Geidea Gateway

iOS SDK Integration Guide

Version 1.0

|  |  |
| --- | --- |
| Document version: | 1.0 |
| Date of version: | 24-03-2021 |
| Author: | Eugen Vidolman |

|  |
| --- |
|  |

Version History

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Changes |
| 1.0 | 24.03.2021 | Eugen Vidolman | Initial version |

|  |  |
| --- | --- |
|  |  |

Contents

[1. Introduction 3](#_Toc67583100)

[1.1 Document Purpose 3](#_Toc67583101)

[2. iOS Integration 4](#_Toc67583102)

[2.1 How to start 4](#_Toc67583103)

[2.1.1 Connect iOS framework: 4](#_Toc67583104)

[2.1.2 Initialize Object: 5](#_Toc67583105)

[2.1.3 Initialize Payment flow using Geidea SDK Payment objects: 6](#_Toc67583106)

[2.1.4 Start payment flow using GeideaPaymentApi.pay function 9](#_Toc67583107)

[2.1.5 Start payment flow using Geidea Form 11](#_Toc67583108)

[2.1.6 Get Filtered Orders, Get Order By id. Optional for Mobile SDK: 13](#_Toc67583109)

[2.1.7 Capture Order, Refund Order and Cancel Order Flow: 14](#_Toc67583110)

[2.1.7.1 Capture Order: 14](#_Toc67583111)

[2.1.7.2 Refund Order (Optional for Mobile SDK): 15](#_Toc67583112)

[2.1.7.3 Cancel Order (Optional for Mobile SDK): 16](#_Toc67583113)

[2.1.8 Pay Token Flow: 17](#_Toc67583114)

[2.1.9 Apple Pay Payment Flow: 19](#_Toc67583115)

[2.1.10 Get Card scheme logo detector flow: 21](#_Toc67583116)

[3.1.10 EInvoice Create, Update, Get and Delete operations: 21](#_Toc67583117)

[3.1.12 SDK Responses from SDK Payment flow: 22](#_Toc67583118)

[2.1.13 SDK Debug Logging System: 24](#_Toc67583119)

[2.2 Field specification (data dictionary) 25](#_Toc67583120)

[2.3 Response codes 26](#_Toc67583121)

[2.3.1 iOS SDK Response codes and messages: 26](#_Toc67583122)

[2.3.2 Geidea Payment Response codes and messages: 28](#_Toc67583123)

[Appendix A. Glossary 33](#_Toc67583124)

# Introduction

## Document Purpose

The purpose of the iOS Software Development Kit (iOS SDK) Integration guide is to serve as a technical documentation for merchants that want to integrate with Geidea and want to use the Payment Gateway services with their systems.

When merchants integrate with the iOS SDK they will be able to send parameters from their iOS App when a client is triggering a payment and visualize Geidea Payment SDK for their clients to use to process an online payment.

# iOS Integration

SDK Requirements:

Minimum iOS version: 11.0

- Swift 4.0, 4.2, 5.X

- Objective C

## How to start

### Connect iOS framework:

To connect iOS framework to your iOS application you should:

1. Drag GeideaPaymentSDK.XCframework to your Frameworks folder (Contains both devices and simulator) – Preferred

OR Drag GeideaPaymentSDK.framework to your Frameworks folder (need devices and Fat framework)

1. Add it your target: General -> Frameworks, Libraries and and Embedded Content.
2. Choose “Embed & Sign” option on Embed tab

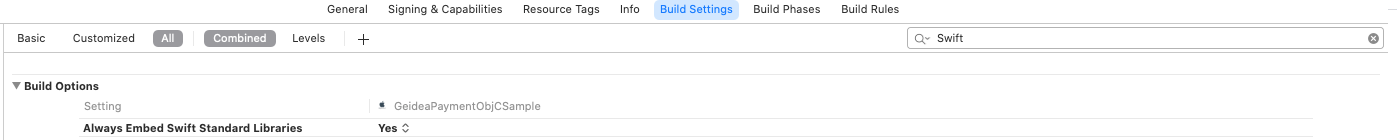
Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

1. If your application is Objective C app perform an additional step: Build settings -> Build Options -> Always Embed Swift Standard Libraries set YES



For using iOS SDK in code remember to import the framework in your existing app code:

Example Swift:

**import** GeideaPaymentSDK

Example Objective C:

#import <GeideaPaymentSDK/GeideaPaymentSDK.h>

### Initialize Object:

At your application start up (for example in the AppDelegate) or when a client hits Pay button on your app the SDK authentication must be done.

1. Check if the credentials are available on SDK secure storage
2. Authenticate with SDK using setCredentials
3. Get merchant configuration from Geidea server
4. Store the configuration for future use

The function GeideaPaymentApi.setCredentials() should have the following parameters:

* **merchantKey**: merchantID assigned to you by Geidea
* **password**: password assigned to you by Geidea

The function GeideaPaymentApi.getMerchantConfig should have the following parameters:

* completion: GDConfigResponse, GDErrorErrorResponse

A success completion GDConfigResponse have the following properties:

* **GDConfigResponse**: Merchant Configuration object: Required
  + - **hppDefaultTimeOut**: Double = 1800
    - **countries**: [ConfigCountriesResponse] - list of countries supported by your mercahantID
    - **is3dsV2Enabled**: Bool - if false payment Geidea Payment SDK will use 3dsV1
    - **isApplePayPaySupported**: Bool - if true you can use applePay functionality
    - **isTokenizationEnabled**: Bool – If true you can set cardOnFile parameter to true for allowing users to pay with an already tokenized card
    - **name**: String – the name of the merchant that will be used on Apple Pay payment
    - **paymentMethods**: [String] – a list of supported paymentMethods e.g. [visa, mastercard, mada, amex,]
    - **allowedInitiatedByValues**: [Sting] - a list of allowed initiated by values, e.g [Internet, Merchant]

Example Swift:

if !GeideaPaymentApi.isCredentialsAvailable() {

GeideaPaymentAPI.setCredentials(withMerchantKey: “merchantKey” andPassword: “password”)

}

GeideaPaymentAPI.getMerchantConfig(completion:{ GDConfigResponse, error **in**

**guard** **let** config = response **else** {

**return**

}

// Store GDConfigResponse for future use

})

Example Objective C:

if (![GeideaPaymentAPI isCredentialsAvailable]) {

[GeideaPaymentAPI setCredentialsWithMerchantKey: @"merchantKey" andPassword: @"password"];

}

[GeideaPaymentAPI getMerchantConfigWithCompletion:^(GDConfigResponse\* config, GDErrorResponse\* error) {

**if** (config != **NULL**) {

}

// Store GDConfigResponse for future use

}];

### Initialize Payment flow using Geidea SDK Payment objects:

The function GeideaPaymentApi.pay should have the following parameters:

* **GDAmount**: SDK Amount object: Required
  + - **amount**: Double (Required) -maximum 2 decimals
    - **currency**: String (Required) - 3 letters and must be one of the approved currencies
* **GDCardDetails**: SDK Card Details object: Required
  + - **cardHolderName**: String (Required) – maximum 255 characters
    - **cardNumber**: String (Required)
    - **cvv**: String (Required) – 3 or 4 (if amex) digits
    - **expiryMonth**: Int (Required) - digits from 1 to 12
    - **expiryYear**: Int (Required) – digits from 1 to 99
* **GDTokenizationDetails**: SDK Tokenization Details object: Optional
  + - **cardOnFile**: Bool (Optional) – used for card tokenization: if true the following optional parameter are required.
    - **initiadedBy**: String (Optional) (Required if cardOnFile true) – when starting the tokenization process must be the following value:
      * + **Internet**
    - **agreementId**: String – can be any value (can be non-unique value) or **NIL**
    - **agreementType**: String – Must be one of the following values or **NIL if** agreementId **is NIL** and if there will be NO subscription payments in the future
      * + **Recurring**
        + **Installment**
        + **Unscheduled**
* **EInvoiceId**: String The parameter used to pay the already created EInvoice: Optional
* **GDCustomerDetails**: SDK Customer Details object used for customer reference: Optional
  + - **customerEmail**: String (Optional)
    - **merchantReferenceID**: String (Optional)
    - **callbackUrl**: String (Optional)
    - **paymentOperation**: PaymentOperation (Optional) – Can be one of the following enum values, if nil the merchant configuration will be used for payment:
      * + **pay** – default Payment, amount will be automatically paid to merchant. The **GDOrderResponse** parameter **detailedStatus** will pe “Paid” on success
        + **preAuthorize** – payment will be just Authorized, not paid. For finishing the payment, you must use the **Capture** API (**GeideaPaymentAPI.capture**) later for capturing the amount with a saved orderId. The **GDOrderResponse** parameter **detailedStatus** will pe “Authorized” on success and **orderId** value must pe persist to finish the flow.
        + **AuthorizeCapture** – authorized Payment, amount will be automatically paid to merchant. The **GDOrderResponse** parameter **detailedStatus** will pe “Captured” on success
        + **none** – same as NIL, merchant configuration will be used for payment
    - **shippingAddress**: GDAddress (Optional)

GDAddress:

* + - * + **countryCode**: String (Optional) – 3 letters
        + **city**: String (Optional)
        + **street**: String (Optional)
        + **postCode**: String (Optional)
    - **billingAddress**: GDAddress (Optional)

GDAddress:

* + - * + **countryCode**: String (Optional) – 3 letters
        + **city**: String (Optional)
        + **street**: String (Optional)
        + **postCode**: String (Optional
* **navController**: UIViewController used for presenting SDK Payment Flow (Required)
  + - can be your UIViewController self SDK flow will be presented modally
    - can be your UINavigationController navigationController SDK flow will be pushed from your navitation controller
* **completion**: (GDOrderResponse?, GDErrorResponse?) -? Void used for returning success **GDOrderResponse**, or failure **GDOrderResponse** to your application

Example parameters Swift:

let amount = GDAmount(amount: 23.10, currency: “SAR“))

let cardDetails = GDCardDetails(withCardHolderName: “name”, andCardNumber: “4111111111111111“, andCVV: “111”, andExpiryMonth: 12, andExpiryYear: 23, andCardOnFile: true or false, initiatedBy: “Internet” - can be nil if cardOnFIle is false, agreementId: String - anything can be non-unique value or nil, agreementType: String – can be one of the following: “Recurring”, “Installment”, “Rescheduled” or nil if agreementId is nil ))))

let tokenizationDetails = GDTokenizationDetails(withCardOnFile: false, initiatedBy: "Internet " or "Merchant", agreementId: someString, agreementType: someString)

let shippingAddress = GDAddress(withCountryCode: “some country Code ex: SAU”, andCity: “some City“, andStreet: “some street address”, andPostCode: “some postCode address”))

let billingAddress = GDAddress(withCountryCode: “some country Code ex: SAU”, andCity: “some City“, andStreet: “some street address”, andPostCode: “some postCode address”))

let customerDetails = GDCustomerDetails (withEmail: “valid email address”, andCallbaclUrl: “valid url “, merchantReferenceID: “your reference id”, paymentOperation: PaymentOperation, shippingAddress: shippingAddress, billingAddress: billingAddress))

Example parameters Objective C:

GDAmount \*amount = [[GDAmount alloc] initWithAmount: 23.10 currency: @”SAR”]

GDCardDetails \*cardDetails = [[GDCardDetails alloc] initWithCardholderName: @”name” andCardNumber: @“4111111111111111” andCVV: @”111”andExpiryMonth: 12 andExpiryYear: 23];

GDTokenizationDetails \*tokenizationDetails = [[GDTokenizationDetails alloc] initWithCardOnFile:Bool initiatedBy: NSString agreementId:NSString agreementType:NSString];

GDAddress \*shippingAddress = [[GDAddress alloc] initWithCountryCode: @”some country Code ex: SAU” andCity:@”some City” andStreet: @”some address” andPostCode:@”some postalCode”];

GDAddress \*billingAddress = [[GDAddress alloc] initWithCountryCode: @”some country Code ex: SAU” andCity:@”some City” andStreet: @”some address” andPostCode:@”some postalCode”];

GDCustomerDetails \*customerDetails = [[GDCustomerDetails alloc] initWithEmail: @”valid email” andCallbackUrl: @”validURL” merchantReferenceId: @” your reference id” paymentOperation: PaymentOperationPay shippingAddress: shippingAddress billingAddress: billingAddress];

### Start payment flow using GeideaPaymentApi.pay function

When the user clicked on the pay button from your application you should call the payment function: GeideaPaymentApi.pay GeideaPaymentApi.pay or GeideaPaymentApi.payWithGeideaForm with the above parameters described on section 3.1.3.

- Override **paymentOperation:** If the optional parameter paymentOperation is not NIL and is preAuthorize, an additional step will be required for completing the flow: 3.1.5 Capture flow

- **Tokenization**:

From persisted **GDConfigResponse** check the flag **isTokenisationEnabled** before starting the flow and showing the save card button

If the **cardOnFile** is set to true, in the **GDOrderResponse** a new field: **tokenId** will be computed. The merchant needs to persist this value together with the card identifier for future payments with card Token and need to save agreementId and agreementType if there will be subscription payments in the future: see **3.1.6 Pay with Token.**

Example Swift:

GeideaPaymentAPI.pay(theAmount: amount, withCardDetails: cardDetails, andTokenizationDetails: tokenizationDetails, andEInvoice: eInvoice, andCustomerDetails: customerDetails, navController: \*\*navVC\*\* or \*\*self\*\*, completion:{ response, error in {

DispatchQueue.main.async {

If let err = error {

If err.errors.isEmpty {

//TODO: display relevant fields from GDErrorResponse as title, responseMessage, responseCode, detailedResponseMessage, detailedResponseCode: String

} else {

//TODO: display relevant fields from GDErrorResponse as title, errors

}

} else {

guard let orderResponse = response else {

return

}

//TODO: Display relevant fields from GDOrderResponse

//TODO: if cardOnFile is true, save tokenId from GDOrderResponse in persistence and agreementId and agreementType for future subscriptions

//TODO: if paymentOperation is PaymentOperation.preAuthorize:

// Save order id in persistence for capturing the payment with

// GeideaPaymentApi.capture

// if orderResponse.detailedStatus == "Authorized" {

// TODO: save order.orderId

// TODO: show capture button on the UID

// }

}

}

})

Example Objective C:

[GeideaPaymentAPI payWithTheAmount:amount withCardDetails:cardDetails andTokenizationDetails: tokenizationDetails andEInvoiceId: eInvoiceId andCustomerDetails:customerDetails dismissAction:NULL navController: \*\*navVC\*\* or \*\*self\*\* completion:^(GDOrderResponse\* order, GDErrorResponse\* error) {

if (error != NULL) {

if (!error.errors || !error.errors.count) {

//TODO: display relevant fields from GDErrorResponse as title, responseMessage, responseCode, detailedResponseMessage, detailedResponseCode: String

} else {

//TODO: display relevant fields from GDErrorResponse as title, errors

}

} else {

if (order != NULL) {

//TODO: display relevant fields from GDOrderResponse

//TODO: display relevant fields from GDOrderResponse

//TODO:

if cardOnFile is true save tokenId from GDOrderResponse in persistence and agreementId and agreementType for subscriptions (InitistedBy = "Merchant")

// TODO: check [order tokenId] != NULL

//TODO: if paymentOperation is PaymentOperation.preAuthorize:

// TODO: Save order id in persistence for capturing the payment with GeideaPaymentApi.capture

if ([[order detailedStatus] isEqual: @"Authorized"]) {

// TODO: save order.orderId

// TODO: show capture button on the UID

}

}

}

}];

### Start payment flow using Geidea Form

If you want to use the Geidea Payment Form for completing the payment you can use the SDK GeideaPaymentApi.payWithGeideaForm. Geidea form includes UI for card details, customer details and Apple Pay embedded, and some optional details that you might want to pass. This replaces the need to use GeideaPaymentApi.pay function when user clicked pay button on the app.

Geidea Form has some features useful for the user:

* + - All fields validations with error messages
    - Embedded feedback messages for API failures
    - UITextField navigation with next button and Done
    - Adaptive UI based on user input or merchant input
    - GDOrderReponse and GDErrorResponse passed in completion for unrecovered errors inside the Form.

There are some optional parameters that you need to input for enabling some features:

1. If you want to use Apple Pay feature you have to pass Apple Pay details. ApplePay Buttton Placeholder and the host UIViewController are not required in this case, because it is inherited by Geidea Form

**GDApplePayDetails**: necessary details for Apple Pay UI and request (Optional)

* + **ApplePayMerchantId**: Your apple account merchantId from the apple pay setup (Required)
  + **GDAmount**: SDK Amount object: Required
    - **amount**: Double (Required) -maximum 2 decimals
    - **currency**: String (Required) - 3 letters and must be one of the approved currencies

2. If you need the Form to display the email and Address you should send showEmail, and showAddress to true.

3. If you send customerDetails already filled (GDCustomerDetails object), the form will be prefilled with the values sent, otherwise will be empty

4. Send the GDTokenizationDetails if you want to enable card tokenization feature see section 2.1.8 Pay With Token for details

Example Swift:

GeideaPaymentAPI.payWithGeideaForm(theAmount: amount, showAddress: showAddressSwitch.isOn, showEmail: showEmailSwitch.isOn, tokenizationDetails: tokenizationDetails, customerDetails: customerDetails, applePayDetails: applePayDetails, config: **self**.merchantConfig, eInvoiceId: eInvoiceId, viewController: **self**, completion:{ response, error **in**

DispatchQueue.main.async { DispatchQueue.main.async {

If let err = error {

If err.errors.isEmpty {

//TODO: display relevant fields from GDErrorResponse as title, responseMessage, responseCode, detailedResponseMessage, detailedResponseCode: String

} else {

//TODO: display relevant fields from GDErrorResponse as title, errors

}

} else {

guard let orderResponse = response else {

return

}

//TODO: Display relevant fields from GDOrderResponse

//TODO: if cardOnFile is true, save tokenId from GDOrderResponse in persistence and agreementId and agreementType for future subscriptions

//TODO: if paymentOperation is PaymentOperation.preAuthorize:

// Save order id in persistence for capturing the payment with

// GeideaPaymentApi.capture

// if orderResponse.detailedStatus == "Authorized" {

// TODO: save order.orderId

// TODO: show capture button on the UID

// }

}

}

})

Example Objective C:

[GeideaPaymentAPI payWithGeideaFormWithTheAmount:amount showAddress:Bool showEmail:Bool tokenizationDetails:tokenizationDetails customerDetails:NULL applePayDetails:applePayDetails config:self.config eInvoiceId: eInvoiceId navController: \*\*self\*\* completion:^(GDOrderResponse\* order, GDErrorResponse\* error) {

if (error != NULL) {

if (!error.errors || !error.errors.count) {

//TODO: display relevant fields from GDErrorResponse as title, responseMessage, responseCode, detailedResponseMessage, detailedResponseCode: String

} else {

//TODO: display relevant fields from GDErrorResponse as title, errors

}

} else {

if (order != NULL) {

//TODO: display relevant fields from GDOrderResponse

//TODO: display relevant fields from GDOrderResponse

//TODO:

if cardOnFile is true save tokenId from GDOrderResponse in persistence and agreementId and agreementType for subscriptions (InitistedBy = "Merchant")

// TODO: check [order tokenId] != NULL

//TODO: if paymentOperation is PaymentOperation.preAuthorize:

// TODO: Save order id in persistence for capturing the payment with GeideaPaymentApi.capture

if ([[order detailedStatus] isEqual: @"Authorized"]) {

// TODO: save order.orderId

// TODO: show capture button on the UID

}}}}

### Get Filtered Orders, Get Order By id. Optional for Mobile SDK:

You can get all your existing orders by filter parameters with paginated results.

- The function GeideaPaymentApi.getOrders should have the following parameters:

* **GDOrdersFilter**: SDK filter object: Required
  + - **Take**: Int (Required) represents number of items, used for pagination with default value is 20
    - **Skip**: Int (Required) represents the skip number items, used for pagination with default value is 0. You need to provide the skip number as items.count
    - **DetailedStatuses**: [String] (Optional) represents an array of DetailedStatuses for your filter, range of available values: ["Initiated","Authenticated", "AuthenticationFailed", "AuthorizationFailed", "CaptureFailed","PayFailed","Authorized","Captured","Paid","Refunded","Cancelled","ServerTimedOut"),"ClientTimedOut"),"Blocked"]
    - **Status**: String (Optional) represents a String for your filter, must be one of available values: "Success", "InProgress", "Failed"
    - **FromDate**: Date (Optional) represents a String for your filter, must be one of available values: "Success", "InProgress", "Failed"
    - **ToDate**: Date (Optional) represents a String for your filter, must be one of available values: "Success", "InProgress", "Failed"
* **navController**: UIViewController used for presenting SDK Payment Flow (Required)
  + - can be your UIViewController self SDK flow will be presented modally
    - can be your UINavigationController navigationController SDK flow will be pushed from your navitation controller

**completion**: (GDOrderResponse?, GDErrorResponse?) -? Void used for returning success **GDOrderResponse**, or failure **GDErrorResponse** to your application

Example Swift:

GeideaPaymentAPI.getOrders(with: filter, completion: {[**self**] (response, error) **in**

**if** **let** err = error {

//todo: displayError(err: err)

} **else** **if** **let** safeResponse = response, **let** orders = safeResponse.orders {

// TODO: Display Orders in a TableView. If paginated append new orders at existing orderList }

})

GeideaPaymentAPI.getOrder(with: order, completion:{ [**self**] (response, error) **in**

**if** **let** err = error {

// TODO: displayError(err: err)

} **else** **if** **let** safeResponse = response {

//TODO: Display relevant Fields from GDOrderResponse

} })

### Capture Order, Refund Order and Cancel Order Flow:

### Capture Order:

If the **paymentOperation** of the GeideaPaymentApi.pay or GeideaPaymentApi.payToken was **preAuthorize** and the **detailedStatus** response of the **GDOrderResponse** is “Authorized” the amount must be captured for the payment to be completed. You need to call the SDK GeideaPaymentAPI.capture function:

Example Swift:

GeideaPaymentAPI.capture(with: orderId, navController: self as UIViewController or navigationController as UINavigationController, completion: response, error in {

DispatchQueue.main.async {

If let err = error {

If err.errors.isEmpty {

//TODO: display relevant fields from GDErrorResponse as title, responseMessage, responseCode, detailedResponseMessage, detailedResponseCode: String

} else {

//TODO: display relevant fields from GDErrorResponse as title, errors

}

} else {

guard let orderResponse = response else {

return

}

//TODO: Display relevant fields from GDOrderResponse

//TODO: remove orderId from persistence and capture button from the UI

}

}

})

Example Objective C:

[GeideaPaymentAPI captureWith: orderId navController: self as UIViewController or navigationController as UINavigationController completion: ^(GDOrderResponse\* order, GDErrorResponse\* error) {

if (error != NULL) {

if (!error.errors || !error.errors.count) {

//TODO: display relevant fields from GDErrorResponse

} else {

//TODO: display relevant fields from GDErrorResponse as title, errors

}

} else {

if (order != NULL) {

//TODO: display relevant fields from GDOrderResponse

//TODO: remove orderId from persistence and capture Button from the UI

}

} } }];

### Refund Order (Optional for Mobile SDK):

If the **detailedStatus** response of the **GDOrderResponse** is “Paid” or “Captured” the payment can be Refunded. You don’t need to expose this option for the user

Example Swift:

GeideaPaymentAPI.refund(with: orderId, navController: self as UIViewController or navigationController as UINavigationController, completion: response, error in {

DispatchQueue.main.async {

DispatchQueue.main.async {

If let err = error {

If err.errors.isEmpty {

//TODO: display relevant fields from GDErrorResponse as title, responseMessage, responseCode, detailedResponseMessage, detailedResponseCode: String

} else {

//TODO: display relevant fields from GDErrorResponse as title, errors

}

} else {

guard let orderResponse = response else {

return

}

//TODO: Display relevant fields from GDOrderResponse

} } })

Example Objective C:

[GeideaPaymentAPI refundWith: orderId navController: self as UIViewController or navigationController as UINavigationController completion: ^(GDOrderResponse\* order, GDErrorResponse\* error) {

if (error != NULL) {

if (!error.errors || !error.errors.count) {

//TODO: display relevant fields from GDErrorResponse as title, errors

}

} else {

if (order != NULL) {

//TODO: display relevant fields from GDOrderResponse

}

}

}

### Cancel Order (Optional for Mobile SDK):

If the **status** response of the **GDOrderResponse** is “InProgress” the payment can be Cancelled. This is done automatically by the SDK and also by the payment gateway after the config hppDefaultTimeout period for every uncompleted order, so you don’t need to use this method.

Example Swift:

GeideaPaymentAPI.cancel(with: orderId, navController: self as UIViewController or navigationController as UINavigationController, completion: response, error in {

DispatchQueue.main.async {

DispatchQueue.main.async {

If let err = error {

If err.errors.isEmpty {

//TODO: display relevant fields from GDErrorResponse as title, responseMessage, responseCode, detailedResponseMessage, detailedResponseCode: String

} else {

//TODO: display relevant fields from GDErrorResponse as title, errors

}

}

}

})

Example Objective C:

[GeideaPaymentAPI cancelWith: orderId navController: self as UIViewController or navigationController as UINavigationController completion: ^(GDCancelResponse\* cancel, GDErrorResponse\* error) {

if (error != NULL) {

if (!error.errors || !error.errors.count) {

//TODO: display relevant fields from GDErrorResponse

} else {

//TODO: display relevant fields from GDErrorResponse as title, errors }}}

### Pay Token Flow:

If the **cardOnFIle** field of the GeideaPaymentApi.pay is set to true, the **tokenId** field of the **GDOrderResponse** must be persisted.

- there are two different flows for paying with card token:

1. InitiatedBy: **Internet** – Single payment, agreementId and agreementType can be empty
2. InitiatedBy: **Merchant** – Subscription payment type, **agrementId** and **agrementType** **are mandatory** and must be the same with values passed on GeideaPaymentApi.pay from where the **tokenId** was computed (retrieved from persistence)

Example Swift:

GeideaPaymentAPI.payWithToken(theAmount: amount, withTokenId: tokenId, tokenizationDetails: tokenizationDetails, andEInvoiceId: eInvoiceId andCustomerDetails: customerDetails, navController: **self**, completion:{ response, error **in** {

DispatchQueue.main.async {

If let err = error {

If err.errors.isEmpty {

//TODO: display relevant fields from GDErrorResponse as title, responseMessage, responseCode, detailedResponseMessage, detailedResponseCode: String

} else {

//TODO: display relevant fields from GDErrorResponse as title, errors

}

} else {

guard let orderResponse = response else {

return

}

//TODO: Display relevant fields from GDOrderResponse

//TODO: if paymentOperation is PaymentOperation.preAuthorize:

// Save order id in persistence for capturing the payment with

// GeideaPaymentApi.capture

// if orderResponse.detailedStatus == "Authorized" {

// TODO: save order.orderId

// TODO: show capture button on the UID

// }

}

} })

Example Objective C:

[GeideaPaymentAPI payWithTokenWithTheAmount:amount withTokenId:@"token" tokenizationDetails:tokenizationDetails andEInvoiceId: eInvoiceId andCustomerDetails:customerDetails navController: navVC completion:^(GDOrderResponse\* order, GDErrorResponse\* error) {

if (error != NULL) {

if (!error.errors || !error.errors.count) {

//TODO: display relevant fields from GDErrorResponse as title, responseMessage, responseCode, detailedResponseMessage, detailedResponseCode: String

} else {

//TODO: display relevant fields from GDErrorResponse as title, errors

}

} else {

if (order != NULL) {

//TODO: display relevant fields from GDOrderResponse

//TODO: display relevant fields from GDOrderResponse //TODO: if paymentOperation is PaymentOperation.preAuthorize:

// TODO: Save order id in persistence for capturing the payment with GeideaPaymentApi.capture

if ([[order detailedStatus] isEqual: @"Authorized"]) {

TODO: save order.orderId

// TODO: show capture button on the UID

}

}

}

}];

### Apple Pay Payment Flow:

From persisted **GDConfigResponse** check the flag **isApplePayMobileSupported and isApplePayMobileCertificateAvaialable** before starting the flow

To pay the amount with **Apple Pay** you need to have the apple pay marchantId configured on your apple account and to enable apple pay capabilities. Please follow this for setup:

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

You must request Geidea for an Apple Pay Certificate Request to have the certificate available in the Payment Gateway. With certificate submitted, you can check the Config for Apple pay isApplePayMobileCertificateAvailable and then call the setupApplePay function.

After this you just must call the SDK function GeideaPaymentAPI.setupApplePay in your **UIViewController** with an **UIView** placeholder to the Apple Pay Button and the SDK will make the payment.

The function GeideaPaymentApi.setupApplePay should have the following mandatory parameters:

* **GDApplePayDetails**: necessary details for Apple Pay UI and request (Required)
  + **ViewController**: your host UIVIewController in which the ApplePay UI will be displayed (Required)
  + **buttonView**: UIView as a placeholder where applePay should be displayed (Required)
  + **GDAmount**: SDK Amount object: Required
    - **amount**: Double (Required) -maximum 2 decimals
    - **currency**: String (Required) - 3 letters and must be one of the approved currencies
  + **ApplePayMerchantId**: Your apple account merchantId from the apple pay setup (Required)
* **GDConfigResponse**: response from config API including fags for applePay supported and Merchant name sent to Apple Pay
* **completion**: (GDApplePayResponse?, GDErrorResponse?) -? Void used for returning success **GDApplePayResponse**, or failure **GDOrderResponse** to your application

Example Swift:

GeideaPaymentAPI.setupApplePay(forApplePayDetails: applePayDetails, with: amount, config: merchantConfig, completion: { response, error **in**

**if** **let** err = error { DispatchQueue.main.async {

if let err = error {

if err.errors.isEmpty {

var message = ""

if err.responseCode.isEmpty {

message = "\n responseMessage: \(err.responseMessage)"

} else {

message = "\n responseCode: \(err.responseCode) \n responseMessage: \(err.responseMessage)"

}

//TODO: display relevant fields from GDErrorResponse

} else {

//TODO: display relevant fields from GDErrorResponse

}

} else {

guard let orderResponse = response else {

return

}

//TODO: display relevant fields from GDApplePayResponse

}

} })

Example Objective C:

[GeideaPaymentAPI setupApplePayForApplePayDetails: applePayDetails with:amount config: **merchantConfig** completion:^(GDApplePayResponse\* response, GDErrorResponse\* error) {

if (error != NULL) {

if (!error.errors || !error.errors.count) {

NSString \*message;

if ( [error.responseCode length] == 0) {

message = [NSString stringWithFormat:@"\n responseMessage: %@", error.responseMessage];

} else {

message = [NSString stringWithFormat:@"\n responseCode: %@ \n responseMessage: %@ ", error.responseCode , error.responseMessage];

}

//TODO: display relevant fields from GDErrorResponse

} else {

//TODO: display relevant fields from GDErrorResponse

}

} else {

if (response != NULL) {

//TODO: display relevant fields from GDApplePayResponse

}

}}];

### Get Card scheme logo detector flow:

- You can use the SDK function GeideaPaymentAPI. getCardSchemeLogo to get an UIImage for the desired CardType

- You can also use the cardType detector embedded in SDK.

Example Swift:

cardSchemeLogoIV.image = GeideaPaymentAPI.getCardSchemeLogo(withCardNumber: cardNumber or if you don't want detector

cardSchemeLogoIV.image = GeideaPaymentAPI.getCardSchemeLogo(withCardNumber: cardNumber)

Example Objective C:

\_cardSchemeLogoIV.image=[GeideaPaymentAPI getCardSchemeLogoWithCardNumber:cardNumber];

or if you don't want detector

\_cardSchemeLogoIV.image = [GeideaPaymentAPI getCardSchemeLogoWithCardType:CardTypeVisa]

### 3.1.10 EInvoice Create, Update, Get and Delete operations:

You can use one of the CRUD operations of the SDK if you want to create create EInvoice in app. The CRUD operations API is public exposed, and you can use it if needed in your App.

Example Swift:

GeideaPaymentAPI.createInvoice(with: invoiceDetails, completion: {response, error in

// save EInvoiceId for future payment

})

GeideaPaymentAPI.updateInvoice(with: invoiceDetails, completion: {response, error in

//Use response

})

GeideaPaymentAPI.getInvoice(with: eInvoice, completion: {response, error in

//Use response

})

GeideaPaymentAPI.deleteInvoice(with: eInvoice, completion: {response, error in

//Use response

})

Example Objective C:

[GeideaPaymentAPI createInvoiceWith:invoiceDetails completion:^(GDEInvoiceResponse\* order, GDErrorResponse\* error) {

// save EInvoiceId for future payment

}];

[GeideaPaymentAPI updateInvoiceWith:invoiceDetails completion:^(GDEInvoiceResponse\* order, GDErrorResponse\* error) {

//Use response

}];

[GeideaPaymentAPI getInvoiceWith:eInvoiceId completion:^(GDEInvoiceResponse\* response, GDErrorResponse\* error) {

//Use response

}];

[GeideaPaymentAPI deleteInvoiceWith:eInvoiceId completion:^(GDEInvoiceResponse\* response, GDErrorResponse\* error) {

//Use response

}];

### 3.1.12 SDK Responses from SDK Payment flow:

In the completion object you will receive two objects one for Order response and another one for errors. Both objects must be checked for null.

* **GDErrorResponse** (Nullable) SDK failure response
  + - **errors**: [**String**: [**String**]] - is not empty when bad request is returned
    - **status**: Int is not empty when bad request is returned
    - **title**: **String** can be empty
    - **traceId: String** is not empty when bad request is returned
    - **type: String** is not empty when bad request is returned
    - **responseCode: String** is empty when bad request is returned
    - **responseMessage: String** is empty when bad request is returned
    - **detailedResponseCode****: String** can be empty
    - **detailedResponseMessage: String** can be empty
    - **orderId: String** can be empty
* **GDOrderResponse** fields examples (Nullable) please use to display any relevant information

"order": {

"createdDate": "2020-11-13T14:40:33.050Z",

"createdBy": "string",

"updatedDate": "2020-11-13T14:40:33.050Z",

"updatedBy": "string",

"orderId": "3fa85f64-5717-4562-b3fc-2c963f66afa6",

"amount": 0,

"currency": "string",

"detailedStatus": "Initiated",

"status": "InProgress",

"threeDSecureId": "3fa85f64-5717-4562-b3fc-2c963f66afa6",

"merchantId": "3fa85f64-5717-4562-b3fc-2c963f66afa6",

"merchantPublicKey": "3fa85f64-5717-4562-b3fc-2c963f66afa6",

"parentOrderId": "3fa85f64-5717-4562-b3fc-2c963f66afa6",

"merchantReferenceId": "string",

"callbackUrl": "string",

"customerEmail": "string",

"billingAddress": {

"countryCode": "string",

"street": "string",

"city": "string",

"postCode": "string"

},

"shippingAddress": {

"countryCode": "string",

"street": "string",

"city": "string",

"postCode": "string"

},

"returnUrl": "string",

"cardOnFile": true,

"tokenId": "3fa85f64-5717-4562-b3fc-2c963f66afa6",

"paymentMethod": {

"type": "Card",

"brand": "string",

"cardholderName": "string",

"maskedCardNumber": "string",

"expiryDate": {

"month": 0,

"year": 0

}

},

"totalAuthorizedAmount": 0,

"totalCapturedAmount": 0,

"totalRefundedAmount": 0,

"transactions": [

{

"createdDate": "2020-11-13T14:40:33.050Z",

"createdBy": "string",

"updatedDate": "2020-11-13T14:40:33.050Z",

"updatedBy": "string",

"transactionId": "3fa85f64-5717-4562-b3fc-2c963f66afa6",

"type": "Authentication",

"status": "InProgress",

"amount": 0,

"currency": "string",

"source": "Mobile",

"authorizationCode": "string",

"rrn": "string",

"paymentMethod": {

"type": "Card",

"brand": "string",

"cardholderName": "string",

"maskedCardNumber": "string",

"expiryDate": {

"month": 0,

"year": 0

}

},

"codes": {

"acquirerCode": "string",

"acquirerMessage": "string",

"responseCode": "string",

"responseMessage": "string",

"detailedResponseCode": "string",

"detailedResponseMessage": "string"

},

"authenticationDetails": {

"acsEci": "string",

"authenticationToken": "string",

"paResStatus": "string",

"veResEnrolled": "string",

"xid": "string",

"accountAuthenticationValue": "string",

"proofXml": "string"

}

}

]

}

### SDK Debug Logging System:

- If you’re application has Debug Configuration, SDK Logs all the requests, Responses, and steps in the application.

- On the Release Configuration SDK does not logs anything.

- You can get the device Logs file from XCode

-For retrieving just SDK logs, you can Filter with Keyword **GeideaPaymentSDK ::** or a particular Topic as **Pay With Geidea Form.**

Log Example:

**GeideaPaymentSDK :: \*\*\*\*\*\*\*\* Pay With Geidea Form \*\*\*\*\*\*\*\***

**GeideaPaymentSDK :: \*\*\*\*\*\*\*\* SETUP ApplePay \*\*\*\*\*\*\*\***

**GeideaPaymentSDK :: \*\*\*\*\*\*\*\* GET Card Logo \*\*\*\*\*\*\*\***

## Field specification (data dictionary)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Parameter name | Description | Type | Validations | Optionality |
| merchantKey | Merchant ID assigned to you from Geidea | String | Must be a GUID and at least 3 characters long | Mandatory |
| amount | Transaction amount | Number | Must be decimal, greater than 0 | Mandatory |
| currency | Currency of the amount, standard:  ISO 4712 currency code | String | Must be 3 characters long | Mandatory |
| callbackURL | The response with order details, will be returned to this URL | String | Must be a valid URL and to have an HTTPS protocol | Optional |
| merchantReferenceID | Use this as your unique reference for each transaction | String | \* | Optional |
| email | A field to input customer email if you already have it | String | Must be a valid email address | Optional |
| address | Object with 3 parameters: showAddress, billing and shipping | Object | - | Optional |
| billing  For billing address – the following parameters can be passed:   * country * streetNameAndNumber * city * postcode | A field to input billing address details for thecustomer if you already have it  country standard:  ISO 3166 – alpha-3 code | String | **country -** must be 3 characters | Optional |
| shipping  For shipping address – the same parameters as for billing can be passed:   * country * streetNameAndNumber * city * postcode | A field to input shipping address details for thecustomer if you already have it  country standard:  ISO 3166 – alpha-3 code | String | **country -** must be 3 characters | Optional |

\*Please note that all above parameters have a max length of 255 symbols.

## Response codes

### iOS SDK Response codes and messages:

These response codes and messages appear when there is an error (if any of the fields are not input as expected in the object specification) with some of the parameters on the “GeideaPaymentAPI.pay”function:

|  |  |  |
| --- | --- | --- |
| **GeideaPaymentSDK errors** |  |  |
| **Response code** | **Response message** | **Detailed Response message** |
| 001 | Missing public key |  |
| 002 | Missing password |  |
| 003 | Invalid amount: | Invalid amount: Amount must have maximum 2 decimals |
| 004 | Missing amount |  |
| 005 | Invalid currency | Invalid currency: Currency must have exactly 3 letters |
| 006 | Missing currency |  |
| 007 | Invalid CVV | Invalid CVV: CV must have 3 or 4 digits |
| 008 | Missing CVV |  |
| 009 | Invalid callback URL | Invalid callback URL: Callback must have a valid URL format |
| 011 | Invalid billing country code | Invalid billing country code: Country Code must have exactly 3 letters and supported |
| 012 | Invalid shipping country code | Invalid shipping country code: Country Code must have exactly 3 letters and supported |
| 013 | Missing Card holder name |  |
| 014 | Missing card number |  |
| 015 | Invalid expiryMonth | Invalid expiry month: Must be a digit from 1 to 12 |
| 016 | Invalid email address | Invalid email address: Email address must be valid |
| 017 | Invalid expiryYear | Invalid expiry year: Must be a digit from 1 to 99 |
| 018 | Invalid billing address | Invalid billing address: All fields must have maximum 255 characters |
| 019 | Invalid shipping address | Invalid shipping address: All fields must have maximum 255 characters |

### Geidea Payment Response codes and messages:

|  |  |  |
| --- | --- | --- |
| **Response code groups** |  |  |
| **Response Code** | **Response Message** | **Description** |
| 000 | Success | Success |
| 100 | General error - Payment was not successful | General error - when the error is outside of the error list |
| 200 | Payment validation failed | Initial backend validation error (card number, luhn check, etc) |
| 300 | 3DS authentication failed | 3DS errors |
| 400 | Payment authorization failed | Authorize operation errors |
| 500 | Payment capture failed | Capture operation errors |
| 600 | Payment blocked by gateway | Filter service errors |
| 700 | Payment failed | Pay operation errors |
| 800 | Refund failed | Refund operation errors |

**Detailed response code and message groups:**

| **Success group (000)** |
| --- |
| **Detailed Response Code** | **Detailed Response Message** |
| 000.000 | Your transaction was successful |
| 000.001 | Your transaction was successful - proceed without 3DS verification code |
| 000.002 | Your transaction was successful - proceed with 3DS verification code |

| **General error group (100)** |  |
| --- | --- |
| **Detailed Response Code** | **Detailed Response Message** |
| 100.001 | HSM invalid input fields |
| 100.002 | Encrypt failed, invalid data |
| 100.003 | Decrypt failed, invalid key |
| 100.004 | Decrypt failed, invalid data |
| 100.005 | Decrypt failed, wrong data-key combination |
| 100.006 | Unable to create key |
| 100.007 | Unable to find key |
| 100.008 | Unable to create order |
| 100.009 | Unable to create transaction |
| 100.010 | Unable to update transaction |
| 100.011 | Order not found |
| 100.012 | Transaction not found |
| 100.013 | Internal Server Error |
| 100.014 | Invalid provider credentials - MPGS |
| 100.015 | MPGS URL not found |
| 100.016 | Invalid provider credentials - GSDK |
| 100.017 | HTTP request failed with connection error |
| 100.018 | Your payment was not successful |
| 100.019 | Unable to update order |
| 100.020 | Cancelled by user |
| 100.021 | Client timed out |
| 100.022 | System timed out |

| **Payment validation error group (200)** |
| --- |
| **Detailed Response Code** | **Detailed Response Message** |
| 200.001 | Card is expired |
| 200.002 | Unknown or unsupported card brand |
| 200.003 | Invalid card number length |
| 200.004 | Currency {currency}  is not supported |
| 200.005 | Card brand {cardBrand}  is not supported |
| 200.006 | Invalid card number, Luhn check failed |
| 200.007 | Invalid cvv |

| **3DS group errors (300)** |
| --- |
| **Detailed Response Code** | **Detailed Response Message** | **Description** |
| 300.001 | The card is not 3DS enrolled | Not enrolled (MPGS - do not proceed) |
| 300.002 | The card does not support 3DS | 3DS not supported / card not participating |
| 300.003 | Authentication enrollment check not available | Connection error on enrollment check |
| 300.004 | Customer authentication failed | Failed authentication (MPGS - do not proceed), N response |
| 300.005 | Customer authentication attempted, but could not be completed | M response from authentication (do not proceed) |
| 300.006 | Customer authentication not available | X or U response from MPGS |
| 300.007 | Error parsing authentication response | P response from MPGS |
| 300.008 | Invalid signature on authentication response | S response from MPGS |
| 300.009 | MPI processing error | I response from MPGS |
| 300.010 | Authentication could not be created | Internal service error |

| **Authorize operation error group (400)** |
| --- |
| **Detailed Response Code** | **Detailed Response Message** |
| 400.001 | Authorization failed |

| **Capture operation error group (500)** |
| --- |
| **Detailed Response Code** | **Detailed Response Message** |
| 500.001 | Capture failed |

| **Filter service error group (600)** |
| --- |
| 600.001 | Blocked card BIN |
| 600.002 | Blocked IP country |
| 600.003 | Blocked customer country |

| **Pay operation error group (700)** |
| --- |
| **Detailed Response Code** | **Detailed Response Message** |
| 700.001 | Payment failed |

| **Refund operation error group (800)** |
| --- |
| **Detailed Response Code** | **Detailed Response Message** | **Description** |
| 800.001 | Refund failed | //when MPGS fails |
| 800.002 | Refund failed - Order does not exist | // when order cannot be found |
| 800.003 | Refund failed - Payment for order has not been completed | // when order has not been paid / captured |
| 800.004 | Refund failed - Invalid order | //when a merchant tries to refund another merchants Paid/Captured order |

1. Glossary

|  |  |
| --- | --- |
| Glossary Item | Description |
| 3DS | 3D Secure - secure protocol designed to ensure enhanced security and strong authentication |
| MPGS | Mastercard Payment Gateway Services |
| BIN | Bank Identification Number - the initial four to six numbers that appear on a credit/debit card |
| HPP | Geidea Hosted Payment Page |