CoinBridge By Nayax

# CoinBridge Solution Implementation Guidelines

Version 2.0

# Table of Contents

General ..... 3
**Customer Pre-Requisites** ..... 4
Existing Customer Mobile Application ..... 4
Existing Customer Loyalty or User Management Platform ..... 4
Solution Components ..... 5
**CoinBridge Mobile SDK: Introduction** ..... 5
The SDK ..... 5
Key Benefits of the CoinBridge SDK ..... 5
**CoinBridge Back-Office Connector: Introduction** ..... 6
Product Implementation Guide ..... 7
**User Journey and Prerequisites** ..... 7
**Pre-Payment User Journey** ..... 8
General ..... 8
Onboarding Your User to the CoinBridge Solution ..... 9
CoinBridge Payment Method Creation Process ..... 10
Finalizing the Creation Process ..... 11
Completing the Pre-Payment Journey ..... 12
**Payment User Journey** ..... 13
General ..... 13
Scenario A: Tap Completed (Happy Flow): ..... 15
Scenario B: Tap Not Completed ..... 16
Scenario C: Payment Action Canceled by User ..... 17
**Transaction Resolution and History** ..... 19
General ..... 19
Additional Transaction Reporting Types ..... 20
Transaction History ..... 21
**User Logout** ..... 22
Appendix ..... 23
**SDK Mandatory Permissions** ..... 23
**Initialization Method** ..... 24
**Authentication Method** ..... 25
**PaymentMethodStatus Method** ..... 26
General ..... 26

Timing ..... 26
Status of Payment Method and Readiness to Perform Payments ..... 26
**PaymentMethodIssuing Method ..... 28**
Manual vs. Automatic Issuing of Payment Method ..... 28
Manual Virtual Card Issuing Flow (using createPaymentMethod) ..... 28
**Android NFC / Contactless Payments Settings Validation ..... 30**
**Transaction Resolution and History ..... 32**
Reasons for Transaction Decline ..... 32
**Card Art Style Guide ..... 34**

# CoinBridge Solution Overview

## General

CoinBridge's patented technology and solution allows for the conversion of any digital asset into real purchases and transactions, at any shop worldwide.

**CoinBridge's technology is comprised of two main elements (services):**

1. CoinBridge SDK – Embeds into any existing mobile application. It acts as an internal e-Wallet within the host mobile app and provides payment capabilities at points of sale (POS), via the NFC/ EMV protocol (with a Tap & Go user experience) over the credit card scheme.
2. CoinBridge Back-Office Connector – Connects loyalty/ user management platforms with the CoinBridge platform, enabling CoinBridge to query user balance and spending policies in real time, whenever a transaction request is received. Once a purchase has been completed, this connector also enables CoinBridge to provide the loyalty/ user management platform with transaction data.

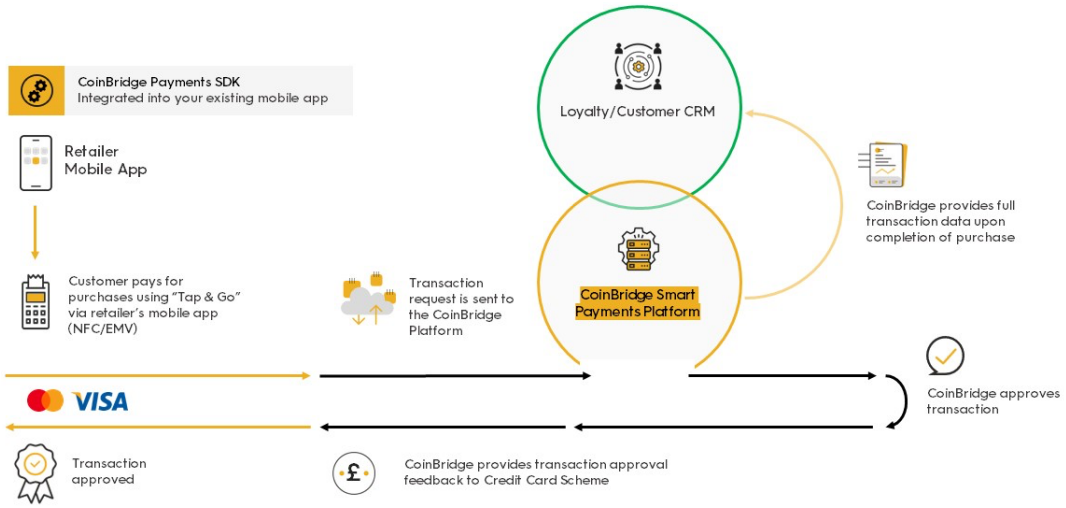



*Diagram 1 Solution Diagram: Transaction Execution Flow*

## Customer Pre-Requisites

The following describes the core technology touchpoints, which are essential to enable the CoinBridge payment service:

### Existing Customer Mobile Application

This app will have to be updated to include the CoinBridge SKD. The CoinBridge SDK will need to be implemented within the mobile app to enable the end-user NFC/ EMV, Tap & Go payment experience. The SDK also provides the mobile application with additional services and features, as described below.

### Existing Customer Loyalty or User Management Platform

A customer management platform is essential for the use of CoinBridge services. It allows for the dynamic, real-time assessment of user balance and related spending policies, so the CoinBridge service can further evaluate that data against any transaction request, and either approve or decline transactions attempts before each transaction by each user.

### Two-Factor Authentication (2FA) for Tokenization Security

According to Financial Regulations and as an integral part of the CoinBridge Tokenization Process, it is a mandatory requirement to incorporate Two-Factor Authentication (2FA) for enhanced security and for regulation purposes. This crucial measure is designed to fortify the protection of user accounts and sensitive data during the tokenization journey.

CoinBridge strongly encourage to enable 2FA during the login process to your application. Additionally, it is permissible to implement 2FA upon the user's initial joining to the CoinBridge solution.

### Android Device Security Requirement: Biometric (Passcode, Pattern, or Fingerprint) for Payment Feature

For optimal security when using the CoinBridge solution on Android devices, users are required to have either a passcode, pattern, or fingerprint lock configured.

This multi-option approach to biometric authentication ensures flexibility and robust security for the utilization of the payment feature.

Configuring a passcode, pattern, or fingerprint on Android devices aligns with CoinBridge's commitment to maintaining high-security standards on the Android platform, offering users the choice of authentication method based on their device capabilities and preferences.

# Solution Components

The following CoinBridge service components are required to enable the service:

1. **CoinBridge Mobile SDK**
2. **CoinBridge Back-Office Connector**

## CoinBridge Mobile SDK: Introduction

### The SDK

- An SDK is the mobile element embedded into a mobile app.
- The CoinBridge SDK is compatible with any Android mobile application.
- With CoinBridge SDK, you can convert digital assets (such as points and rewards) into real purchases (transactions) through EMV over NFC, directly from the mobile application.

### Key Benefits of the CoinBridge SDK

- Easily embeds into any existing mobile application.
- Transparent to the user (does not change the look & feel of the application).

The SDK must be used in tandem with the CoinBridge Back-Office Connector to enable the full functionality of the solution.

## CoinBridge Back-Office Connector: Introduction

The CoinBridge Back-Office Connector connects the customer's loyalty or CRM (or similar) platform with the CoinBridge platform.

This connection is imperative to enable CoinBridge to dynamically draw information from the customer's platform whenever they make a purchase (driving a transaction), and for the CoinBridge platform to provide transaction details feedback back to the customer's platform.

Connection to the CoinBridge Platform allows for the following main process:

- Verification of user service eligibility.
- Retrieval of the user's current balance (or balances - when dealing with multiple products per end-user).
- Retrieval of the user's related spending policy(ies).
    - Spending policy(ies) are an integral part of the program that permits or limits user spending in any way or form.
    - A spending policy is a rule or set of rules that details all allowances and restrictions relevant to a specific user at the time of transaction. These rules are set and managed by the customer on the customer's own platform.
    - For example, such policies may include (but are not limited to):
        - Restricting or allowing certain merchant IDs (MID)
        - Restricting or allowing certain merchant categories (MCC)
        - Limiting spending per transaction
- Data and feedback – CoinBridge updates the customer's loyalty or CRM platform, depending on the transaction type.
    - **Payment Transaction** - Transmits the transaction result, whether it has been approved or declined, and full transaction details.
    - **Refund Transaction** – Transmits refund transaction feedback and additional data.
    - **Transaction Update** – Transmits details of a transaction cancellation, on an already-reported successful transaction, due to a problem that may have occurred.

⚠ NOTE: Full Product & Technical information about the CoinBridge Back-Office Connector can be found in the dedicated CoinBridge Back-Office Connector documentation.

# Product Implementation Guide

(Mobile Application Adaptation)

## User Journey and Prerequisites

Please ensure that the following steps have been completed prior to proceeding with this chapter:

1. Integration of CoinBridge SDK
2. Integration of CoinBridge Back-Office Connector
3. Giving CoinBridge SDK all mandatory permissions to function (See Appendix for more information)

⚠ **Mandatory Two-Factor Authentication (2FA) for User Login**: As mentioned earlier, for payment applications, it is imperative that your application must implement Two-Factor Authentication (2FA) for every user login. This critical security measure is essential to fortify the protection of user accounts and sensitive payment data.

The user journey is divided into two phases:

- **Pre-payment user journey** – How users will experience your application when they open it for the first time, and onboarding.
- **Payment user journey** – How users will experience the solution each time they use it.

The **Pre-Payment User Journey** is the process a user goes through when they open your application for the first time (after CoinBridge SDK has been integrated).
As soon as the user shows intent, CoinBridge issues a virtual card, tokenizes it, and pushes it to the device's wallet.
After this journey is completed, the user will not need to go through it again, and their virtual card will be available whenever they want to make a payment.

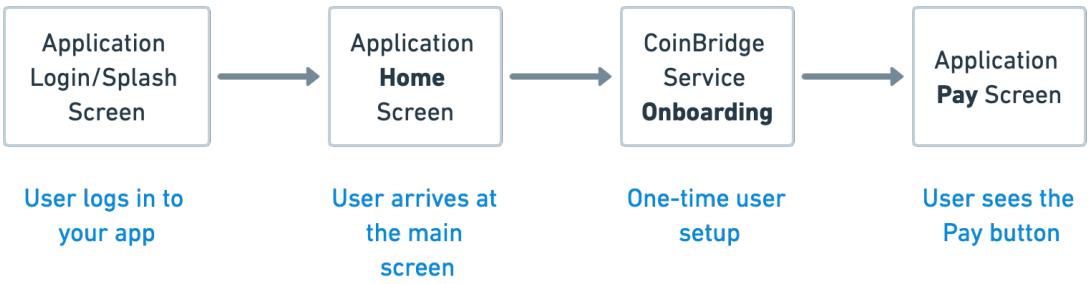
The **Payment User Journey** represents the Tap & Go experience that a user goes through whenever they wish to pay using your application.

In the following chapters, we will explain each step of the user journey, with technical information and design suggestions for your application.

⚙ TECHNICAL INFO: The combination of the `isAuthenticated` method followed by the `initialization` method is crucial and should be applied every time your application starts, ensuring the seamless functioning of the payment process. For a deeper understanding of the initialization method, refer to Appendix 1 in this document.

# Pre-Payment User Journey

## General

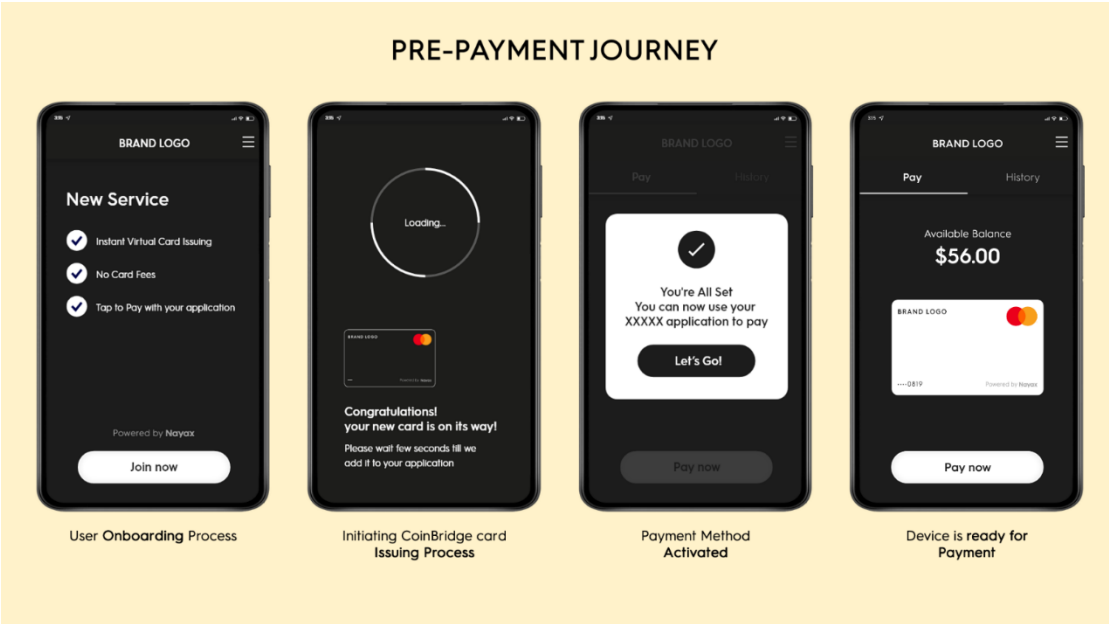



The Pre-Payment process covers all processes that the mobile application should perform, in order to reach the point at which the application is ready to make payments using the CoinBridge SDK.

The set of processes includes:

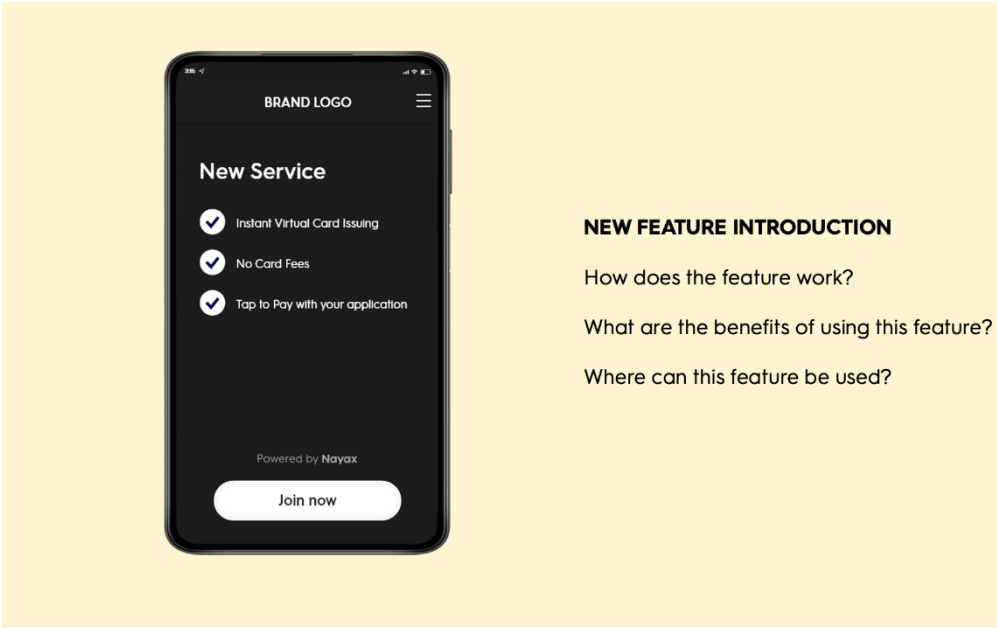
- The steps that the SDK and your application must perform to get to the point where the user can make payments with your app.
- User experience & user interface changes that should take place.

As you read on, we will elaborate, step-by-step, on the technical steps to be carried out, using various CoinBridge SDK methods, and what the user should see on-screen at each point of the flow.



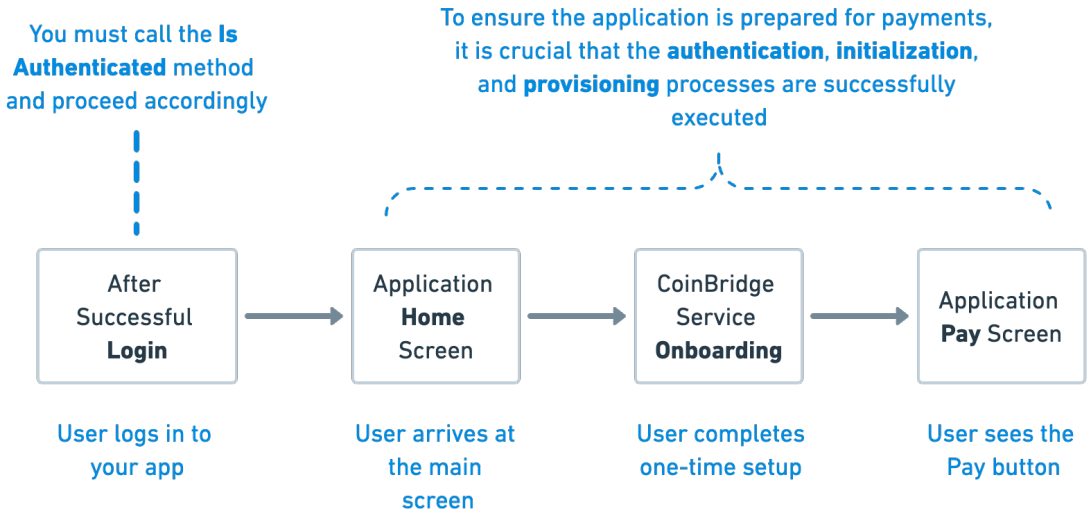
## Onboarding Your User to the CoinBridge Solution

### CoinBridge Service Introduction and Provision:



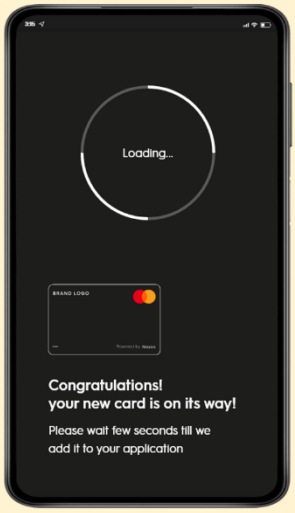

It is crucial to present your user with a proper promotion and introduction to the CoinBridge solution, while emphasizing its benefits and explaining related flows, including the payment flow. It is recommended to do this via a dedicated onboarding page, with a clear CTA for the user to acknowledge.

⚙ TECHNICAL INFO: As mentioned earlier, make sure to use the `isAuthenticated` method every time your application starts. If this call returns ***false***, it means that CoinBridge SDK did not recognize the user/ device, and the Application should call the `Authentication` method. If this call returns ***true***, it means that CoinBridge SDK is authenticated, and the Application should call the `Initialization` method. Further details regarding Initialization & Authentication can be found in Appendices 1 and 2 to this document.




## CoinBridge Payment Method Creation Process

For first time users only, CoinBridge will initiate the payment method creation process.




**VIRTUAL CARD CREATION PROCESS**

- The virtual card creation process may take up to 10 seconds.
- To ensure a smooth user experience, we suggest notifying the user to wait few moments until the process is complete.
- This is the time to provide information about any spending policy restrictions.
- The user must stay in your application and not close it during the process.
- Once the virtual card is ready, promptly notify the user to start using it.

The virtual card issuing process (and provisioning) usually takes up to 10 seconds, so while CoinBridge creates the payment method for your user, is the following steps are **highly recommended**:

- To ensure a smooth user experience, show a loader animation with appropriate text. We suggest notifying the user to wait a few moments until the process is completed.
- It's important that the user does not leave the application before this process is finalized.

  - If the process is aborted, the process will restart automatically upon reentering the application (and performing a successful initialization or the CoinBridge SDK).
- This is the time to provide information about any spending policy restrictions (or advise where they can be found, for reference).
- Once the process is completed, the CoinBridge SDK will notify your application.

⚙ TECHNICAL INFO: Upon successful completion of the initialization process, you will receive an `initiCompleted` callback providing details about the current state of the virtual card. Within this callback, `hasCard` will be either true or false, indicating whether a virtual card has been issued for the user, or not.
The `CBPaymentStatus` object provides additional information about the Virtual Card. If `hasCard` returns False, it signifies that a virtual card should be issued for the user (either automatically or manually). Conversely, if it returns True, the virtual card has already been issued for this user, and card details will be available within the("`CBPaymentStatus`") object.
Detailed information regarding the `CBPaymentStatus` method can be found in the Appendix 3 to this document.

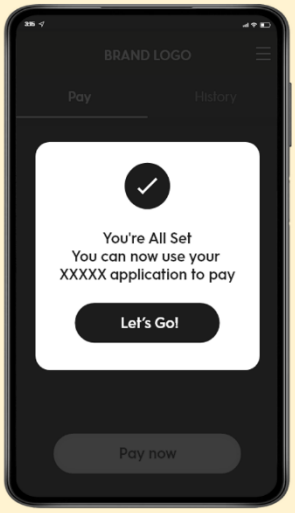
## Finalizing the Creation Process

In the Virtual Card creation workflow, the concluding step involves pushing the virtual card token into the CoinBridge SDK wallet.
This step is indicated by the status where `hasCard` is true and `isProvisioned` is false.
In this scenario, your application is required to invoke the `Provision card` method and anticipate the `onPaymentMethodStatusChange` event through the `CBNotification` service. This event will convey an updated status, with the isProvisioned parameter transitioning to true.

This finalization step can be executed in two ways:

- **User-Initiated Approach**
  Utilize a call-to-action (CTA) button within your user interface. This button triggers the Provision card method, allowing the application to progress to the next step.
- **Automated Approach**
  Alternatively, your application can autonomously initiate the Provision card method while in a "waiting screen" mode. It will then await the updated status of the `isProvisioned` parameter before advancing to the subsequent step.

## Completing the Pre-Payment Journey




**YOUR VIRTUAL CARD IS NOW IN YOUR APPLICATION**

- Congratulate your user once their virtual card is ready for use.
- Provide additional information about what to do next.
- Let them know that they can start using your application to perform payments!

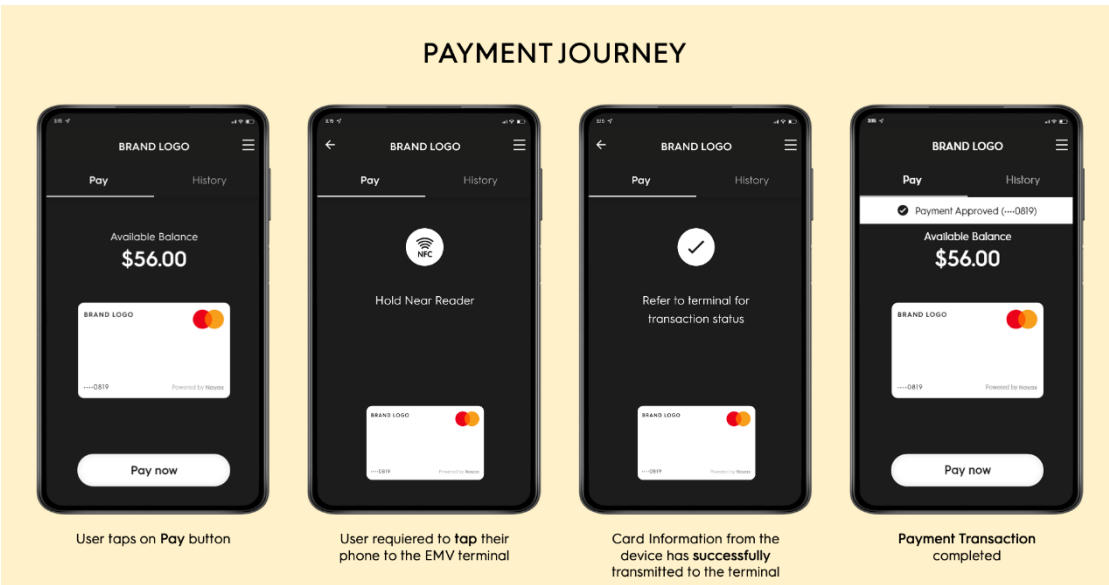
The CoinBridge SDK will notify your application once the payment method creation flow has been successfully completed.

- Upon completion, it is important to congratulate the user that their device is now ready to perform payments using your application.
- This congrats screen / popup is a great place to provide additional information, right before the user makes their first payment!
- Once the user has acknowledged the above, the pre-payment flow has successfully concluded, and you can redirect the user to the new payment home screen.
- From this moment on, the payment home screen will serve as the user's main screen for performing Tap & Go payments. (See the next chapter for further details.)

TECHNICAL INFO: The process of creating a payment method is relevant only for new users of the CoinBridge payment solution. If an existing user chooses to log in to your app using a different mobile device, Authentication will be required during Initialization, but new virtual card issuing won't be required. The CoinBridge SDK will simply carry out the Card Provisioning step in the background. If manual card issuance is necessary, please refer to the relevant appendix for additional information. The relevant appendix also contains information on design guidelines and non-happy flows.

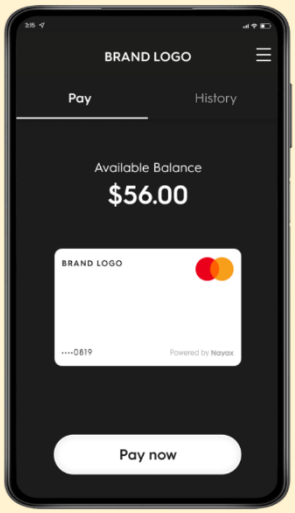
# Payment User Journey

## General




The Payment User Journey represents the Tap & Go experience that a user goes through whenever they wish to pay using your application.

As you read on, we will elaborate, step-by-step, on what the user should see on-screen at each point of the flow.




**PAYMENT SCREEN**

The Payment screen represents the "Home Screen" of the CoinBridge solution.

Your users will be able to perform payments and view payment-related information/ activities via this screen.

The Payment screen must include:

- User available balance
- An image of the virtual card (see Appendix for card art style guide
- **"Pay Now"** CTA button

This is the main payment screen of your application, from which your users will be able to perform all Tap & Go related activities.

To enable a proper payment experience, your application must include the following:

1. **Payment Screen** must include the following:
    a. The user's available balance to perform payments (in the relevant currency).
    b. Image of the virtual card:
        i. Card art refers to the visual design and illustration that will appear within your application (in Android), or within the Apple Pay digital wallet.
        ii. Please follow the style guide found in the Appendix to this document to properly display the card art.
    c. Payment restrictions - spending policy guidelines (if applicable)
    d. Payment CTA button
2. **Payment flow screens** (see below mockups for reference) to support:
    a. Happy flow
    b. Unhappy flow
    c. Cancel payment flow

## Android Biometric Prompt

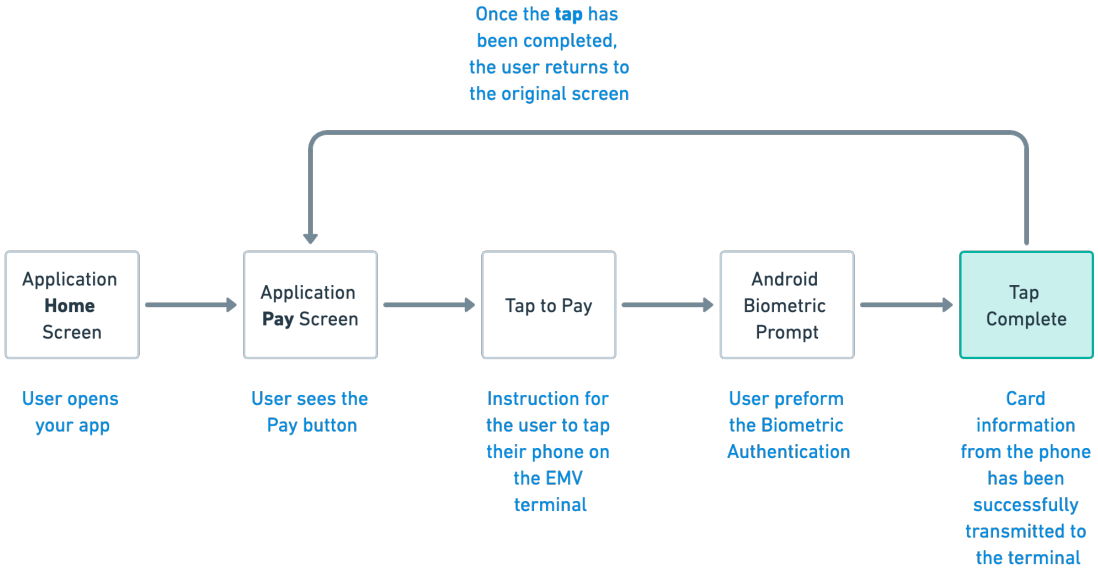
The `Biometric Prompt` in the Android operating system is a feature that provides a standardized and secure method for your application to request biometric authentication, including fingerprint or face recognition.
It is essential to invoke the `Biometric Prompt` and request biometric authentication before executing the actual tap action.
Failure to make this call will result in an `onCredentialsRequired (Context Context` error being returned on the `CBTerminalTapListener` interface.
For additional information regarding `Biometric and CBTerminalTapListener` , please consult our SDK documentation.

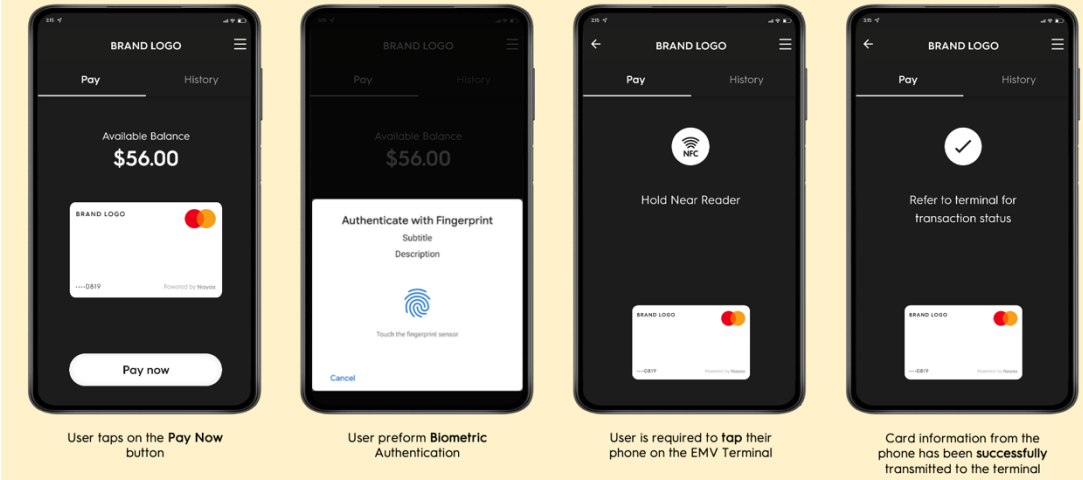
⚙ TECHNICAL INFO: When your user is showing intent to pay (i.e., taps the Pay button) – after calling the Android operating system Biometric Prompt your application should call the `PrepareToPay` method.
A successful response from this method (`ReadyToPay`) will automatically open the NFC tool of the Android device to allow the user to pay via the EMV terminal.

⚠ NOTE: To enable your application (or any payment application) to perform payments, the relevant NFC settings must be enabled within your user's Android device. These permissions include the ability to pay using NFC, as well as (in some cases) the ability to pay using a currently open application. For further details on these permissions, please refer to the relevant section of the Appendix to this document.

The payment journey contains a few steps and SDK calls, as set out in the flows below.

## Scenario A: Tap Completed (Happy Flow):

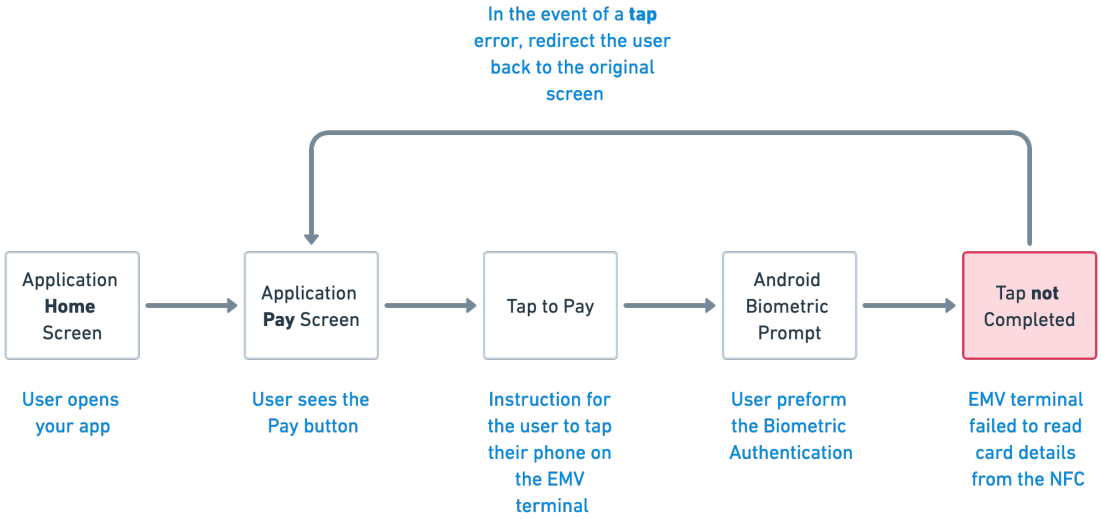


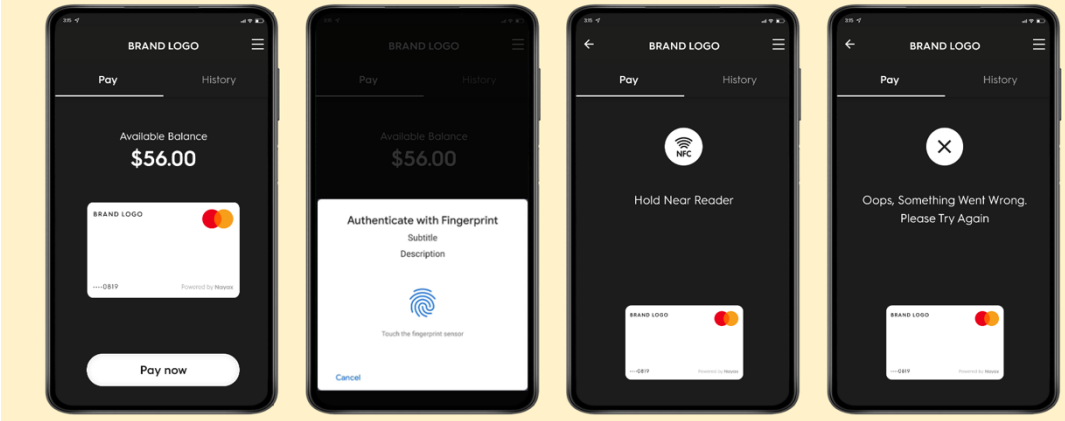




**Tap complete** is the end of the tap process. However, a successful flow does not indicate that the transaction has been approved, only that information from the app was successfully transmitted to the EMV terminal via the device's NFC.

**Transaction Resolution** is the outcome of the payment flow (Approved / Declined) – more details on this can be found in the next chapter.

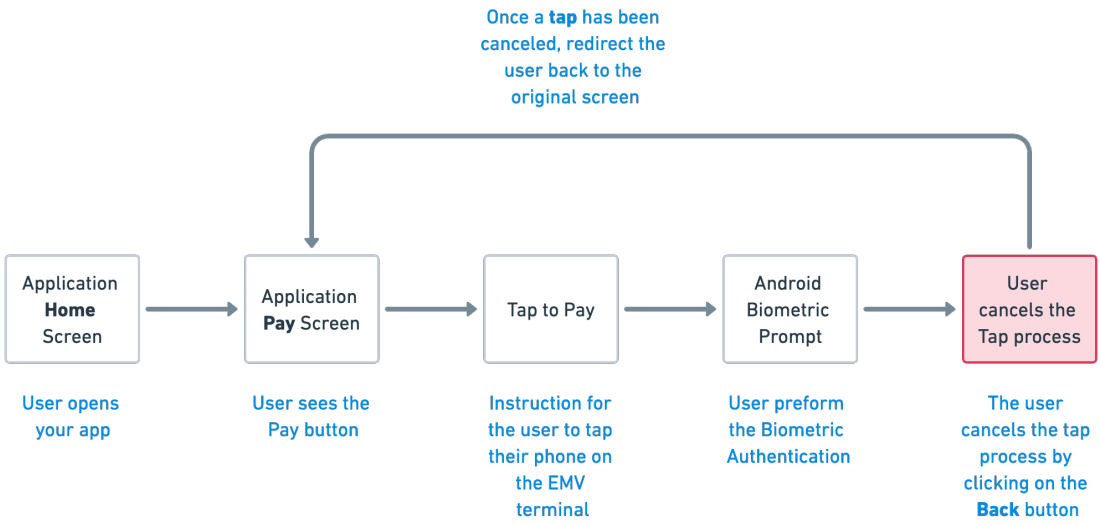
## Scenario B: Tap Not Completed

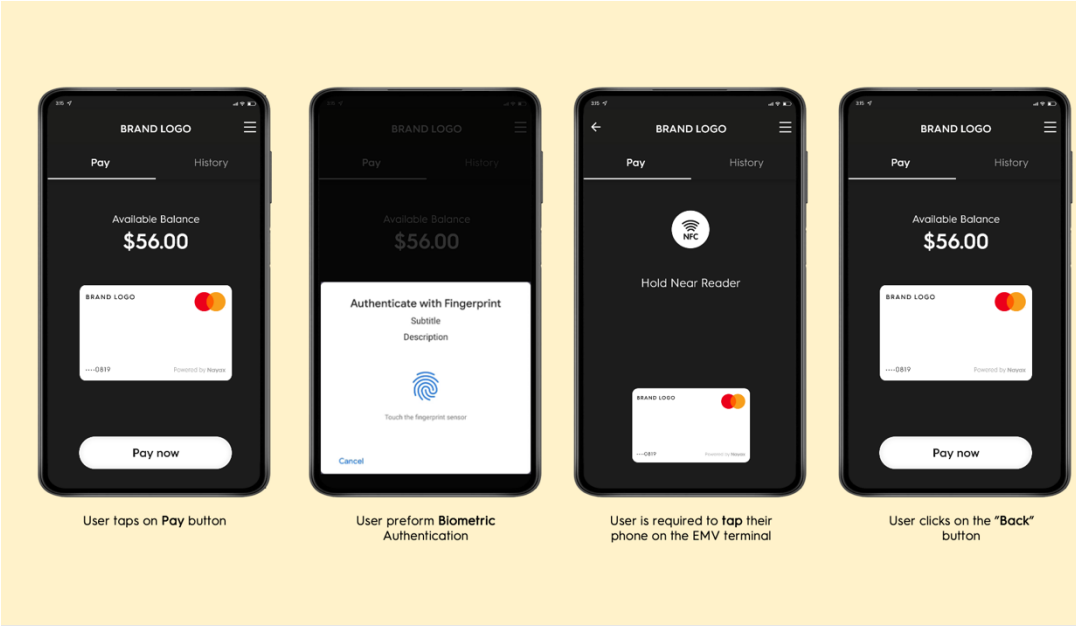



**Tap not completed** does not indicate that the transaction was declined; it indicates that something went wrong during the communication flow between the device and the EMV terminal.

## Scenario C: Payment Action Canceled by User




When the SDK is prepared for payment and expects an EMV-over-NFC transaction (before the payment timeout is reached), there's sometimes a need for the app to abort the process.
This can happen if the user has aborted the Hold near Reader screen (by pressing Back for example), or your application decided that it shouldn't allow the user to perform a payment attempt at this stage.

In this case, the application must call the `stopPayment` method from the payment activity.

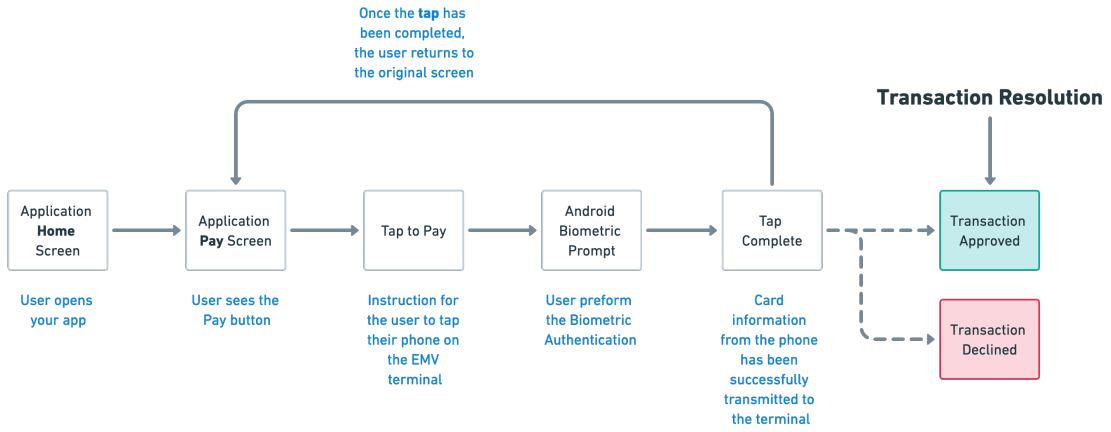
After the call to `stopPayment`, the CoinBridge SDK will no longer be prepared for payment (and tapping an EMV terminal won't result in communication to the EMV terminal).

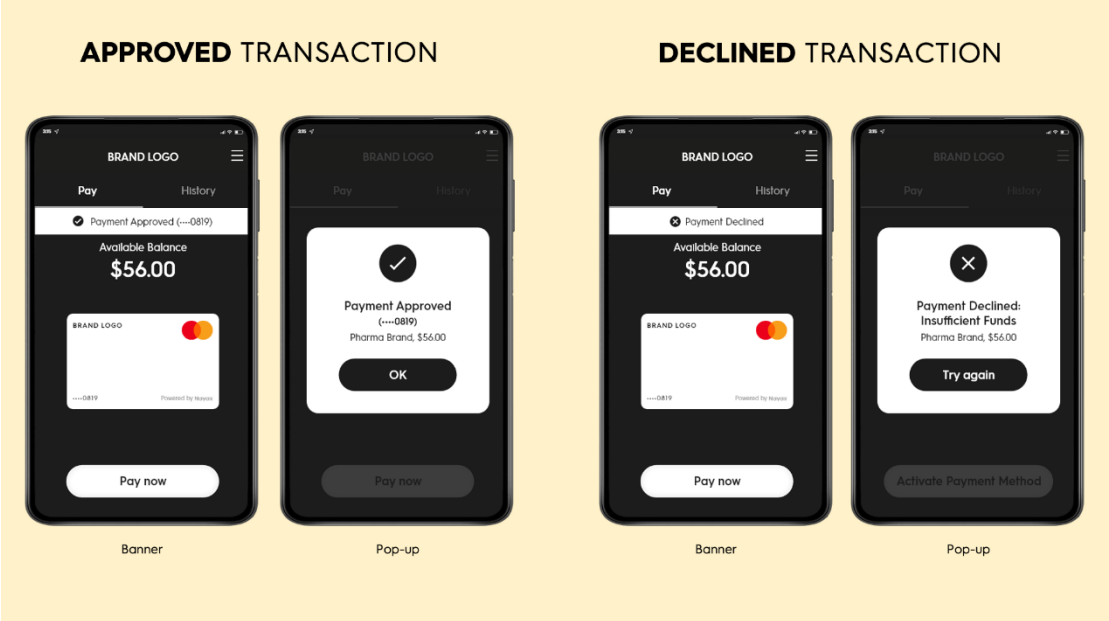
In order to get the SDK prepared after calling `stopPayment`, the payment flow should be restarted as a new payment flow (for more details, refer to our SDK documentation).

# Transaction Resolution and History

## General

The transaction result and all information related to the transaction will be communicated to your loyalty/ user management platform via **CoinBridge's Back-End Connector**.




After a transaction attempt has been performed via your application, the merchant's EMV terminal will communicate with the Visa or Mastercard network to authorize the transaction request.

CoinBridge's Back-End Connector will receive a transaction authorization request (from Visa or Mastercard's card network).

CoinBridge's Back-End Connector will compare the transaction details with the user's available balance and spending policies.

Based on the above (and additional extensive account, fraud and risk related verifications), the CoinBridge Back-End Connector will determine whether to approve or decline the transaction attempt.

The decision (with additional supportive data) will be communicated to your loyalty or user management platform via CoinBridge's Back-End Connector.

**Once a transaction is completed, the user must receive proper feedback based on the transaction resolution.**

The transaction result (**approved** or **declined**) must be shown to the user and must contain the following:

In the case of an **approved** transaction:

- Indication that the transaction was approved.
- Additional transactional data (transaction amount, merchant name, etc.)

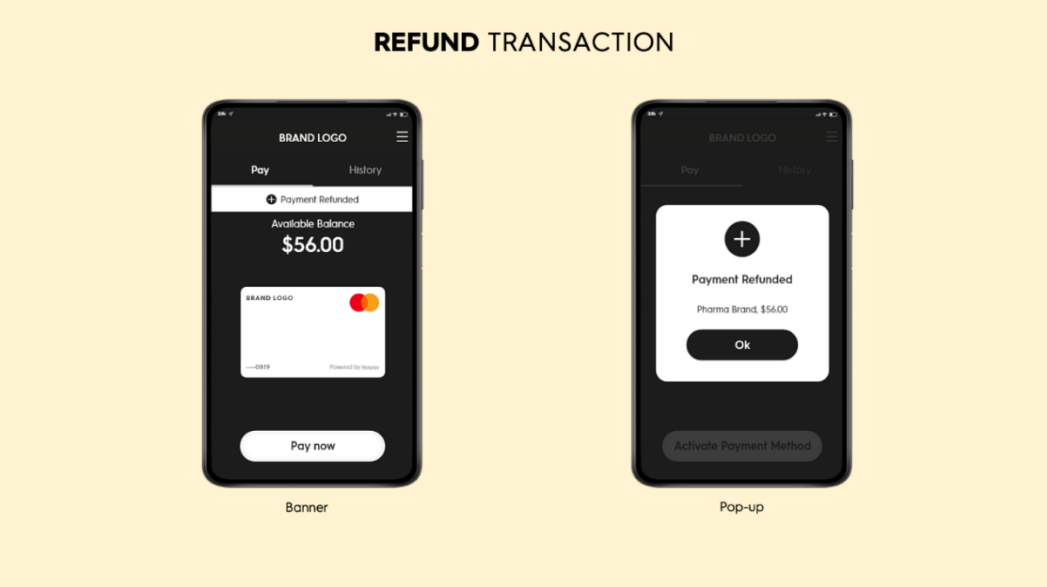
In the case of a **declined** transaction:

- Indication that the transaction was declined.
- Reason(s) for declining the transaction, using the data communicated from the CoinBridge Back-End Connector to your platform (see relevant section of the Appendix to this document for further information regarding decline reasons).

It is crucial to use push notifications in addition to pop-ups to notify the user of the transaction resolution.

## Additional Transaction Reporting Types

**Transaction Refund** refers to a case where the user has been refunded by the merchant.



The refund transaction will appear as an independent transaction, with a unique transaction ID, and won't be connected to any historical transaction that may exist.

As in a regular transaction, the refund amount will always have a non-negative value, with a minimum value of zero, but in this case, you will have to top-up the user's balance accordingly.

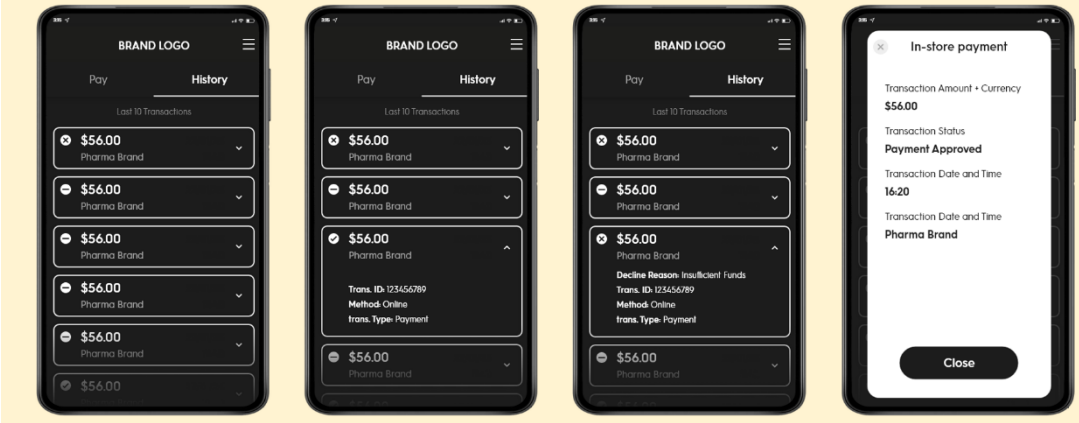
For more information about refund transactions, please see the **CoinBridge Back-End Connector Manual**.

In the event of a refund transaction, we suggest you show an appropriate message in the **Transaction History** (see the following chapter), such as: "You have been refunded the amount of [xx] points."

**Transaction Update** refers to an update sent to your application regarding a case in which we sent a transaction for approval that actually failed due to a card network error (the user saw it failed but we reported this transaction as a success).

For more information about this transaction reporting type, please see our dedicated CoinBridge Back-End Connector documentation.

## Transaction History




Transaction History List

Transaction History (Approved Transaction Details)

Transaction History (Declined Transaction Details)

Transaction History Specifics (Details of a specific Declined Transaction)

**It is important for users to be able to access their transaction history.**

We recommend creating a dedicated transaction history screen to display end-user payment actions and provide end-users with additional information for each transaction.
In most cases, a payment/ transaction history module already exists within your application. If so, only minor adjustments are required.

⚠ NOTE: The user must be able to differentiate between approved and declined transactions.

⚠ NOTE: In the event of declined transactions, it is crucial to display the decline reason to the user.

CoinBridge will provide you with detailed information about the transaction once it has been completed. Please note that the transaction details in the examples above are for illustration purposes only.

## User Logout

This section covers what needs to happen when a user logs out of your application.
It's crucial to quickly let the CoinBridge SDK know when this happens.

Initiating the logout method within the SDK triggers the systematic clearance of all user-specific payment data. This action ensures that the SDK is left in a pristine state, ready for the subsequent login sequence.

⚠ NOTE: It's really important to do this. If you forget to inform the CoinBridge SDK when a user logs out, it could cause issues with how you identify users and track transactions, especially if the next person logging in is different from the previous one.

# Appendix

## SDK Mandatory Permissions

You will need to make sure that the CoinBridge SDK is granted all the mandatory permissions to function:

- `android.permission.INTERNET`
  This permission is required to communicate with the CoinBridge back-end platform.
- `android.permission.ACCESS_NETWORK_STATE`
  This permission is required to properly schedule back-end synchronization tasks.
- `android.permission.READ_PHONE_STATE`
  This permission is required to properly manage the SDK state.
- `android.permission.RECEIVE_BOOT_COMPLETED`
  This permission is required to perform background processing.
- `android.permission.USE_BIOMETRIC`
  This permission is required to carry out biometric authentications on android 8+ devices.

**Additional Information:**

1. None of the above required permissions are user permissions, so no interaction with the user is required for your app to obtain them.
2. For a more in-depth explanation regarding mandatory permissions, please refer to our SDK documentation.

⚠ NOTE: If any one of the above permissions is not available for the CoinBridge SDK, you should not offer this feature to the user, or advise the user to use a different device.

## Initialization Method

Initialization method is an asynchronous operation that may take a few seconds to complete.

each time your application starts, but only after the SDK has been successfully authenticated (when the IsAuthenticated call returns **True**).

It is strongly advised to perform SDK initialization in the onCreate() method of the payment activity. Once the initialization process wraps up, the OnInitCallback will hold details about the user's payment method readiness state, incorporating the "HasCard" and "CBPaymentStatus" classes.

If the SDK integration has been completed correctly, the initialization method should be failproof.

For more technical information, please refer to our SDK documentation.

Please see below a list of errors that may occur:

| Initialization Call Error Values | Error Description |
|---|---|
| SDK_NOT_AUTHENTICATED | The CoinBridge SDK is not authenticated, SDK should first authenticate before initialization |
| DEVICE_NOT_ELIGIBLE | The device is not eligible for payment (this may be because the device is not NFC equipped or is rooted, etc.) |
| PERMISSIONS_NOT_GRANTED | The CoinBridge SDK did not receive one or more of the mandatory permissions |
| INIT_ALREADY_STARTED | Not an error!<br>Please wait for a corresponding callback |
| NETWORK_ERROR | The CoinBridge SDK could not access the network. This may be because there is no internet connectivity, or because the CoinBridge SDK lacks permission to access the network |
| GENERAL_ERROR | An unknown error has occurred. Please contact CoinBridge support for further information |

## Authentication Method

Authentication, commonly referred to as Auth, is a critical process that involves synchronizing the unique identifier of a user with the CoinBridge SDK and the CoinBridge Back-End Connector.

It is important to note that this method must be called only when required, namely when the IsAuthenticated call returns False.

Additionally, the authentication call should contain the user's specific identifier, which will later be used for communication between the CoinBridge Back-End Connector and your loyalty or user management platform.

For further information, please refer to the SDK documentation provided by CoinBridge.

| Authentication Call Error Values | Error Description |
|---|---|
| `INVALID_AUTHENTICATION` | Authentication failed. This may happen due to an invalid User ID |
| `ALREADY_AUTHENTICATED` | Not an error!<br>Please action this as a successful call – Next step is to call Initialization method |
| `NETWORK_ERROR` | The CoinBridge SDK could not access the network. This may be because there is no internet connectivity, or because the CoinBridge SDK lacks permission to access the network |
| `GENERAL_ERROR` | An unknown error has occurred. Please contact CoinBridge support for further information |

## PaymentMethodStatus Method

### General

The process for creating the CoinBridge payment method is comprised of two main flows that must take place in order for the user to be able to perform payments:

- Virtual card issuing – Issuing a virtual card for the specific user.
- Virtual card provisioning - Adding the token of this virtual card to the CoinBridge SDK (integrated within the specific instance of the mobile application).

⚠ NOTE: The above processes are performed by the CoinBridge SDK, in the background. Please make sure to call the Provision card method to provision the virtual card.

### Timing

Once the **Initialization** flow has been successfully completed, you will be notified of the CoinBridge **Virtual Card** issuing (HasCard status & provisioning status).
At this point, CoinBridge will initiate card issuing automatically.

⚠ NOTE: The virtual card creation process will be initiated automatically (by CoinBridge), or manually (by the user, using the `CreatePaymentMethod` method), while the Provision process will always be triggered by calling the Provision card Method.
Please consult the CoinBridge onboarding team to find out which flow applies to your application.

### Status of Payment Method and Readiness to Perform Payments

The payment method (card) status will be provided by the CoinBridge SDK in the following scenarios:

- Following successful (completed) initialization.
- Following successful authentication.
- In the event of an update in card payment method status, for example the card status has changed, the last 4 digits of the card are updated, or the card has been provisioned.

The payment method status (`CBPaymentStatus`) will always include the following information:

| Response Component | Response Description | Possible answers |
|---|---|---|
| `CBCardStatus` | Virtual Card Status | - **Active**<br>- **Unavailable** |
| `lastfourDigits` | Last 4 Digits | - **Empty**: there is no card number<br>- **4 Digits**: the last 4 digits to display on the card |
| `isProvisioned` | Card In Wallet | **- True:** When the card is in the wallet, it indicates that the user & device can be used to perform payments<br>**- False**: When the card is not yet in the wallet |

# PaymentMethodIssuing Method

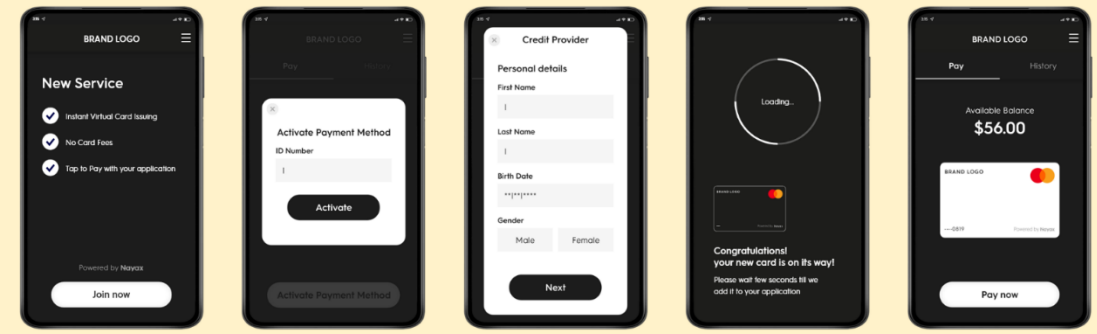
## Manual vs. Automatic Issuing of Payment Method

Based on the regulations in your country, and as agreed in the MOU, the CoinBridge payment method may be issued either automatically or manually.

- **Automatic** – CoinBridge will create the payment method and push it to the SDK wallet in the background, without any action being required from the user side.
- **Manual** – The user will have to initiate the issuing process, and may have to provide certain details in order to complete the creation of the payment method.

⚠ NOTE: The manual process may require your user to insert some personal details, and most may not understand why they need to provide personal information. Please make sure to properly onboard the user and engage them with the new CoinBridge feature to ensure a high onboarding rate.

## Manual Virtual Card Issuing Flow (using createPaymentMethod)




This method should be used only in the event that **all** of the following applies:

1. Your application is configured to support manual virtual card issuing.
2. The user has successfully completed Initialization & Authentication.
3. `initiCompleted(HasCard = False)`

In some cases, manual card issuing may require additional user-specific information to be provided (based on the regulations/ financial authority in your country):

1. In such cases, please make sure to provide this information as part of the `createPaymentMethod` payload.
2. In the event that your application does not have this information, make sure to ask for it from your user in an appropriate manner (i.e., explain why this information is needed after introducing the new service).
3. User-specific information Examples: SSN / National ID number.

Based on the card issuing flow, a successful `createPaymentMethod` may require the following (managed by the CoinBridge SDK):

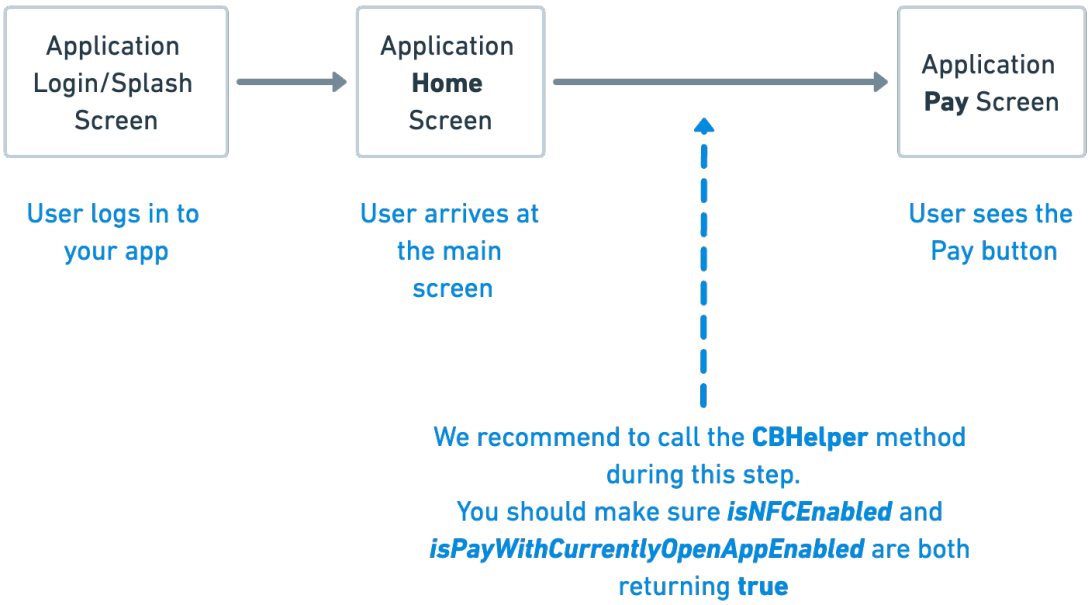
1. User to provide additional details via a dedicated card-issuer form.
2. User to consent to additional terms (based on issuer/ regulations).

A dedicated **Success** `createPaymentMethod` callback will be returned once the manual virtual card issuing flow has been finalized.

From this moment, the user should be notified, and flow should continue as explained above in the Completing the Pre-Payment Journey section.

## Android NFC / Contactless Payments Settings Validation

(Using the CoinBridge Helper toolkit)




To enable payments using your mobile application, the CoinBridge SDK requires that user's device to have the following settings enabled (Android):

1. **NFC Capability** setting.
2. **Pay with open app** setting.

As part of the services provided by CoinBridge SDK, before allowing your user to perform a payment (by calling the `prepareToPay` method or enabling your user to tap the **Pay** button), your app should call the relevant CBHelper methods to ensure that both **NFC** and **Pay with open app** flags are enabled. (Further details can be found in the CoinBridge SDK documentation).

**The CoinBridge Helper provides the following relevant tools:**

1. `isNFCEnabled` – Checks whether the NFC setting on the end-user's device is enabled or disabled.
2. `LaunchNfcSettings` – Redirects the end-user to the relevant NFC settings page, if the NFC capability is currently disabled, so they can enable it.
3. `IsPayWithCurrentlyOpenAppEnabled` – Checks whether the **Pay with currently open app** setting is enabled or disabled.
4. `LaunchContactlessPaymentsSettings` - Redirects the end-user to the **pay with currently opened app** settings page, if it is currently disabled, so they can enable it.

⚠ NOTE: These settings are crucial. If they are disabled, the default payment method will be triggered in all cases where the user holds his mobile device near an EMV terminal.

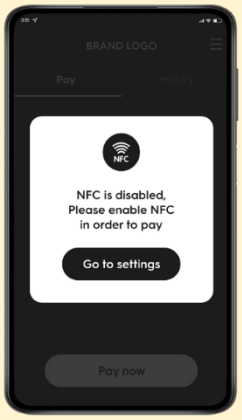
⚠ NOTE: The end-user does not need to set your application as the default payment method.

If either one of these settings is set to **disabled**, you must advise the end-user to enable them before moving on to the Payment Activity.

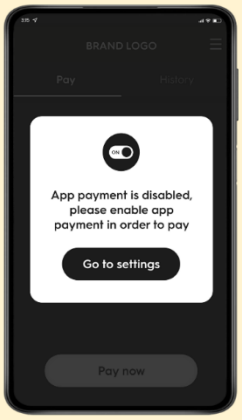
It is crucial to call tools 1 & 3 above every time the user enters your application's main payments screen (and act accordingly in the event that one of these settings is disabled).

For further information regarding these configurations, please contact the CoinBridge Product Team.

Please see below some design examples for pop-ups instructing end-users to enable the relevant settings.




Enable **NFC**
Pop-up Design




Enable **Pay with open App**
Pop-up Design

## Transaction Resolution and History

### Reasons for Transaction Decline

Whenever a transaction is declined, CoinBridge's Back-End Connector will transmit a decline reason code and a description to your loyalty/ user management platform.

A decline reason code provides a high-level explanation for the decline, while a decline reason description provides a more detailed description and may include parameters (shown below in brackets) relevant to the declined transaction.

Whatever the decline reason transmitted by the CoinBridge Back-End Connector, it is important that your app represents your own messaging syntax aimed at your uses.

Please see below the list of our standard decline reasons (as a function of your spending policies):

| Decline Reason Code | Decline Reason Description |
|---|---|
| `CB_SYSTEM_ERROR` | Unexpected CoinBridge Error |
| `INSUFFICIENT_END_USER_FUNDS` | Transaction Amount {`Transaction_Amount` & `Transaction_Currency`} exceeds end-user's available balance |
| `INSUFFICIENT_APPLICATION_FUNDS` | Transaction amount {`Transaction_Amount` & `Transaction_Currency`} exceeds application's available balance |
| `MCC_NOT_ALLOWED` | This MCC is a restricted {`MCC_Code`} under this policy |
| `MID_NOT_ALLOWED` | This MID is restricted {`MID_Number`} under this policy |
| `COUNTRY_NOT_ALLOWED` | This country is restricted {`Country_Code`} under this policy |
| `AMOUNT_BELOW_THRESHOLD` | Transaction amount {`Transaction_Amount` & `Transaction_Currency`} is below the minimum allowed amount |
| `AMOUNT_EXCEEDS_THRESHOLD` | Transaction amount {`Transaction_Amount` & `Transaction_Currency`} exceeds the maximum allowed amount |

⚠ NOTE: Additional decline reasons not shown in the above table do exist, in the event of specific dedicated spending policies that you have chosen to define. In such cases, CoinBridge Support will provide the specific Code and Description.

## Card Art Style Guide

All design elements can be found via this link: [Card Art Design Elements]