# Waivpay React Native SDK

[Waivpay React Native SDK 1](#_Toc109894643)

[Version History 1](#_Toc109894644)

[Introduction 3](#_Toc109894645)

[Architecture 3](#_Toc109894646)

[SDK Access 3](#_Toc109894647)

[Apple configuration 4](#_Toc109894648)

[Capabilities 4](#_Toc109894649)

[Additional Capabilities 5](#_Toc109894650)

[App Name, Identifier & Bundle ID 5](#_Toc109894651)

[Provisioning Profile 6](#_Toc109894652)

[Certificates 8](#_Toc109894653)

[Stripe Apple Pay Integration 9](#_Toc109894654)

[Google configuration 12](#_Toc109894655)

[App Identifier 12](#_Toc109894656)

[SHA256 Fingerprint 12](#_Toc109894657)

[Installation 13](#_Toc109894658)

[Module Dependencies 14](#_Toc109894659)

[APIs 15](#_Toc109894660)

[Configuration 15](#_Toc109894661)

[Data Models 22](#_Toc109894662)

[Marketing Requirements 26](#_Toc109894663)

# Version History

|  |  |  |
| --- | --- | --- |
| 0.1 | Initial draft | Kelly Lamborn |
| 0.2 | Reviewed | Glenn Murray |
| 0.3 | Marketing Requirements | Glenn Murray |
| 0.4 | Addition of cashback models and apis, Typescript, Yarn installation, more information for google and changes are per certification | Kelly Lamborn |
| 0.5 | Fix the constructor documentation, add the npm repository access notes | Tathagata Chowdhury / Kelly Lamborn |
| 0.2.5 | getCardDetails | Tathagata Chowdhury |
| 0.2.6  0.2.7 | Apple Pay inclusion on how to setup, internal changes for certification, change error handling to show all details | Tathagata Chowdhury / Kelly Lamborn |
| 0.2.8 | getCardDetails to return recipient\_first\_name and recipient\_last\_name  Changed the way you import the google SDK  Added flow to the documentation  Added information about the call back fields | Kelly Lamborn |
| 1.0 | Handover version | Glenn Murray |
| 1.1 | SDK Access updates + rebranding | Matt Hillman |

# 

# Introduction

The Waivpay SDK provides a simple interface to allow an application to be built with access to Waivpay Web Services which provision digital cards. The main features provided by the SDK include:

* Information about the brands.
* Digital and physical card information.
* Purchasing a gift card.
* Adding a card to a digital wallet.
* Retrieving balance of a card.
* Retrieving transaction history of a card.
* Access to promotions and claims.
* User management.
* Promotions and cash back functionality.

# Architecture

The Waivpay SDK is written for a React Native application and provides simple JavaScript / TypeScript methods and classes for implementation. It also provides a native wrapper that uses iOS and Android methods to add a card to a digital wallet. The card partner Karta provides the digital cards and resources for the wallet.

Diagram

Description automatically generated

# SDK Access

You will need to be setup with access to the Waivpay APIs to use the SDK. Waivpay will provide a client id”, “client secret and application id”, that will be used within the app.

The SDK is hosted on GitHub Packages in the NPM registry. You will need to use a personal access token for authentication and access to the package. More information on this can be found at: https://docs.github.com/en/packages/working-with-a-github-packages-registry/working-with-the-npm-registry

If you are using this SDK to purchase gift cards or collect money via the device wallet”, “you will also need to configure your apple and google applications for the correct entitlements.

# Apple configuration

The minimum support iOS version is 12.

Each app that is deployed to the App store will need its own Merchant Identifier that you will need to setup and provide to Waivpay

<https://developer.apple.com/documentation/passkit/apple_pay/setting_up_apple_pay>

Within Apple Developer you will need to configure your application to have the capabilities and additional capabilities:

## Capabilities

Certificates, Identifiers & Profiles > Identifiers > Your App > Capabilities

Graphical user interface, text, application, email

Description automatically generated

You will need to add capabilities to your app to use the SDK

In Capabilities, perform the following:

1. Check Apple Pay Payment Processing



1. Provide the Merchant ID
2. Click Edit and provide the Merchant Id to Waivpay to setup with their payment processor.

Graphical user interface, text, application

Description automatically generated

1. Check Wallet Capability  
     
   

## Additional Capabilities

Go to Certificates, Identifiers & Profiles > Identifiers > Your App > Additional Capabilities

1. Check In-App Provisioning in Additional Capabilities  
     
   Graphical user interface, text, application, chat or text message, email

   Description automatically generated

## App Name, Identifier & Bundle ID

Go to App Store Connect > My Apps > Your App > General > App Information

1. Provide the App Name, Identifier & Bundle ID to Waivpay for Whitelisting

Graphical user interface, application

Description automatically generated

## Provisioning Profile

Go to Certificates, Identifiers & Profiles > Profiles

1. Create your provisioning profile and ensure it has the capabilities you have specified above.

Graphical user interface, text, application, email

Description automatically generated

1. Manually configure the signing of your application within Xcode, ensure you import or select the correct provisioning profile.

Graphical user interface, text, application, email

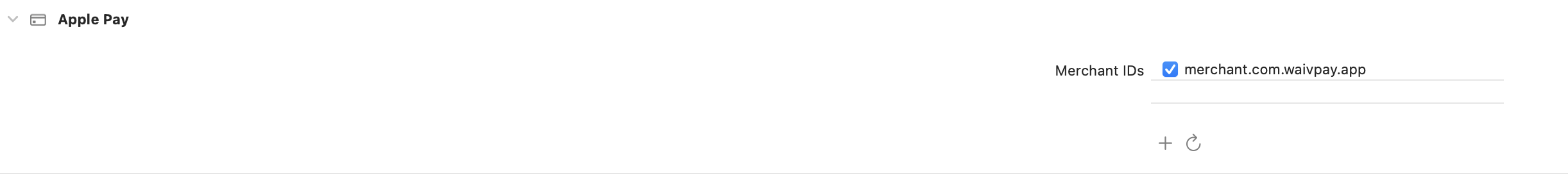
Description automatically generated

1. Add the apple pay entitlement to your project.

Graphical user interface, application

Description automatically generated

1. Ensure that you have the merchant IDs selected



1. Ensure that the payment pass has been added to your entitlements file.

Graphical user interface, application

Description automatically generated

## Certificates

For apple pay you will need to create an Apple Pay Payment Processing certificate thru the Create a New Certficate.

Text, application

Description automatically generated

Follow the process which includes selecting the Merchant ID and download the certificate. This will need to be provided to WaivPay to configure against their Stripe configuration.

## Stripe Apple Pay Integration

Refer to Stripe document if required: <https://stripe.com/docs/apple-pay> also the SampleApp is configured for Apple Pay.

**Steps to Implement**

1. Install Stripe- Apple Pay SDK **:** <https://github.com/stripe/stripe-react-native>

|  |
| --- |
| Login into npm repository |
| yarn add @stripe/stripe-react-native |

1. Initialize Stripe

|  |
| --- |
| Call following function from useEffect() of App.tsx |
| import { initStripe } from '@stripe/stripe-react-native';  ….  initStripe({  publishableKey: "To-be-obtained-from-Waivpay",  merchantIdentifier: "merchant.com.xxx"  }); |

1. Add the following to App.json

|  |
| --- |
|  |
| "expo": {  "plugins": [  [ "@stripe/stripe-react-native",  {  "merchantIdentifier": "com.XXX",  "enableGooglePay": false  }  ]  ]  } |

1. Add the button to your check out page

This will check if apple pay is support and also show the button to the user.

|  |
| --- |
|  |
| import { ApplePayButton, useApplePay } from '@stripe/stripe-react-native';  const { isApplePaySupported } = useApplePay();    return (     <View>       {isApplePaySupported && (         <ApplePayButton           onPress={pay}           type="plain"           buttonStyle="black"           borderRadius={4}           style={{             width: '100%',             height: 50,           }}         />       )}     </View>  ); |

Newer versions of Stripe SDK will have something more like

|  |
| --- |
|  |
| import { PlatformPay, PlatformPayButton, usePlatformPay } from '@stripe/stripe-react-native';  const { isPlatformPaySupported } = usePlatformPay ();    return (     <View>       {isPlatformPaySupported && (         <PlatformPayButton           onPress={pay}  type={PlatformPay.ButtonType.Order}  appearance={PlatformPay.ButtonStyle.Black}  borderRadius={4}           style={{             width: '100%',             height: 50,           }}         />       )}     </View>  ); |

1. Make the payment. Dependent on Stripe SDK versions, there are a couple of different flows.

Older versions:

* 1. Setup the cart item and present the payment sheet from the Stripe SDK
  2. Force user to enter billing address by passing requiredBillingContactFields: ['postalAddress'] into presentApplePay call.
  3. Create Order object using SDK. Pass value : “apple\_pay” to “payment\_method” field of Order model in the SDK. Call createOrder function from SDK by passing order object into the function.  N.B: When payment\_method is “apple\_pay”, capture billing address details from result of presentApplePay call and populate billing address of order with values from result.
  4. The response of the function (OrderResponse model) comes back with “stripe\_client\_secret” field populated.
  5. Call confirmApplePayPayment from Stripe SDK by passing the acquired stripe\_client\_secret

|  |
| --- |
|  |
| const { presentApplePay, isApplePaySupported } = useApplePay();    var cartItem = JSON.parse('{ "paymentType": "Immediate", "label": "Waivpay Gift Card", "amount": "14.00" }');  const { error } = await presentApplePay({  cartItems: [cartItem],  country: 'AU',  currency: 'AUD',  requiredBillingContactFields: ['postalAddress'],  }).then(async (value) => {  var nameR = value.paymentMethod.billingDetails.name;  var first\_name = nameR.split(" ")[0];  var index = nameR.indexOf(first\_name);  var length = first\_name.length;  var last\_name = nameR.slice(index + length + 1);  var billingAddress = new BillingAddress();  billingAddress.first\_name = first\_name;  billingAddress.last\_name = last\_name;  billingAddress.postcode = value.paymentMethod.billingDetails.address.postalCode;  billingAddress.state = value.paymentMethod.billingDetails.address.state  ;  billingAddress.street\_address = add1 = value.paymentMethod.billingDetails.address.line1  + “ ” + value.paymentMethod.billingDetails.address.line2;  billingAddress.suburb = value.paymentMethod.billingDetails.address.city;  order.billing\_address = billingAddress;  var clientSecret = orderReponse.stripe\_client\_secret;  console.log("Client Secret is :" + clientSecret);  const { error: confirmError } = await confirmApplePayPayment(  clientSecret  );    createOrder(order).then(async (value) => {  console.log("Order response is :" + JSON.stringify(value));  var orderReponse = new OrderResponse();  orderReponse = value;  if (orderReponse.hasError != null) {  Alert.alert(orderReponse.error);  } else {  var clientSecret = orderReponse.stripe\_client\_secret;  console.log("Client Secret is :" + clientSecret);  const { error: confirmError } = await confirmApplePayPayment(  clientSecret  );    if (confirmError) {  Alert.alert("Drats! There's been an error :" + confirmError);  } else {  Alert.alert("Bingo! Payment via Apple Pay has been successful");  }  }  });  }).catch(console.log("Error in presentApplePay")); |

 Newer versions (0.29.0+):

|  |
| --- |
|  |
| const { createPlatformPayPaymentMethod } = usePlatformPay();    var cartItem = JSON.parse('{ "paymentType": PlatformPay.PaymentType.Immediate, "label": "Waivpay Gift Card", "amount": "14.00" }');  const { error } = await createPlatformPayPaymentMethod ({  applePay: {  cartItems: [cartItem],  country: 'AU',  currency: 'AUD',  requiredBillingContactFields: ['postalAddress']  }  }).then(async (paymentResult) => {  var nameR = value.paymentMethod.billingDetails.name;  var first\_name = nameR.split(" ")[0];  var index = nameR.indexOf(first\_name);  var length = first\_name.length;  var last\_name = nameR.slice(index + length + 1);  var billingAddress = new BillingAddress();  billingAddress.first\_name = first\_name;  billingAddress.last\_name = last\_name;  billingAddress.postcode = value.paymentMethod.billingDetails.address.postalCode;  billingAddress.state = value.paymentMethod.billingDetails.address.state  ;  billingAddress.street\_address = add1 = value.paymentMethod.billingDetails.address.line1  + “ ” + value.paymentMethod.billingDetails.address.line2;  billingAddress.suburb = value.paymentMethod.billingDetails.address.city;  order.billing\_address = billingAddress;  order.stripe\_payment\_method\_id = paymentResult.paymentMethod.id;  var clientSecret = orderReponse.stripe\_client\_secret;  console.log("Client Secret is :" + clientSecret);  const { error: confirmError } = await confirmApplePayPayment(  clientSecret  );    createOrder(order).then(async (value) => {  console.log("Order response is :" + JSON.stringify(value));  var orderReponse = new OrderResponse();  orderReponse = value;  if (orderReponse.hasError != null) {  Alert.alert(orderReponse.error);  } else {  Alert.alert("Bingo! Payment via Apple Pay has been successful");  }  });  }).catch(console.log("Error in createPlatformPayPaymentMethod ")); |

# Google configuration

The minimum support Android version is 31.

## App Identifier

Go to Google Cloud Platform > Console > Select your Project > Project Settings

1. Provide the following to Waivpay

Text

Description automatically generated with medium confidence

## SHA256 Fingerprint

Go to Google Play Console > Select your App > App Integrity > App Signing

You will need to provide:

1. SHA-256 certificate fingerprint

Graphical user interface, text, application, email

Description automatically generated

You will also need to provide your SHA-256 fingerprint from your local application to ensure it is whitelisted with google.

|  |
| --- |
| Run command – substitute path to your location |
| keytool -printcert -jarfile /Application/android/app/build/outputs/apk/debug/app-debug.apk | grep SHA256 |

# Installation

Once you have been provided your credentials you can run the following command to add the SDK to your app.

**Setup NPM Access to Repository for the Application**

For npm/yarn to access the private repository automatically you can login locally as per the command below.

|  |
| --- |
| Login into npm repository |
| npm login --scope=@waivpay --registry=https://npm.pkg.github.com |

You will be prompted for your username, password (“Personal access token”) and email address for github.

Private access token within github

Graphical user interface, text, application, email

Description automatically generated

**For NPM Installation**

Download the file called .npmrc which will guide npm on the repository to download the SDK from. This file is to be in the same directory as the package.json.

|  |
| --- |
| .npmrc |
| @waivpay:registry=https://npm.pkg.github.com |

The install command requires you to specify the correct version number.

|  |
| --- |
| NPM Install |
| npm install @waivpay/mobile-sdk@x.x.x  cd ios  pod install |

In some setups there is a bug in which the dependency is not recognized when installed as above. In this case the best resolution is to download the tar file from the repository and manual install as a tarball. Update the version to the latest version in the repository.

|  |
| --- |
| NPM Install |
| npm install mobile-sdk-x.x.x.tgz  cd ios  pod install |

**For Yarn Installation**

You need a file called .yarnrc which will guide yarn on the repository to download the SDK from. This file is to be in the same directory as the package.json.

|  |
| --- |
| .yarnrc |
| "@waivpay:registry" "https://npm.pkg.github.com" |

To install the latest version in yarn run the following but first update the version number.

|  |
| --- |
| Yarn Install |
| yarn add @waivpay/mobile-sdk@x.x.x  cd ios  pod install |

**Android Installation**

Android Google Tap and Pay services are only available for authorised users and their SDK is required for the add to wallet functionality. The SDK comes with a private maven repository packaged in the file path as per google recommended documentation. You will need to setup in your Android project to use the private maven repository in your root level build.gradle.

|  |
| --- |
| build.gradle |
| allprojects {  repositories {  ……  maven { url "file:/$rootDir/../node\_modules/@waivpay/mobile-sdk/android/privateMavenRepository" }  }  } |

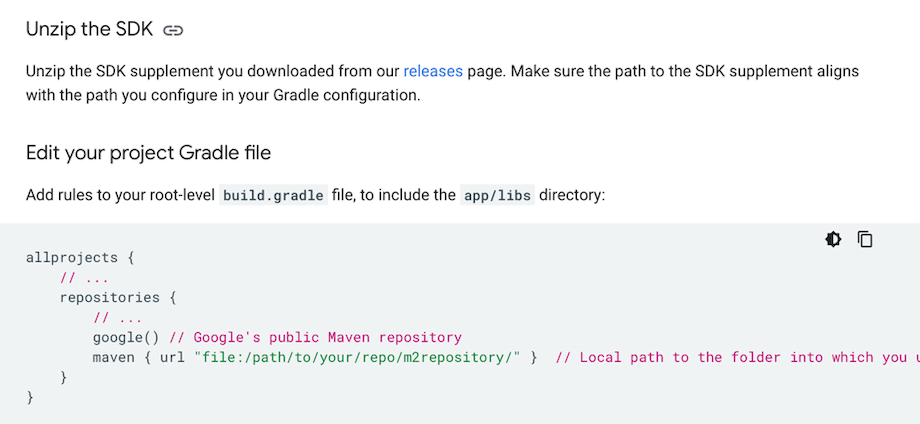


Figure 1 Snippet from Google SDK Installation notes

# Module Dependencies

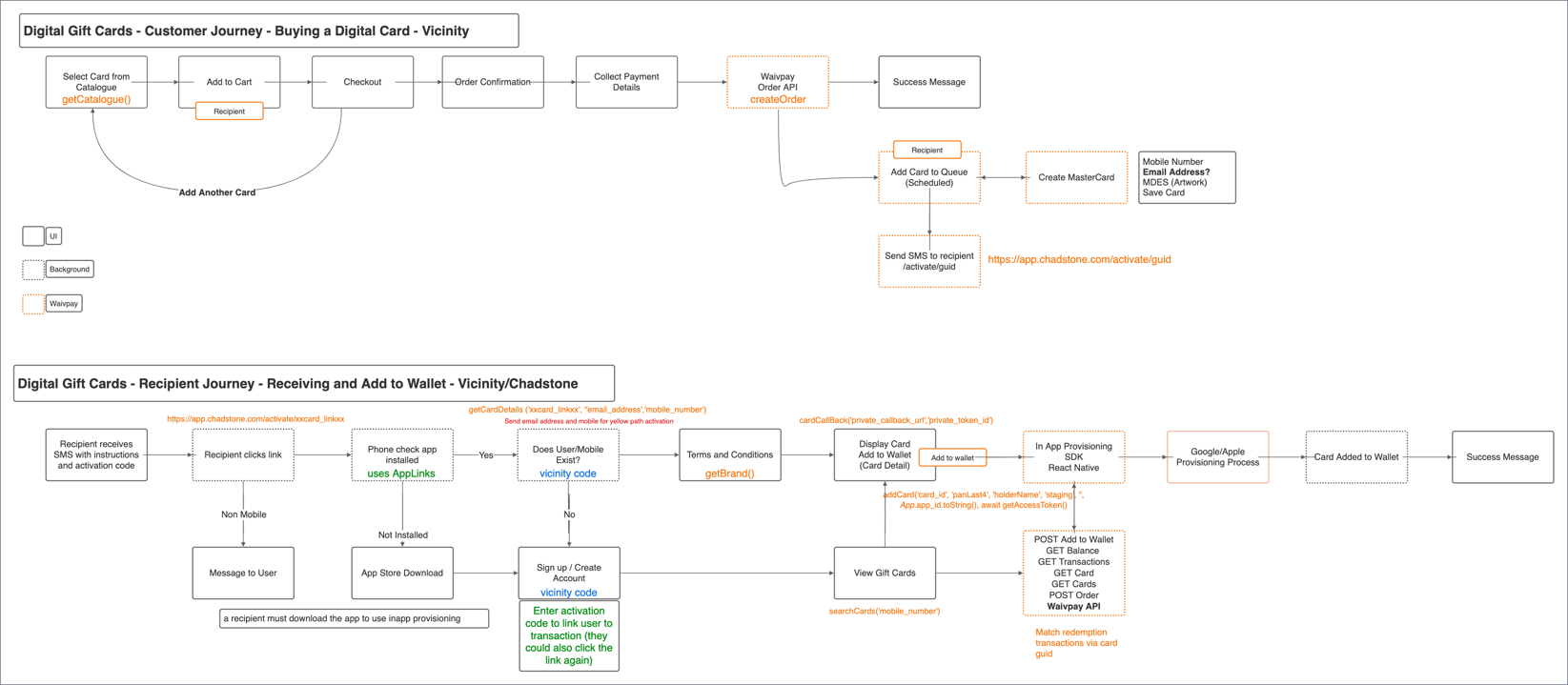
You will need to install these module dependencies to use the SDK:

[react-native-encrypted-storage@^4.0.2](mailto:react-native-encrypted-storage@%5e4.0.2)

# APIs

For details of the APIs can be reference here <https://docs.waivpay.com>. This documentation shows the raw APIs and sample request and responses. You can also see a basic SampleApp showing the main usage of the APIs.

Here is the flow with the recommended API calls.



## Configuration

Before calling any functions on the SDK, you will have to configure it by calling the setConfig function with the access information provided by Waivpay. This will allow the SDK to authentication your access to the Waivpay services.

|  |
| --- |
| JavaScript / TypeScript |
| import {setConfig} from "@waivpay/mobile-sdk/src/ApiCall";  import {AppConfig} from "@waivpay/mobile-sdk/src/Models/AppConfig";  …..  let appconfig = new AppConfig();  appconfig.app\_id = "app-id";  appconfig.client\_id = "client\_id";  appconfig.client\_secret = "client\_secret";  appconfig.environment = "staging";  setConfig(appconfig); |

Appconfig class also takes two optional parameters – host and headers. This sets the host name for all API calls and adds extra headers to existing ones.

|  |
| --- |
| JavaScript / TypeScript |
| import {setConfig} from "@waivpay/mobile-sdk/src/ApiCall";  import {AppConfig} from "@waivpay/mobile-sdk/src/Models/AppConfig";  …..  let appconfig = new AppConfig();  appconfig.app\_id = "app-id";  appconfig.client\_id = "client\_id";  appconfig.client\_secret = "client\_secret";  appconfig.environment = "staging";  appconfig.host = "https://webstores-staging.herokuapp.com";  appconfig.headers ={a : 'xx', b :  'yy'};  setConfig(appconfig); |

activateBeacon

This method can only be called once per application session. This method call starts a riskified beacon session used for fraud detection. It is recommended to be called at the launch of the app and after setConfig method has been called.

|  |
| --- |
| JavaScript / TypeScript |
| import {activateBeacon} from "@waivpay/mobile-sdk/src/ApiCall";  …..  setConfig(appconfig);  activateBeacon(); |

getCatalogue

The getCatalogue function provides a list of available gift cards for the configured application. These are the cards that are available for sale by your application.

|  |
| --- |
| JavaScript / TypeScript |
| import {getCatalogue} from "@waivpay/mobile-sdk/src/ApiCall";  import {Catalogue} from "@waivpay/mobile-sdk/src/Models/Catalogue";  …..  getCatalogue().then((value) => {  const cat = new Catalogue();  cat = value;  }); |

getBalance

The getBalance function provides the balance of a gift card.

|  |
| --- |
| JavaScript / TypeScript |
| import {getBalance} from "@waivpay/mobile-sdk/src/ApiCall";  import {Balance} from "@waivpay/mobile-sdk/src/Models/Balance";  …..  getBalance('Card\_Id').then(async (value) => {  console.log('Good - Application ' + JSON.stringify(value));  Alert.alert(JSON.stringify(value));  let balance = new Balance();  balance = value;  }).catch((error)=> {  console.log('error - Application deal with');  console.log(error.message);  Alert.alert(error.message);  }).finally(()=> {  console.log('finally - Application ');  }); |

getTransactions

The getTransactions function provides the list of transactions against a card.

|  |
| --- |
| JavaScript / TypeScript |
| import {TransactionList} from "@waivpay/mobile-sdk/src/Models/TransactionList"  import {Transaction} from "@waivpay/mobile-sdk/src/Models/Transaction"  import {getTransactions} from "@waivpay/mobile-sdk/src/ApiCall";  …..  getTransactions('Card\_Id').then((value) => {  let transactionList = new TransactionList(new Array());  transactionList = value;  let transaction = new Transaction();  transaction = transactionList.transactions[0];  }).catch((error)=> {  console.log('error - Application deal with');  console.log(error.message);  Alert.alert(error.message);  }).finally(()=> {  console.log('finally - Application ');  }); |

getCardDetails

The getCardDetails function provides details about the card passed in. This method will also take the passed in mobile\_number to confirm that the referenced card number belongs to this mobile number. It will also update the email\_address to the card for provisioning later. The response in this call returns 2 fields: private\_callback\_url and private\_token\_id, these values are used in the cardCallBack function to get the card pan, ccv and expiry date.

|  |
| --- |
| JavaScript / TypeScript |
| import {Card} from "@waivpay/mobile-sdk/src/Models/Card"  import {getCardDetails} from "@waivpay/mobile-sdk/src/ApiCall";  …..  getCardDetails("Card\_id","email\_Address","mobile\_number").then((value) => {  let card = new Card();  card = value;  }).catch((error)=> {  console.log('error - Application deal with');  console.log(error.message);  Alert.alert(error.message);  }).finally(()=> {  console.log('finally - Application ');  }); |

cardCallBack

The cardCallBack function is used when you want to display the card information to user or to get the pan to send when adding to wallet. The private\_callback\_url and private\_token\_id are found from the getCardDetails call. This function should be only be called when it is needed. The private\_token\_id is only active for 5 minutes.

|  |
| --- |
| JavaScript / TypeScript |
| import {CardCallBackResponse} from "@waivpay/mobile-sdk/src/Models/ CardCallBackResponse "  import {cardCallBack} from "@waivpay/mobile-sdk/src/ApiCall";  …..  cardCardBack("private\_callback\_url","private\_token\_id").then((value) => {  let cardCallBackResponse = new CardCallBackResponse ();  cardCallBackResponse = value;  }).catch((error)=> {  console.log('error - Application deal with');  console.log(error.message);  Alert.alert(error.message);  }).finally(()=> {  console.log('finally - Application ');  }); |

sendTwoFactor

The sendTwoFactor function send an auth code to the user’’s provided mobile number.

|  |
| --- |
| JavaScript / TypeScript |
| import {sendTwoFactor} from "@waivpay/mobile-sdk/src/ApiCall";  …..  sendTwoFactor ("mobileNumber"); |

verifyTwoFactor

The verifyTwoFactor function will verify that the auth code and mobile number are correct.

|  |
| --- |
| JavaScript / TypeScript |
| import {verifyTwoFactor} from "@waivpay/mobile-sdk/src/ApiCall";  …..  verifyTwoFactor ("Code").then((value) => {  let user = value;  }).catch((error)=> {  console.log('error - Application deal with');  console.log(error.message);  Alert.alert(error.message);  }).finally(()=> {  console.log('finally - Application ');  }); |

getBrand

The getBrand function sends details about the brand / application.

|  |
| --- |
| JavaScript / TypeScript |
| import {Brand} from "@waivpay/mobile-sdk/src/Models/Brand";  import {Location} from "@waivpay/mobile-sdk/src/Models/Location";  import {getBrand} from "@waivpay/mobile-sdk/src/ApiCall";  …..  getBrand().then((value) => {  let brand = value;  }).catch((error)=> {  console.log('error - Application deal with');  console.log(error.message);  Alert.alert(error.message);  }).finally(()=> {  console.log('finally - Application ');  }); |

createOrder

The createOrder function will take the order and payment information and send to Waivpay for processing.

|  |
| --- |
| JavaScript / TypeScript |
| import {Order} from "@waivpay/mobile-sdk/src/Models/Order";  import {OrderItem} from "@waivpay/mobile-sdk/src/Models/OrderItem";  import {OrderResponse} from "@waivpay/mobile-sdk/src/Models/OrderResponse";  import {createOrder} from "@waivpay/mobile-sdk/src/ApiCall";  …..  var billingAddress = new BillingAddress();  billingAddress.company\_name = "";  billingAddress.first\_name = "Bruce";  billingAddress.last\_name = "Wayne";  billingAddress.postcode = "2060";  billingAddress.state = "NSW";  billingAddress.street\_address = "Street Address";  billingAddress.suburb = "North Sydney";    var orderItem = new OrderItem();  orderItem.amount=Number(amount);  orderItem.card\_design\_id =Number(cardDesignId);  orderItem.card\_type = "digital";  orderItem.delivery\_email = delEmail;  orderItem.delivery\_first\_name = "Bruce";  orderItem.delivery\_last\_name = "Wayne";  orderItem.delivery\_sms\_number = delSmsNumber;  orderItem.from = "";  orderItem.include\_message = false;  orderItem.quantity = 1;  orderItem.to = to;    var orderItems = new Array<OrderItem>();  orderItems.push(orderItem);  var order = new Order();  order.billing\_address=billingAddress;  order.order\_items=orderItems;  order.email="example@somehost.com";  order.phone\_number="9876543210";  order.billing\_same\_address=false;  order.credit\_card\_number="4444333322221111";  order.credit\_card\_expiry\_month="08";  order.credit\_card\_expiry\_year="2023";  order.credit\_card\_security\_code="123";  order.credit\_card\_name="Bruce Wayne";  createOrder(order).then((value)=> {  var orderResponse = new OrderResponse();  OrderResponse = value;  }).catch((error)=> {  console.log('error - Application deal with');  console.log(error.message);  Alert.alert(error.message);  }).finally(()=> {  console.log('finally - Application ');  }); |

searchCards

The searchCards function will return a list of cards for the provided mobile number.

|  |
| --- |
| JavaScript / TypeScript |
| import {CardList} from “@waivpay/mobile-sdk/src/Models/CardList";  import {Card} from “@waivpay/mobile-sdk/src/Models/Card";  import {searchCards} from “@waivpay/mobile-sdk/src/ApiCall";  …..  searchCards("mobile\_number").then((value) => {  var cardList = new CardList();  cardList = value;  var card = new Card();  card = cardList.cards[0];  }).catch((error)=> {  console.log('error - Application deal with');  console.log(error.message);  Alert.alert(error.message);  }).finally(()=> {  console.log('finally - Application ');  }); |

addCard

The addCard function will enable you to add the card to wallet. This function will take the provided Card Number call waivpay to confirm and call the native phone functionality to add the card to the wallet. Note: The card\_panLast4 is found from the cardCallBack function.

|  |
| --- |
| JavaScript / TypeScript |
| import {addCard} from “@waivpay/mobile-sdk";  …..  addCard('card\_id', 'card\_panLast4', 'holderName', 'env', 'deliveryEmail', 'appId', getAccessToken()); |

cardExists

The cardExists function will enable check if the card already exists in the device wallet, therefore allows you to determine if you should display the add to wallet button.

|  |
| --- |
| JavaScript / TypeScript |
| import {cardExists} from “@waivpay/mobile-sdk";  …..  cardExist('last 4 pan number').then((value) => {  }); |

getProfile

The getProfile function will return the profile information for the user account.

|  |
| --- |
| JavaScript / TypeScript |
| import {Profile} from “@waivpay/mobile-sdk/src/Models/Profile"  import {getProfile} from “@waivpay/mobile-sdk/src/ApiCall";  …..  getProfile ("user\_id").then((value) => {  var profile = new Profile ();  profile = value;  }).catch((error)=> {  console.log('error - Application deal with');  console.log(error.message);  Alert.alert(error.message);  }).finally(()=> {  console.log('finally - Application ');  }); |

createProfile

The createProfile function will create a user profile for the application.

|  |
| --- |
| JavaScript / TypeScript |
| import {Profile} from “@waivpay/mobile-sdk/src/Models/Profile"  import {createProfile} from “@waivpay/mobile-sdk/src/ApiCall";  ….. |

updateProfile

The updateProfile function will update a user profile for the application.

|  |
| --- |
| JavaScript / TypeScript |
| import {Profile} from “@waivpay/mobile-sdk/src/Models/Profile"  import {updateProfile} from “@waivpay/mobile-sdk/src/ApiCall";  ….. |

setConfigCashBack

The setConfigCashBack function will prepare the application for cashback api access.

|  |
| --- |
| JavaScript / TypeScript |
| import {setConfig} from "@waivpay/mobile-sdk/src/ApiCall";  import {AppConfig} from “@waivpay/mobile-sdk/src/Models/AppConfig";  …..  var appConfig = new AppConfig("client\_id”, “client\_secret”, “app\_id”, “environment(staging|prod)");  setConfigCashBack(appConfig); |

listPromotions

The listPromotions function will list the promotions available for the application.

|  |
| --- |
| JavaScript / TypeScript |
| ….. |

getPromotion

The getPromotion function returns the promotion details of a given promotion.

|  |
| --- |
| JavaScript / TypeScript |
| ….. |

createClaim

The createClaim function will update a user profile for the application.

|  |
| --- |
| JavaScript / TypeScript |
| ….. |

getClaims

The getClaims function will return the list of promotion claims that the user have applied for.

|  |
| --- |
| JavaScript / TypeScript |
| ….. |

fileUpload

The fileUpload function will upload the claim receipt for processing with a claim.

|  |
| --- |
| JavaScript / TypeScript |
| ….. |

# Data Models

|  |  |
| --- | --- |
| AppConfig | * client\_id * client\_secret * app\_id * environment |
| Balance | * balance * expiry\_date |
| BillingAddress | * company\_name * suburb * postcode * state * first\_name * last\_name * street\_address |
| Brand | * identifier * name * iap\_enabled * locations * terms\_of\_use * card\_terms\_and\_conditions * tiles * faqs * home\_page\_text * disable\_parking\_collection |
| Card | * card\_id * amount * recipient\_first\_name * recipient\_last\_name * delivery\_email * delivery\_sms\_number * expiry\_date * card\_design\_id * image * private\_callback\_url * private\_token\_id * created\_at * type * status |
| CardCallBackResponse | * creationTime * modifiedTime * id * cardNumber * panFirst6 * panLast4 * state * sequenceNumber * cardProfileName * pinFailCount * reissue * expiry * customerNumber * embossedName * programName * pan * cvv2 |
| CardList | * cards |
| Catalogue | * products |
| Form\_Schema\_Item | * id * type * label * hint * required * example\_image * example\_text |
| Location | * id * name * state |
| Order | * email * phone\_number * user\_id * validate * external\_user\_id * delivery\_option\_id * billing\_same\_address * recipient\_address * credit\_card\_number * credit\_card\_name * credit\_card\_expiry\_month * credit\_card\_expiry\_year * credit\_card\_security\_code * payment\_method * billing\_address * order\_items * delivery\_option\_id * reference * created\_at * total\_cost |
| OrderItems | * card\_design\_id * card\_type * amount * quantity * delivery\_sms\_number * delivery\_email * include\_message * message * to * delivery\_first\_name * delivery\_last\_name * from * scheduled\_time * card\_fee * card\_image |
| OrderList | * orders |
| OrderResponse | * order\_reference * status * total\_cost * total\_card\_fees * gst * delivery\_fee * error * hasError * stripe\_client\_secret |
| Product | * id * name * active * physical * digital * description * image * fee * minimum\_card\_value * maximum\_card\_value |
| Profile | * id * guid * first\_name * last\_name * email * mobile\_number * marketing\_preference * number\_plate * number\_plate\_state * preferred\_location\_id * allow\_notifications * allow\_location * notifications\_guid * device\_type |
| Promotion | * id * name * title * summary * tile\_image * active * open * start\_date * end\_date * trackable * confirmable * external * country * form\_schema * competition * desktop\_banner * mobile\_banner * banner\_background * form\_header\_image * products * fulfilment\_types * disclaimer * terms\_and\_conditions * privacy\_policy * faqs:string; * support\_email * support\_phone * contact\_us\_content * extra\_claim\_cta * custom\_tabs * has\_preparation\_checks * preparation\_check\_content * preparation\_checks * has\_support\_categories * support\_categories |
| PromotionList | * promotions |
| RecipientAddress | * first\_name * last\_name * company\_name * street\_address * suburb * postcode * state |
| Tile | * name * disable\_purchases |
| Transaction | * description * date * amount * type |
| TransactionList | * transactions |

# Marketing Requirements

The following activities will need to be completed as part of Apple’s marketing requirements for Apple Pay. We can provide templates if required.

|  |  |
| --- | --- |
| Card Artwork |  |
| MDES Configuration | * Logo 1372 x 283 pixels. PNG, (no larger than 3MB) * Customer Service Website, Email & Phone Number * URL for T&Cs and Privacy Policy |
| Apple Pay Landing Page | * Hosted on client website * Live from launch * Link on Home page to landing page Either a Banner or ‘Apple Pay’ button/link * Must sit ‘above the fold’ e.g. not have to scroll to find it * Must be active from launch date * Must be prominent on homepage for first 3 months of term. * You will need to provide Evidence of Apple Pay link on homepage to include in Pack to submit. |
| * Social Media Posts | * Includes an active social media e.g. LinkedIn, Facebook, Instagram * Must post on day of launch * Then once per month for first year of term * Once per quarter for 2nd year of term |
| * EDM | * Sent out within 4 days of launch |
|  |  |
|  |  |