



Geidea React-Native Plugin

1.0.1

—

Mimocodes

248 El Narges Villas, 5th Settlement, Cairo, Egypt

Overview

The purpose of this Integration guide is to serve as technical documentation for merchants who wish to integrate the Geidea Payment plugin for React-native so that they can use Payment Gateway services in their application.

This guide describes the functionality and APIs provided by the plugin and different approaches to integrating the plugin and customizing it.

Integration

Please install the prerequisites first, then you can either integrate the plugin using

- I. **Simple** integration – The plugin hosts the entire UI flow and performs all transactions. A “turnkey” solution that requires minimal setup. You simply call a method to start the Payment flow and then receive your Order after everything is ready. This can be done as
 - A. **Payment modal**
 - B. **Checkout screen**
- II. **Custom** integration – the Merchant app hosts the entire UI flow (payment form, authentication) and performs all transactions by calling the Direct APIs through the plugin.

Prerequisites

1 - Install the plugin in your React-native app with the following command

```
npm install --save react-native-webview react-native-vector-icons
react-native link react-native-webview
react-native link react-native-vector-icons

npm install
https://github.com/GeideaSolutions/ReactNative-SDK/react_geideapay?main
```

I. Simple integration

A. Payment modal

1 - Import the required libraries

```
import PaymentModal from 'react_geideapay/components/PaymentModal';
```

2 - Define a state variable to control the visibility of the payment modal

```
state = {  
  checkoutVisible: false,  
};
```

3 - Define the following functions

```
closePaymentModal() {  
  this.setState({checkoutVisible: false});  
}  
onPaymentSuccess(response) {  
  //handle payment success response here  
}  
onPaymentFailure(response) {  
  //handle payment error response here  
}
```

4 - Implement the **PaymentModal** in your render function as follows

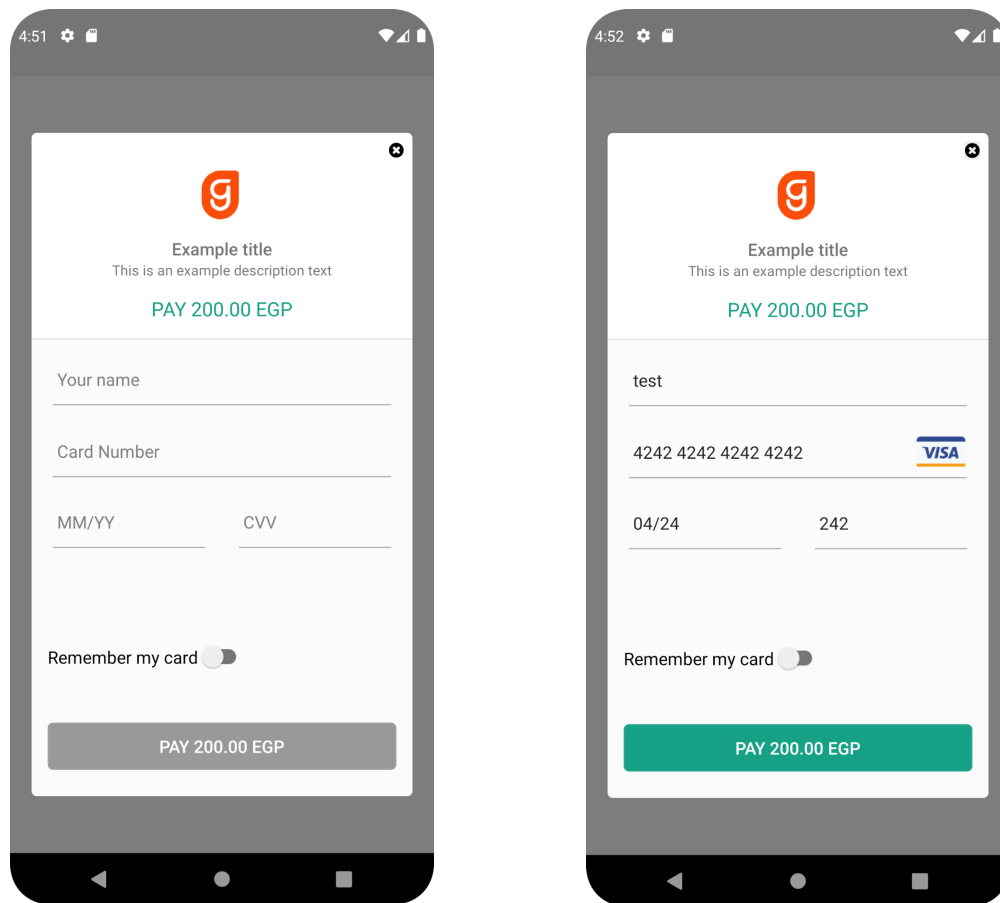
```
render() {  
  return  
    <View style={styles.container}>  
      {this.renderPaymentModal()}  
    </View>;  
}  
  
renderPaymentModal() {  
  const {checkoutVisible} = this.state;  
  return (  
    <PaymentModal  
      amount={200}  
      visible={checkoutVisible}  
      title="Example title"  
      description="This is an example description text"
```

```
        currency="EGP"
        callbackUrl="https://callbackUrl.com"
        onRequestClose={this.closePaymentModal}
        publicKey={publicKey}
        apiPassword={apiPassword}
        onPaymentSuccess={this.onPaymentSuccess}
        onPaymentFailure={this.onPaymentFailure}
    />
  );
}
```

Then you can display the dialog by setting the `checkoutVisible` state to true as follows

```
this.setState({checkoutVisible: true});
```

After this, you should get a modal dialog displayed as shown below



B. Checkout screen

1 - Make sure that your app is ready for stack navigation as shown here <https://reactnavigation.org/docs/native-stack-navigator>. An example of an app with a navigation stack is shown below

```
import * as React from 'react';

import {NavigationContainer} from '@react-navigation/native';

import {createNativeStackNavigator} from
 '@react-navigation/native-stack';

import HomeScreen from './HomeScreen';

import CheckoutScreen from
 'react_geideapay/components/CheckoutScreen';
```

```
const Stack = createNativeStackNavigator();

const App = () => {
  return (
    <NavigationContainer>
      <Stack.Navigator>
        <Stack.Screen
          name="Home"
          component={HomeScreen}
          options={{title: 'Geidea App'}}
        />
        <Stack.Screen
          name="CheckoutScreen"
          component={CheckoutScreen}
          options={({route}) => ({title:
route.params.screenTitle})}
        />
      </Stack.Navigator>
    </NavigationContainer>
  );
};

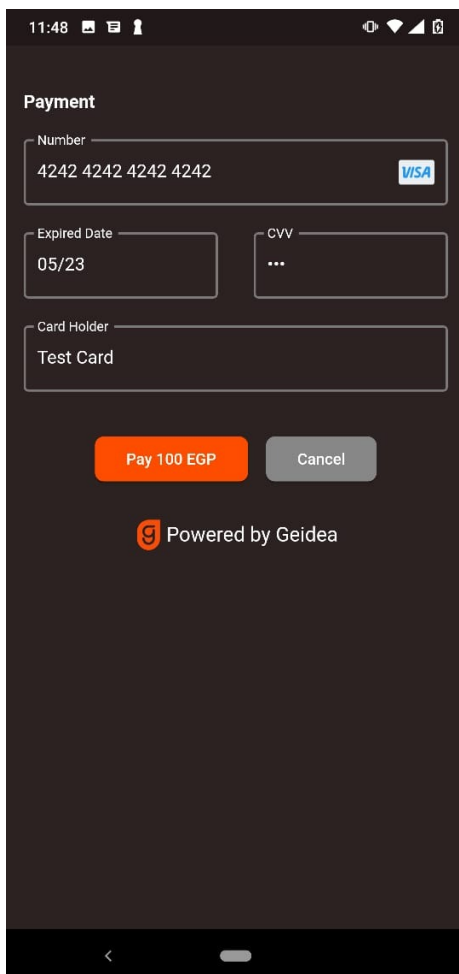
export default App;
```

2 - Implement a function similar to the following to navigate to the checkout screen.

```
showScreenCheckout() {
  this.props.navigation.push('CheckoutScreen', {
    amount: 200,
    screenTitle: 'Checkout',
    title="Example title"
    description="This is an example description text"
    currency: 'EGP',
```


```
        callbackUrl="https://callbackUrl.com",
        publicKey: publicKey,
        apiPassword: apiPassword,
        successResponse: '',
        failureResponse: '',
        backgroundColor: '#fff',
        cardColor: '#583e9e',
        previousScreen: 'Home', // name as in the navigation stack
    });
}
```

After you call the above function, this, the app will navigate to the checkout screen as shown below



11:48

Payment


Number
4242 4242 4242 4242 

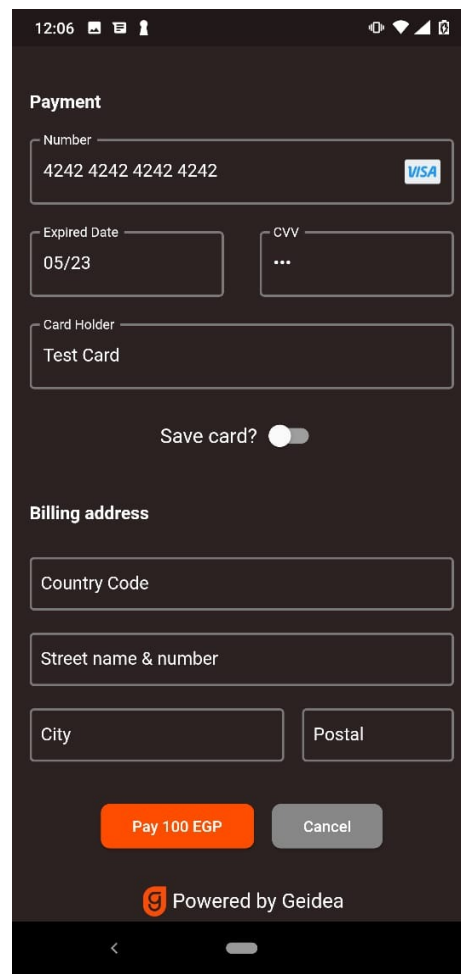
Expired Date
05/23

CVV
...

Card Holder
Test Card


Pay 100 EGP Cancel

 Powered by Geidea



12:06

Payment

Number
4242 4242 4242 4242 

Expired Date
05/23

CVV
...

Card Holder
Test Card

Save card? ☐


Billing address

Country Code

Street name & number

City Postal

Pay 100 EGP Cancel

 Powered by Geidea

Parameters

The following table shows the meaning of the parameters used in both the payment modal and the checkout screen

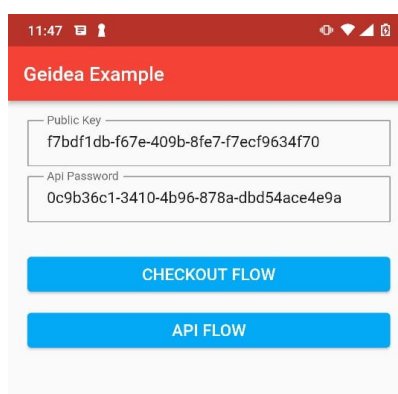
Parameter name	Description	Type	Optionality
amount	Transaction amount Must be convertible to decimal, greater than 0 and must not have more than 2 digits after decimal point	String	Mandatory
currency	Currency of the amount, standard: ISO 4217 currency code Must be 3 characters long	String	Mandatory
cardNumber	cardNumber: must be a numeric value for MasterCard, VISA, MADA this is 16 digits, for AMEX – 15 digits	String	Mandatory
merchantReferenceID	Merchant unique reference for each order/transaction	String	Optional
paymentOperation	Provide a payment operation to choose what transactions to execute. Must be one from expected values: "Pay", "AuthorizeCapture", "PreAuthorize". If value not provided – this parameter takes the pre-configured value for this merchant(can be null)	String	Optional
callbackURL	The response with order details, will also be returned to this URL Must be a valid URL and to have an HTTPS protocol	String	Optional
billingAddress For billing address – the following parameters can be passed: <ul style="list-style-type: none"> country 	A field to input billing address details for the customer if you already have it country standard: ISO 3166 – alpha-3 code	String	Optional

<ul style="list-style-type: none"> streetNameAndNumber city postcode 	<ul style="list-style-type: none"> country - must be 3 characters 		
showBilling	Request billing address details at checkout form (Default False)	Boolean	Optional
shippingAddress For shipping address – the same parameters as for billing can be passed: <ul style="list-style-type: none"> country streetNameAndNumber city postcode 	A field to input shipping address details for the customer if you already have it country standard: ISO 3166 – alpha-3 code <ul style="list-style-type: none"> country - must be 3 characters 	String	Optional
showShipping	Request shipping address details at checkout form (Default False)	Boolean	Optional
showSaveCard	Ask the user whether to save card information at checkout (Default False)	Boolean	Optional
customerEmail	Must be a valid email address	String	Optional
paymentIntentId	Must be a valid paymentIntentId generated by E-Invoice API – add this field to pay an E-Invoice	GUID	Optional

II. Example Application

For the sake of proper demonstration of all features of Geidea Flutter SDK, an example application is readily available to help developer maximize their capability of using our Flutter SDK features. In the example application you can:

- Configure your merchant public key and API password



11:47

Geidea Example

Public Key
f7bdf1db-f67e-409b-8fe7-f7ecf9634f70

API Password
0c9b36c1-3410-4b96-878a-dbd54ace4e9a

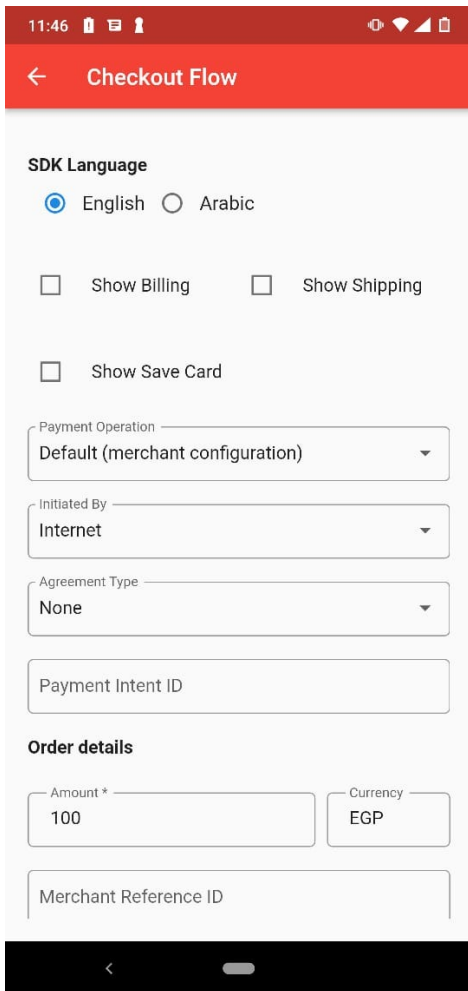
CHECKOUT FLOW

API FLOW

- B. Choose between our readily available checkout solution or choose to design a custom checkout flow.
- C. In Gediea checkout flow you will have a set of controls over the checkout form to be presented to your customers:
 - 1. **Language control:** English - Arabic
 - 2. **Show Billing Address:** enabling this feature will ask your customer to enter his billing address information during checkout. It defaults to false as billing address is an optional parameters. In case you have provided the checkout method with the customer billing information while enabling this feature, the provided billing information will be auto filled in the checkout form.
 - 3. **Show Shipping Address:** enabling this feature will ask your customer to enter his shipping address information during checkout. It defaults to false as shipping address is an optional parameters. In case you have provided the checkout method with the customer shipping information while enabling this feature, the provided shipping information will be auto filled in the checkout form.
 - 4. **Show Save Card:** enabling this feature will ask your customer whether to save card information for future use. If your customer choose to save his card information, the card token will be returned back in the payment processing result object.
 - 5. **Callback URL:** In order to test the functionality of your backend to receive callbacks from Geidea, you may configure the callback endpoint to which you need to listen to the callback requests whenever a customer issue a payment through your application.
- D. If you decided to customize your checkout and choose the API flow, the example application will guide you through the correct sequence of API flow to get a fully functional checkout routine:
 - 1. **Initiate authentication API:** For the detailed request and response parameters, please refer to the next section of this document.
 - 2. **Payer authentication API:** at this step, your customer will be redirected to the 3DS authentication to fulfill the authentication session with his bank. Once the customer completes this step, the API response will be returned for you to complete, or terminate, the payment flow. For the detailed request and response parameters, please refer to the next section of this document.
 - 3. **Pay with Card API:** For the detailed request and response parameters, please refer to the next section of this document.

4. **Refund API:** For the detailed request and response parameters, please refer to the next section of this document.

Geidea Checkout Flow



11:46

← Checkout Flow

SDK Language

☒ English ☐ Arabic

☐ Show Billing ☐ Show Shipping

☐ Show Save Card

Payment Operation
Default (merchant configuration)

Initiated By
Internet

Agreement Type
None

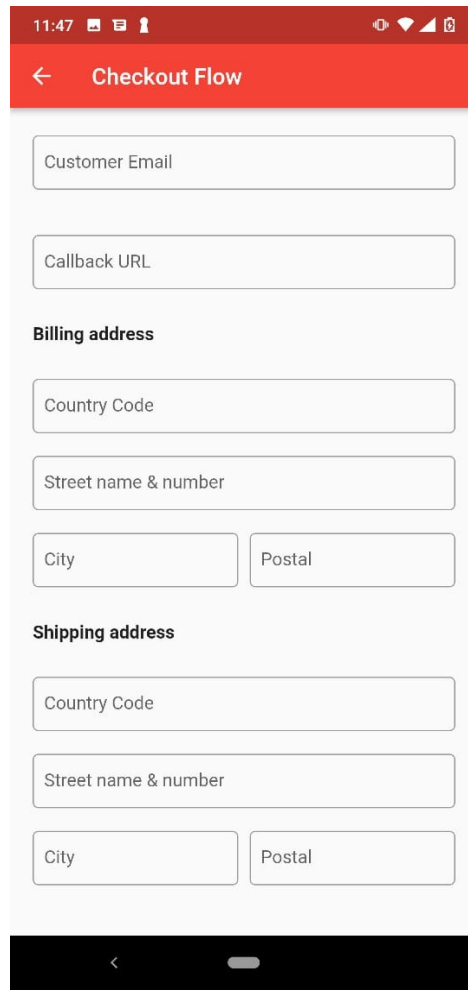
Payment Intent ID

Order details

Amount *
100

Currency
EGP

Merchant Reference ID



11:47

← Checkout Flow

Customer Email

Callback URL

Billing address

Country Code

Street name & number

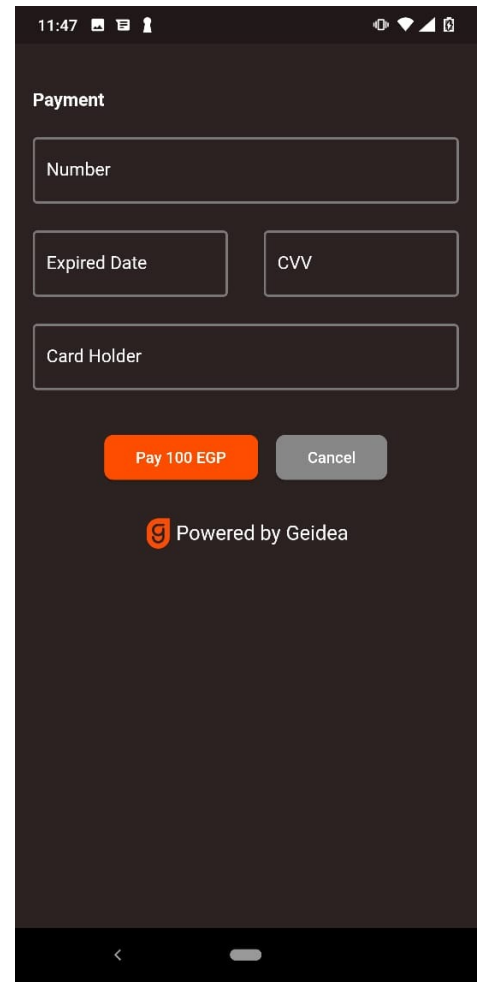
City Postal

Shipping address

Country Code

Street name & number

City Postal



11:47


Payment

Number

Expired Date CVV


Card Holder

Pay 100 EGP Cancel

 Powered by Geidea

12:07

Payment

Number
4242 4242 4242 4242 

Expired Date
05/23

CVV
...

Card Holder
Test Card

Save card? ☒


Shipping address

Country Code

Street name & number


City Postal

Pay 100 EGP **Cancel**

 Powered by Geidea

12:06

Payment

Number
4242 4242 4242 4242 

Expired Date
05/23

CVV
...

Card Holder
Test Card

Save card? ☒


Billing address

Country Code

Street name & number


City Postal

Pay 100 EGP **Cancel**

 Powered by Geidea

12:06

Payment

Number
4242 4242 4242 4242 

Expired Date
05/23

CVV
...

Card Holder
Test Card

Save card? ☐


Billing address

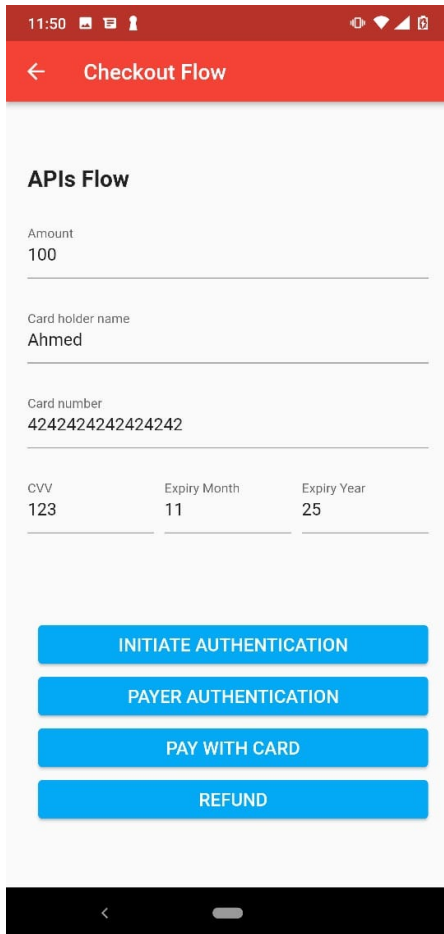
Country Code

Street name & number

City Postal

Pay 100 EGP **Cancel**

 Powered by Geidea



11:50

← Checkout Flow

APIs Flow

Amount
100

Card holder name
Ahmed

Card number
4242424242424242

CVV
123

Expiry Month
11

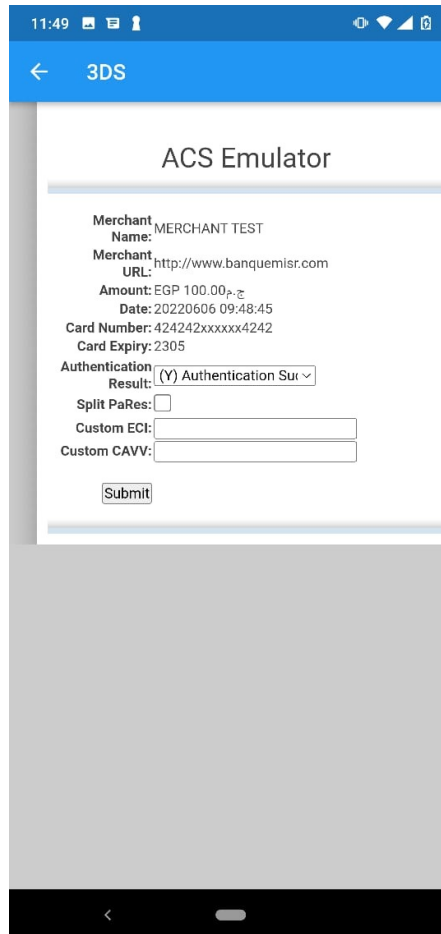
Expiry Year
25

INITIATE AUTHENTICATION

PAYER AUTHENTICATION

PAY WITH CARD

REFUND

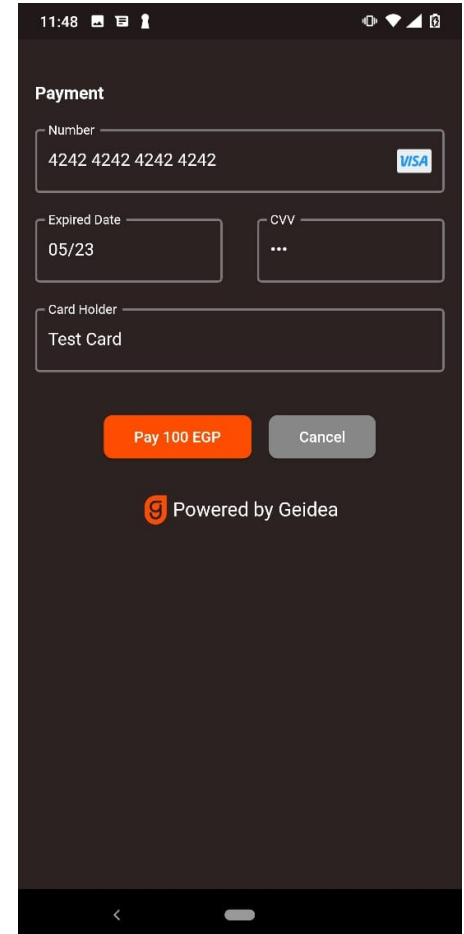


11:49

← 3DS


ACS Emulator

Merchant Name: MERCHANT TEST
Merchant URL: http://www.banquemisr.com
Amount: EGP 100.00
Date: 20220606 09:48:45
Card Number: 424242xxxxxx4242
Card Expiry: 2305
Authentication Result: (Y) Authentication Successful
Split PaRes: ☐
Custom ECI:
Custom CAVV:
Submit



11:48

Payment


Number
4242 4242 4242 4242 

Expired Date
05/23

CVV
...

Card Holder
Test Card

Pay 100 EGP Cancel

 Powered by Geidea

API Checkout Flow

III. Custom integration

The Merchant app hosts the entire UI flow (payment form, authentication) and performs all transactions by calling the Direct APIs through the plugin. The APIs must be called in the following sequence

A. Initiate authentication

Use the following code

```
let initiateAuthenticationRequestBody =  
  new InitiateAuthenticationRequestBody(  
    amount,  
    currency,  
    cardNumber  
  );  
  
GeideaApi.initiateAuthentication(  
  initiateAuthenticationRequestBody,  
  publicKey,  
  apiPassword,  
)  
  .then(res => {  
    console.log(res);  
    orderId = res.orderId;  
  })  
  .catch(err => {  
    console.log(err);  
  });
```

The fields of the `InitiateAuthenticationRequestBody` object are defined in the following table

Parameter name	Description	Type	Optionality
amount	Transaction amount Must be decimal, greater than 0 and must not have more than 2 digits after decimal point	decimal	Mandatory
currency	Currency of the amount, standard: ISO 4217 currency code Must be 3 characters long	String	Mandatory
cardNumber	cardNumber: must be a numeric value for MasterCard, VISA, MADA this is 16 digits, for AMEX – 15 digits	String	Mandatory
merchantReferenceID	Merchant unique reference for each order/transaction	String	Optional

paymentOperation	<p>Provide a payment operation to choose what transactions to execute.</p> <p>Must be one from expected values: "Pay", "AuthorizeCapture", "PreAuthorize".</p> <p>If value not provided – this parameter takes the pre-configured value for this merchant(can be null)</p>	String	Optional
callbackURL	<p>The response with order details, will also be returned to this URL</p> <p>Must be a valid URL and to have an HTTPS protocol</p>	String	Optional
billingAddress For billing address – the following parameters can be passed: <ul style="list-style-type: none"> country streetNameAndNumber city postcode 	<p>A field to input billing address details for the customer if you already have it</p> <p>country standard: ISO 3166 – alpha-3 code</p> <ul style="list-style-type: none"> country - must be 3 characters 	String	Optional
shippingAddress For shipping address – the same parameters as for billing can be passed: <ul style="list-style-type: none"> country streetNameAndNumber city postcode 	<p>A field to input shipping address details for the customer if you already have it</p> <p>country standard: ISO 3166 – alpha-3 code</p> <ul style="list-style-type: none"> country - must be 3 characters 	String	Optional
customerEmail	Must be a valid email address	String	Optional
paymentIntentId	Must be a valid paymentIntentId generated by E-Invoice API – add this field to pay an E-Invoice	GUID	Optional

B. Payer authentication

Use the following code

```
let expiryDate = new expiryDate(month, year);

let card = new PaymentCard(cardholderName, cardNumber, cvv,
expiryDate);

let payerAuthenticationRequestBody =
  new PayerAuthenticationRequestBody(
    amount,
    currency,
    card,
    orderId,
  );

GeideaApi.payerAuthentication(
  payerAuthenticationRequestBody,
  publicKey,
  apiPassword,
  this.props.navigation,
)
.then(res => {
  console.log(res);
  threeDSecureId = res.threeDSecureId;
})
.catch(err => {
  console.log(err);
});
```

The fields of the **PayerAuthenticationRequestBody** object are defined in the following table

Parameter name	Description	Type	Optionality
amount	Transaction amount	decimal	Mandatory

	Must be decimal, greater than 0 and must not have more than 2 digits after decimal point		
currency	Currency of the amount, standard: ISO 4217 currency code Must be 3 characters long	String	Mandatory
card: The following parameters can be passed: <ul style="list-style-type: none"> • cardholderName • cardNumber • cvv • expiryDate (with following) <ul style="list-style-type: none"> ○ month ○ year 	Payment details <ul style="list-style-type: none"> • cardholderName – must be a string value • cardNumber: must be a numeric value for MasterCard, VISA, MADA this is 16 digits, for AMEX – 15 digits • cvv: must be a numeric value for MasterCard, VISA, MADA this is 3 digits, for AMEX – 4 digits • expiryDate <ul style="list-style-type: none"> ○ month: must be a numeric value (01-12) ○ year: must be a numeric value (00-99) 	String	All are Mandatory
orderId	Id of the order. A new order is being created with an Authenticate operation. If this is a new Authenticate operation – expected input is null. orderId will be then created and returned in the case of successful response. If an Authenticate operation has been executed and returned an error an order has been created and the operation can be retried – in this scenario use the created orderId. Must be a GUID	String	Mandatory
cardOnFile	If tokenization will be used this needs to be true in order to save	Boolean	Optional

	cardholder card details for future payments		
merchantReferenceID	Merchant unique reference for each order/transaction	String	Optional
paymentOperation	<p>Provide a payment operation to choose what transactions to execute.</p> <p>Must be one from expected values: "Pay", "AuthorizeCapture", "PreAuthorize".</p> <p>If value not provided – this parameter takes the pre-configured value for this merchant(can be null)</p>	String	Optional
callbackURL	<p>The response with order details, will also be returned to this URL</p> <p>Must be a valid URL and to have an HTTPS protocol</p>	String	Optional
billingAddress For billing address – the following parameters can be passed: <ul style="list-style-type: none"> country streetNameAndNumber city postcode 	<p>A field to input billing address details for the customer if you already have it</p> <p>country standard: ISO 3166 – alpha-3 code</p> <ul style="list-style-type: none"> country - must be 3 characters 	String	Optional
showBilling	Request billing address details at checkout form (Default False)	Boolean	Optional
shippingAddress For shipping address – the same parameters as for billing can be passed: <ul style="list-style-type: none"> country streetNameAndNumber city postcode 	<p>A field to input shipping address details for the customer if you already have it</p> <p>country standard: ISO 3166 – alpha-3 code</p> <ul style="list-style-type: none"> country - must be 3 characters 	String	Optional

showShipping	Request shipping address details at checkout form (Default False)	Boolean	Optional
showSaveCard	Ask the user whether to save card information at checkout (Default False)	Boolean	Optional
customerEmail	Must be a valid email address	String	Optional
returnURL	Client to be redirected to this URL once Authenticate has been successful. Must be a URL with an HTTPS protocol	String	Optional
paymentIntentId	Must be a valid paymentIntentId generated by E-Invoice API – add this field to pay an E-Invoice	GUID	Optional

Note:

As **PayerAuthentication** operation navigate to a 3D secure screen, you must add the following to the navigation stack

```
import ThreeDSScreen from 'react_geideapay/components/
ThreeDSScreen';
<Stack.Screen
  name="Browser"
  component={ThreeDSScreen}
  options={({route}) => ({title: route.params.title})}
/>
```

C. Pay operation

Use the following code

```
let expiryDate = new expiryDate(month, year);

let card = new PaymentCard(cardholderName, cardNumber, cvv,
expiryDate);

let payDirectRequestBody = new PayDirectRequestBody(
threeDSecureId, orderId, amount, currency, card);
GeideaApi.payDirect(payDirectRequestBody, publicKey, apiPassword)
```

```

    .then(res => {
        console.log(res);
    })
    .catch(err => {
        console.log(err);
    });

```

The fields of the **PayDirectRequestBody** object are defined in the following table

Parameter name	Description	Type	Optionality
threeDSecureId	Three D Secure ID of the order. This has been created with the Authenticate operation and client must have passed Authentication successfully for the Pay operation to be successful as well	String	Mandatory
orderId	Id of the order. A new order is being created with an Authenticate operation. The returned value when Authentication has been successful need to be input in this parameter. Cannot be null or empty. Must be a GUID	String	Mandatory
amount	Transaction amount Must be decimal, greater than 0 and must not have more than 2 digits after decimal point	decimal	Mandatory
currency	Currency of the amount, standard: ISO 4712 currency code Must be 3 characters long	String	Mandatory
card: Following parameters can be passed: <ul style="list-style-type: none">cardholderNamecardNumbercvvexpiryDate (with following)	Payment details <ul style="list-style-type: none">cardholderName – must be a string valuecardNumber: must be a numeric value for MasterCard, VISA, MADA this is 16 digits, for AMEX – 15 digits	String	All are Mandatory

<ul style="list-style-type: none"> ○ month ○ year 	<ul style="list-style-type: none"> ● cvv: must be a numeric value for MasterCard, VISA, MADA this is 3 digits, for AMEX – 4 digits ● expiryDate <ul style="list-style-type: none"> ○ month: must be a numeric value (01-12) ○ year: must be a numeric value (00-99) 		
paymentOperation	<p>Provide a payment operation to choose what transactions to execute.</p> <p>Must be one from expected values: "Pay", "AuthorizeCapture", "PreAuthorize".</p> <p>If value not provided – this parameter takes the pre-configured value for this merchant(can be null)</p>	String	Optional
callbackURL	<p>The response with order details, will also be returned to this URL</p> <p>Must be a valid URL and to have an HTTPS protocol</p>	String	Optional
paymentIntentId	<p>Must be a valid paymentIntentId generated by E-Invoice API – add this field to pay an E-Invoice</p>	GUID	Optional

Custom APIs

The following APIs are included in the plugin.

I. Refund

Use the following code

```
let refundRequestBody = new RefundRequestBody(orderId);
GeideaApi.refund(refundRequestBody, publicKey, apiPassword)
    .then(res => {
        console.log(res);
    })
    .catch(err => {
        console.log(err);
    });
```

The fields of the `RefundRequestBody` object are defined in the following table

Parameter name	Description	Type	Optionality
orderId	Id of the order. A new order is being created with an Authenticate operation. The returned value when Authentication has been successful need to be input in this parameter. Cannot be null or empty. Must be a GUID	String	Mandatory
callbackURL_	The response with order details, will also be returned to this URL Must be a valid URL and to have an HTTPS protocol	String	Optional

II. Void

Use the following code

```
let refundRequestBody = new RefundRequestBody(orderId);
GeideaApi.voidOperation(refundRequestBody, publicKey,
    apiPassword)
```

```
.then(res => {  
    console.log(res);  
})  
.catch(err => {  
    console.log(err);  
});
```

The fields of the **RefundRequestBody** object are as defined in the previous table

III. Cancel

Use the following code

```
let cancelRequestBody = new CancelRequestBody(orderId,  
"CancelledByUser");  
  
GeideaApi.cancel(refundRequestBody, publicKey, apiPassword)  
    .then(res => {  
        console.log(res);  
    })  
    .catch(err => {  
        console.log(err);  
    });
```

The fields of the **CancelRequestBody** object in this case are defined in the following table

Parameter name	Description	Type	Optionality
orderId	Id of the order. A new order is being created with an Authenticate operation. The returned value when Authentication has been successful need to be input in this parameter. Cannot be null or empty. Must be a GUID	String	Mandatory
reason	The reason for the Direct API call can only be set to 'CancelledByUser'	String	Mandatory
callbackURL	The response with order details, will also be returned to this URL	String	Optional

	Must be a valid URL and to have an HTTPS protocol		
--	---	--	--

IV. Capture

Use the following code

```
let captureRequestBody = new CaptureRequestBody(orderId);
GeideaApi.capture(refundRequestBody, publicKey, apiPassword)
    .then(res => {
        console.log(res);
    })
    .catch(err => {
        console.log(err);
    });
```

The fields of the **CaptureRequestBody** object in this case are defined in the following table

Parameter name	Description	Type	Optionality
orderId	Id of the order. A new order is being created with an Authenticate operation. The returned value when Authentication has been successful need to be input in this parameter. Cannot be null or empty. Must be a GUID	String	Mandatory
callbackURL	The response with order details, will also be returned to this URL Must be a valid URL and to have an HTTPS protocol	String	Optional
paymentIntentId	Must be a valid paymentIntentId generated by Payment Intent API – add this field to pay an E-Invoice	GUID	Optional