~" + sPassword;
String sKey1 = Utils.sha256(sTemp3);

// checksum
sAllData1 = sKey1 + "@" + sAllData1;

String sChecksum1 = Utils.sha256(sAllData1);
Log.d("Checksum", sChecksum1);

AirpayConfig.Builder builder = new AirpayConfig.Builder(MainActivity.this, airpayLauncher);
builder.setEnvironment(ConfigConstants.PRODUCTION);
builder.setType(ConfigConstants.AIRPAY_KIT);
builder.setPrivateKey(sPrivateKey);
```

```

        builder.setMerchantId(sMid);
        builder.setOrderId(orderId);
        builder.setCurrency("356");
        builder.setIsoCurrency("INR");
        builder.setEmailId(email);
        builder.setMobileNo(phone);
        builder.setBuyerFirstName(firstName);
        builder.setBuyerLastName(lastName);
        builder.setBuyerAddress(address);
        builder.setBuyerCity(city);
        builder.setBuyerState(state);
        builder.setBuyerCountry(country);
        builder.setBuyerPinCode(pincod);
        builder.setAmount(appendDecimal(amount));
        builder.setWallet("0");
        builder.setCustomVar("");
        builder.setTxnSubType(txnSubtype);
        builder.setChmod("");
        builder.setChecksum(sChecksum1);
        builder.setMerDom(merdom);
        builder.setSuccessUrl(successUrl);
        builder.setFailedUrl(failureUrl);
        builder.setLanguage("EN");
        builder.setClient_id(client_id);
        builder.setClient_secret(client_secret);
        builder.setGrant_type("client_credentials");
        builder.setAesDesKey(sTemp3);
        if (!TextUtils.isEmpty(txnSubtype) && txnSubtype.equals("12")) {
            builder.setNextrundate(nextRunDate);
            builder.setPeriod(period);
            builder.setFrequency(frequency);
            builder.setMaxAmount(appendDecimal(maxAmount));
            builder.setSubscriptionAmt(appendDecimal(subscriptionAmount));
            builder.setIsRecurring("1");
            builder.setRecurringCount(recurringCount);
            builder.setRetryAttempts(retryAttempts);
            builder.build().initiatePayment();
        } else {
            builder.build().initiatePayment();
        }

    } catch (Exception e) {
        e.printStackTrace();
    }
}
}
}

```

```

};

@Override
public void onCreate(Bundle savedInstanceState) {

    registerPlugin(AirpayPlugin.class);

    super.onCreate(savedInstanceState);
    // pluginCall = new AirpayPlugin();
    AppCompatDelegate.setDefaultNightMode(AppCompatDelegate.MODE_NIGHT_NO);

    airpayLauncher = registerForActivityResult(
        new ActivityResultContracts.StartActivityForResult(),
        result -> {
            if (result.getResultCode() == RESULT_OK) {
                Intent data = result.getData();
                if (data != null) {
                    Transaction transaction = (Transaction) data.getSerializableExtra("response");
                    if (transaction != null) {
                        Log.d("MyCustomPlugin", "Transaction status: " + transaction.getStatus());
                        Toast.makeText(this, "Transaction Status: " + transaction.getStatus(),
                            Toast.LENGTH_SHORT).show();

                        JSONObject response = new JSONObject();
                        response.put("STATUS", transaction.getStatus());
                        response.put("STATUSMSG", transaction.getStatusMsg());
                        response.put("TXN_MODE", transaction.getTXN_MODE());
                        response.put("TRANSACTIONID", transaction.getTransactionID());
                        response.put("TRANSACTIONAMT", transaction.getTransactionAmt());
                        response.put("TRANSACTIONSTATUS", transaction.getTransactionStatus());
                        response.put("MERCHANTTRANSACTIONID", transaction.getMERCHANTTRANSACTIONID());
                        response.put("MERCHANTPOSTTYPE", transaction.getMERCHANTPOSTTYPE());
                        response.put("MERCHANTKEY", transaction.getMERCHANTKEY());
                        response.put("SECUREHASH", transaction.getSECUREHASH());
                        response.put("CUSTOMVAR", transaction.getCUSTOMVAR());
                        response.put("TXN_DATE_TIME", transaction.getTXN_DATE_TIME());
                        response.put("TXN_CURRENCY_CODE", transaction.getTXN_CURRENCY_CODE());
                        response.put("TRANSACTIONVARIANT", transaction.getTransactionVariant());
                        response.put("CHMOD", transaction.getCHMOD());
                        response.put("BANKNAME", transaction.getBANKNAME());
                        response.put("CARDISSUER", transaction.getCARDISSUER());
                        response.put("FULLNAME", transaction.getFULLNAME());
                        response.put("EMAIL", transaction.getEmail());
                        response.put("CONTACTNO", transaction.getCONTACTNO());
                        response.put("ISRISK", transaction.getISRISK());
                        response.put("MERCHANT_NAME", transaction.getMERCHANT_NAME());
                        response.put("SETTLEMENT_DATE", transaction.getSETTLEMENT_DATE());
                        response.put("SURCHARGE", transaction.getSURCHARGE());
                    }
                }
            }
        }
    );
}

```

```

response.put("BILLEDAMOUNT", transaction.getBILLEDAMOUNT());
response.put("CUSTOMERVPA", transaction.getCUSTOMERVPA());

String transid = transaction.getMERCHANTTRANSACTIONID();
String apTransactionID = transaction.getTransactionID();
String amount = transaction.getTransactionAMT();
String transtatus = transaction.getTransactionSTATUS();
String message = transaction.getStatusMSG();

String customer_vpa = "";
if (!TextUtils.isEmpty(transaction.getCHMOD()) &&
transaction.getCHMOD().equalsIgnoreCase("upi")) {
    customer_vpa = ":" + transaction.getCUSTOMERVPA();
    Log.e("Verified Hash ==", "INSIDE CHMODE UPI CONSIDTION");
}

String merchantid = ""; //Please enter Merchant Id
String username = ""; //Please enter Username
String sParam = transid + ":" + apTransactionID + ":" + amount + ":" + transtatus + ":" + message
+ ":" + merchantid + ":" + username + customer_vpa;
CRC32 crc = new CRC32();
crc.update(sParam.getBytes());
String sCRC = "" + crc.getValue();
Log.e("Verified Hash ==", "sParam= " + sParam);
Log.e("Verified Hash ==", "Calculate Hash= " + sCRC);
Log.e("Verified Hash ==", "RESP Secure Hash= " + transaction.getSECUREHASH());

if (sCRC.equalsIgnoreCase(transaction.getSECUREHASH())) {
    Log.e("Verified Hash ==", "SECURE HASH MATCHED");
} else {
    Log.e("Verified Hash ==", "SECURE HASH MIS-MATCHED");
}

AirpayPlugin.resolvePaymentResult(response.toString());

} else {
    Log.e("MyCustomPlugin", "Transaction object is null");
    AirpayPlugin.rejectPayment("Transaction object is null");
}
} else {
    Log.e("MyCustomPlugin", "Intent data is null");
    AirpayPlugin.rejectPayment("Payment failed: No data received");
}
} else {
    Log.e("MyCustomPlugin", "Payment failed, Result Code: " + result.getResultCode());
    AirpayPlugin.rejectPayment("Payment failed, Result Code: " + result.getResultCode());
}
}
}

```

```
);

}

@SuppressLint("MissingSuperCall")
@Override
public void onBackPressed() {
    if (doubleBackToExitPressedOnce) {
        finishAffinity(); // Closes the app
        return;
    }

    this.doubleBackToExitPressedOnce = true;
    Toast.makeText(this, "Press back again to exit", Toast.LENGTH_SHORT).show();

    new Handler().postDelayed(() -> doubleBackToExitPressedOnce = false, 2000);
}

public String appendDecimal(String input) {
    if (input == null || input.isEmpty()) {
        return "0.00"; // Default value for empty input
    }

    // Check if input already has a decimal
    if (!input.contains(".")) {
        return input + ".00"; // Append .00 if no decimal exists
    }

    return input; // Return as-is if it already has a decimal
}

@Override
public void onResume() {
    super.onResume();
    LocalBroadcastManager.getInstance(this)
        .registerReceiver(receiver, new IntentFilter(AirpayPlugin.LOCAL_AIRPAY_BROADCAST_EVENT));
}

@Override
public void onPause() {
    super.onPause();
    LocalBroadcastManager.getInstance(this).unregisterReceiver(receiver);
}

@Override
public void onResult(Object o) {
```

```

if (o instanceof Transaction) {
    Transaction transaction = (Transaction) o;

    Toast.makeText(MainActivity.this, transaction.getTransactionStatus() + "\n" +
transaction.getStatusMsg(), Toast.LENGTH_LONG).show();
    if (transaction.getStatus() != null) {
        Log.e("STATUS -> ", "=" + transaction.getStatus());
    }
    if (transaction.getMerchantKey() != null) {
        Log.e("MERCHANT KEY -> ", "=" + transaction.getMerchantKey());
    }
    if (transaction.getMerchantPostType() != null) {
        Log.e("MERCHANT POST TYPE ", "=" +
transaction.getMerchantPostType());
    }
    if (transaction.getStatusMsg() != null) {
        Log.e("STATUS MSG -> ", "=" + transaction.getStatusMsg()); // success or fail
    }
    if (transaction.getTransactionAmt() != null) {
        Log.e("TRANSACTION AMT -> ", "=" + transaction.getTransactionAmt());
    }
    if (transaction.getTXN_MODE() != null) {
        Log.e("TXN MODE -> ", "=" + transaction.getTXN_MODE());
    }
    if (transaction.getMerchantTransactionID() != null) {
        Log.e("MERCHANT_TXN_ID -> ", "=" + transaction.getMerchantTransactionID()); // order id
    }
    if (transaction.getSecureHash() != null) {
        Log.e("SECURE HASH -> ", "=" + transaction.getSecureHash());
    }
    if (transaction.getCustomVar() != null) {
        Log.e("CUSTOMVAR -> ", "=" + transaction.getCustomVar());
    }
    if (transaction.getTransactionID() != null) {
        Log.e("TXN ID -> ", "=" + transaction.getTransactionID());
    }
    if (transaction.getTransactionStatus() != null) {
        Log.e("TXN STATUS -> ", "=" + transaction.getTransactionStatus());
    }
    if (transaction.getTXN_DATE_TIME() != null) {
        Log.e("TXN_DATETIME -> ", "=" + transaction.getTXN_DATE_TIME());
    }
    if (transaction.getTXN_CURRENCY_CODE() != null) {
        Log.e("TXN_CURRENCY_CODE -> ", "=" + transaction.getTXN_CURRENCY_CODE());
    }
    if (transaction.getTransactionVariant() != null) {
        Log.e("TRANSACTIONVARIANT -> ", "=" + transaction.getTransactionVariant());
    }
}

```

```

    if (transaction.getCHMOD() != null) {
        Log.e("CHMOD -> ", "=" + transaction.getCHMOD());
    }
    if (transaction.getBANKNAME() != null) {
        Log.e("BANKNAME -> ", "=" + transaction.getBANKNAME());
    }
    if (transaction.getCARDISSUER() != null) {
        Log.e("CARDISSUER -> ", "=" + transaction.getCARDISSUER());
    }
    if (transaction.getFULLNAME() != null) {
        Log.e("FULLNAME -> ", "=" + transaction.getFULLNAME());
    }
    if (transaction.getEmail() != null) {
        Log.e("EMAIL -> ", "=" + transaction.getEmail());
    }
    if (transaction.getCONTACTNO() != null) {
        Log.e("CONTACTNO -> ", "=" + transaction.getCONTACTNO());
    }
    if (transaction.getMERCHANT_NAME() != null) {
        Log.e("MERCHANT_NAME -> ", "=" + transaction.getMERCHANT_NAME());
    }
    if (transaction.getSETTLEMENT_DATE() != null) {
        Log.e("SETTLEMENT_DATE -> ", "=" + transaction.getSETTLEMENT_DATE());
    }
    if (transaction.getSURCHARGE() != null) {
        Log.e("SURCHARGE -> ", "=" + transaction.getSURCHARGE());
    }
    if (transaction.getBILLEDAMOUNT() != null) {
        Log.e("BILLEDAMOUNT -> ", "=" + transaction.getBILLEDAMOUNT());
    }
    if (transaction.getISRISK() != null) {
        Log.e("ISRISK -> ", "=" + transaction.getISRISK());
    }
    String transid = transaction.getMERCHANTTRANSACTIONID();
    String apTransactionID = transaction.getTransactionID();
    String amount = transaction.getTransactionAMT();
    String transtatus = transaction.getTransactionSTATUS();
    String message = transaction.getStatusMSG();

    String customer_vpa = "";
    if (!TextUtils.isEmpty(transaction.getCHMOD()) &&
transaction.getCHMOD().equalsIgnoreCase("upi")) {
        customer_vpa = ":" + transaction.getCUSTOMERVPA();
        Log.e("Verified Hash ==", "INSIDE CHMODE UPI CONSIDTION");
    }

    String merchantid = ""; //Please enter Merchant Id
    String username = ""; //Please enter Username

```



```

        String sParam = transid + ":" + apTransactionID + ":" + amount + ":" + transtatus + ":" + message + ":"
+ merchantid + ":" + username + customer_vpa;
        CRC32 crc = new CRC32();
        crc.update(sParam.getBytes());
        String sCRC = "" + crc.getValue();
        Log.e("Verified Hash ==", "sParam= " + sParam);
        Log.e("Verified Hash ==", "Calculate Hash= " + sCRC);
        Log.e("Verified Hash ==", "RESP Secure Hash= " + transaction.getSECUREHASH());

        if (sCRC.equalsIgnoreCase(transaction.getSECUREHASH())) {
            Log.e("Verified Hash ==", "SECURE HASH MATCHED");
        } else {
            Log.e("Verified Hash ==", "SECURE HASH MIS-MATCHED");
        }
    }
}
}
}

```

build.gradle (app-level)

Add the following dependencies:

```

implementation("com.airpay:Airpay-India-Kit-V4:1.0.0") {
    exclude group: 'androidx.core', module: 'core'
    // or
    exclude group: 'androidx.legacy', module: 'legacy-support-v4'
}
implementation 'androidx.localbroadcastmanager:localbroadcastmanager:1.0.0'

```

build.gradle (project-level)

Add the following repository configurations: Note:- For the username and password , please refer the ionic capacitor kit on sanctum link.

```

maven { url 'https://maven.google.com' }
maven {
    url "https://gitlab.com/api/v4/projects/69276629/packages/maven"
    credentials(HttpHeaderCredentials) {
        name = "Private-Token"
        value = "glpat-T1rgMKUrc686pzp_s9yw"
    }
    authentication {
        header(HttpHeaderAuthentication)
    }
}

```

```
}
```

gradle.properties

Add the following properties:

```
android.enableJetifier=true
org.gradle.java.home=C:\\Program Files\\Java\\jdk-21
```

AirpayPlugin.java

Update the following fields with your merchant configuration details:

```
// Merchant details
String sMid = ""; // Enter Merchant ID
String sSecret = ""; // Enter Secret Key
String sUserName = ""; // Enter Username
String sPassword = ""; // Enter Password
String client_id = ""; //please enter client id
String client_secret = ""; //please enter client secret

.setSuccessUrl("") // Enter Success URL
.setFailedUrl("") // Enter Failed URL
.setMerDom("") //Enter Success URL Domain
```

Angular Code - home.page.ts

Example of calling the .open function on button click:

```
await MyCustomPlugin.open({ value: jsonData })
  .then((result) => {
    alert("Return value is " + result.value);
  })
  .catch((error) => {
    alert("Error is " + error);
  });
```

iOS Integration

The following iOS files are crucial for integrating Airpay with a Capacitor:

Adding Framework

Need to change the general setting -(Go to project settings -> general -> Frameworks, libraries, and Embedded Content -> Select the library and set Embed & sign)

Please refer to the Ionic Capacitor sample kit code available on the Sanctum portal for integration, which includes the Airpay Framework. Ensure the following changes are applied to your project.

Info.plist - Adding below code

```
<key>LSApplicationQueriesSchemes</key>
  <array>
    <string>phonepe</string>
    <string>gpay</string>
    <string>bhim</string>
    <string>paytmmp</string>
    <string>amazonToAlipay</string>
    <string>whatsapp</string>
    <string>credpay</string>
    <string>mobikwik</string>
  </array>
```

AirpayDemoViewModel.swift file changes -

Need to configured the Merchant Configuration details inside the AirpayDemoViewModel.swift file -

```
@Published var kAirPaySecretKey: String = "" //(Enter the Secret Key value)
@Published var kAirPayUserName: String = "" //(Enter the Username value)
@Published var kAirPayPassword: String = "" //(Enter the Password value)
@Published var successURL: String = "" //(Enter the Success url value)
@Published var merchantID:String = "" //(Enter the Merchant Id value)
@Published var client_secret:String = "" //(Enter the Client_secret Id value)
@Published var client_id:String = "" //(Enter the Client Id value)
```

Refer the PrivateKey function -

```
func privateKey()->String {
    let sTemp = "\(kAirPaySecretKey)\("@")\kAirPayUserName\)(":|:")\kAirPayPassword)"
    let hashCode1 = "\(sTemp)"
    let sPrivateKey = hashCode1.sha256Hash()
    print("sPrivateKey: \(sPrivateKey)")
    return sPrivateKey
}
```

Refer the Checksum Calculation function inside the AirpayDemoViewModel.swift

```
func getChecksum(privateKey:String, currentDate:String) -> String {
    var siIndexVar = String()

    if subscriptionType != "12" {
        siIndexVar = ""
    }else{
        if subscriptionPeriod == "A" {
            siIndexVar =
            "\(\subscriptionPeriod)\(\subscriptionAmount)\(\subscriptionIsRecurring)\(\subscriptionRetryattempts)"
        }else{
            siIndexVar =
            "\(\subscriptionNextRundate)\(\subscriptionFrequency)\(\subscriptionPeriod)\(\subscriptionAmount)\(\subscriptionIsRecurring)\(\subscriptionRecurringcount)\(\subscriptionRetryattempts)"
        }
    }

    print(siIndexVar)
    let stringAll = "\(\email)\(\firstName)\(\lastName)\(\address)\(\city)\(\state)\(\country)\(\amount)\(\orderId)\(\siIndexVar)\(\currentDate)"

    print("stringAll: \(stringAll)")

    let sTemp2 = "\(\kAirPayUserName)\(~::~)"\(\kAirPayPassword)"
    let hashCode2 = "\(\sTemp2)"
    let sKey = hashCode2.sha256Hash()

    let sAllData = "\(\sKey)@\(\stringAll)"

    print("sAllData: \(sAllData)")

    let checksumStr = "\(\sAllData.sha256Hash())"

    print("sKey: \(sKey)")
    print("checksumStr: \(checksumStr)")
    return checksumStr
}
```

(Validation of fields , checksum , private key, date calculation, encodeDomainToBase64 , sha256Hash all are functions logics mentioned inside the AirpayDemoViewModel.swift class)

Response handling will be managed by the finishPayment() method –

Note :- AirPayDelegate class was extended to the AirpayDemoViewModel.swift class hence we are able to get the finishPayment method on AirpayDemoViewModel.swift class

```
// Delegate method to call when payment finishes
func finishPayment(success: Bool, response: [String: Any]?, error: Error?) {
    // Extract values from the response
    let status = response?["TRANSACTIONSTATUS"] as? String ?? ""
    let chmod = response?["CHMOD"] as? String ?? ""
    var customerVPA = ""

    if chmod == "upi" {
        customerVPA = ":" + (response?["CUSTOMERVPA"] as? String ?? "")
    }

    let apSecureHash = response?["AP_SECUREHASH"] as? String ?? ""
    let transID = response?["MERCHANTTRANSACTIONID"] as? String ?? ""
    let apTransactionID = response?["TRANSACTIONID"] as? String ?? ""
    let transactionAmount = response?["TRANSACTIONAMT"] as? String ?? ""
    let transactionStatus = response?["TRANSACTIONSTATUS"] as? String ?? ""
    let statusMsg = response?["STATUSMSG"] as? String ?? ""

    let strParam = "\(transID)" + ":" + "\(apTransactionID)" + ":" + "\(transactionAmount)" + ":" +
    "\(transactionStatus)" + ":" + "\(statusMsg)" + ":" + "\(merchantID)" + ":" +
    "\(kAirPayUserName)\(customerVPA)"

    let crc32Str = strParam.data(using: .utf8)
    let calculatedHash = crc32Str?.withUnsafeBytes {
        crc32(0, $0.bindMemory(to: Bytef.self).baseAddress, numericCast(crc32Str?.count ?? 0))
    }

    let sCRC = "\(calculatedHash ?? 0)"
    print("Calculated Hash: \(sCRC)")
    print("AP Secure Hash: \(apSecureHash)")

    var upiStatus = ""
    if sCRC == apSecureHash {
        print("Secure hash matched")
        upiStatus = "SECURE HASH MATCHED"
    } else {
        print("Secure hash mismatch")
        upiStatus = "SECURE HASH MIS-MATCHED"
    }

    // Update SwiftUI state variables
    DispatchQueue.main.async {
        self.isPresentingWebView = false
    }
}
```

```

    if success {
        self.alertTitle = status
        self.alertMessage = "\\(String(describing: response)) \\(upiStatus)"
    } else {
        self.alertTitle = "status"
        self.alertMessage = "\\(String(describing: response)) \\(upiStatus)"
    }
}
showTransactionAlert = true
}

```

Note:- Securehash logic was mentioned inside the finishPayment method. Calculated securehash will be matched with securehash getting from the transaction response, By matching the securehash value we can validate the transaction response getting from the server.

If the securehash is mismatched then the reponse which received from server is improper for the requested transaction id.

AppDelegate.swift file changes -

In this class, handleAirPayNotification method contains the logic to send the request parameters to the framework and handling the response to the ionic app side using notification centre.

Code logic -

```

import UIKit
import SwiftUI
import AirpayKitPlugin
import Capacitor
import Airpay_Kit_Swiftui

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate, AirPayDelegate {

    var window: UIWindow?
    var hasAlreadyHandledPayment = false
    var paymentModel = AirpayDemoViewModel.shared

    // MARK: - AirPay Callback
    func finishPayment(success: Bool, response: [String: Any]?, error: (any Error)?) {
        guard !hasAlreadyHandledPayment else { return }
        hasAlreadyHandledPayment = true

        var resultDict: [String: Any] = ["success": success]
        if let response = response {
            resultDict["response"] = response
        }
        if let error = error {

```

```

        resultDict["error"] = error.localizedDescription
    }

    DispatchQueue.main.async {
        self.window?.rootViewController?.dismiss(animated: true, completion: {
            NotificationCenter.default.post(
                name: Notification.Name("PaymentResponse"),
                object: nil,
                userInfo: resultDict
            )
            print("PaymentResponse notification posted")

            DispatchQueue.main.asyncAfter(deadline: .now() + 1) {
                self.hasAlreadyHandledPayment = false
            }
        })
    }
}

// MARK: - App Launch
func application(
    _ application: UIApplication,
    didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?
) -> Bool {
    let bridge = CAPBridgeViewController()
    bridge.bridge?.registerPluginType(AirpayPlugin.self)

    NotificationCenter.default.addObserver(
        self,
        selector: #selector(handleAirPayNotification(_:)),
        name: Notification.Name("AirPayCall"),
        object: nil
    )

    return true
}

// MARK: - Notification Handler
@objc func handleAirPayNotification(_ notification: Notification) {
    guard let userInfo = notification.userInfo as? [String: Any] else {
        print("No userInfo found in notification")
        return
    }

    let dateFormat = DateFormatter()
    dateFormat.dateFormat = "yyyy-MM-dd"
    let sCurrentDate = dateFormat.string(from: Date())

```

```

// Extract fields
let email = userInfo["email"] as? String ?? ""
let phoneNumber = userInfo["phone"] as? String ?? ""
let orderID = userInfo["orderId"] as? String ?? ""
let amount = "\\(userInfo["amount"] ?? 0)"
let firstName = userInfo["firstName"] as? String ?? ""
let lastName = userInfo["lastName"] as? String ?? ""
let address = userInfo["fullAddress"] as? String ?? ""
let city = userInfo["city"] as? String ?? ""
let state = userInfo["state"] as? String ?? ""
let country = userInfo["country"] as? String ?? ""
let pincode = userInfo["pincode"] as? String ?? ""
let subscriptionType = userInfo["txnSubtype"] as? String ?? ""
let subscriptionNextRunDate = userInfo["nextRunDate"] as? String ?? ""
let subscriptionPeriod = userInfo["period"] as? String ?? ""
let subscriptionFrequency = userInfo["frequency"] as? String ?? ""
let subscriptionAmount = userInfo["subscriptionAmount"] as? String ?? ""
let subscriptionIsRecurring = userInfo["pincode"] as? String ?? ""
let subscriptionRecurringCount = userInfo["recurringCount"] as? String ?? ""
let subscriptionRetryAttempts = userInfo["retryAttempts"] as? String ?? ""
let subscriptionMaxAmount = userInfo["maxAmount"] as? String ?? ""

let checksumStr = self.paymentModel.getChecksum(privateKey: self.paymentModel.privateKey(),
currentDate: self.paymentModel.DateCalculation())

// ✅ All UIKit / SwiftUI view & model creation MUST be on main thread
DispatchQueue.main.async {
    let airpayViewModel = AirPayWebViewModel(
        envConfigString: "production",
        email: email,
        phoneNumber: phoneNumber,
        orderID: orderID,
        amount: amount,
        secretAPIKey: self.paymentModel.kAirPaySecretKey,
        successURL: self.paymentModel.successURL,
        userName: self.paymentModel.kAirPayUserName,
        password: self.paymentModel.kAirPayPassword,
        privateKey: self.paymentModel.privateKey(),
        checksumStr: checksumStr,
        firstName: firstName,
        lastName: lastName,
        address: address,
        city: city,
        state: state,
        country: country,
        pincode: pincode,
        mode: self.paymentModel.mode,
        merchantID: self.paymentModel.merchantID,

```



```

        customVariable: self.paymentModel.customVariable,
        transactionSubType: self.paymentModel.transactionSubType,
        currencyValue: self.paymentModel.currencyValue,
        isoCurrency: self.paymentModel.isoCurrency,
        wallet: self.paymentModel.wallet,
        token: self.paymentModel.token,
        delegate: self,
        subscriptionType: subscriptionType,
        subscriptionNextRundate: subscriptionNextRundate,
        subscriptionPeriod: subscriptionPeriod,
        subscriptionFrequency: subscriptionFrequency,
        subscriptionAmount: subscriptionAmount,
        subscriptionIsRecurring: "1",
        subscriptionRecurringcount: subscriptionRecurringcount,
        subscriptionRetryattempts: subscriptionRetryattempts,
        subscriptionMaxamount: subscriptionMaxamount,
        client_id: self.paymentModel.client_id,
        client_secret: self.paymentModel.client_secret,
        merDom: self.paymentModel.encodeDomainToBase64(from: self.paymentModel.successURL)
    )

    let airpayWebView = AirPayWebViewContainer(viewModel: airpayViewModel)
    let hostingController = UIHostingController(rootView: airpayWebView)
    hostingController.modalPresentationStyle = .fullScreen
    self.window?.rootViewController?.present(hostingController, animated: true)
}
}

```

Note: For merchant configuration details, please contact the Airpay Support Team.

HELP

For troubleshooting, use:

- npx cap doctor

SUPPORT

For assistance, please contact:

- Technical/Integration Support Team
- Customer Support Team

VERSION HISTORY

- Initial Release

LICENSE

This project is licensed under the [Airpay Payment Service] License.