Pinelabs React JS SDK Integration

This document explains the integration process of the Pinelabs React js SDK for your React js applications

By Pinelabs Team

PineLabs React JS SDK

This SDK offers simple to use api for integrating PineLabs api in your react js applications. It provide several easy methods for creating, fetching orders and calculate EMIs and verify hash.

Installation

Prerequisites

Before installing the React js SDK, make sure you meet the following prerequisites:

- NODE version 18.17.1 or higher
- NPM version 9.6.7 or higher

In order to install this SDK locally from a folder you'll need to run the following commands. It will link and install the SDK in your react js project.

```
npm link "../path-to-sdk-folder"
npm install "../path-to-sdk-folder"
```

Usage For SDK

Create Instance of PineLabs SDK

Import usePinelabs hooks from pinelabs sdk . It takes 4 parameters which are as follows:

1. Merchant ID (string) : Merchant ID provided by PineLabs
2. Merchant Access Code (string) : Merchant Access Code Provided by PineLabs
3. Merchant Secret (string) : Merchant Secret
4. isTest (boolean) : If using test mode then set this to `true`

```
    const pinelabs = usePineLabs("merchant_id", "merchant_access_code", "merc
hant_secret", isTest)
```

Create Order

This section explains how to create order for payment processing. There are a couple of things required in order to create an order.

Parameters Required & Optional

```
    // Transaction Data ( Mandatory )
const txn_data = {
    txn_id: "", // String
```

```
    callback: '', // String
    amount_in_paisa: '1000', // String
}

    // Customer Data ( Optional )
const customer_data = {
    email_id: "", // String
    first_name: "", // String
    last_name: "", // String
    mobile_no: "", // String
    customer_id: "", // String
}

    // Billing Data ( Optional )
const billing_data = {
    address1: "", // String
    address2: "", // String
    address3: "", // String
    pincode: "", // String
    city: "", // String
    state: "", // String
    country: "", // String
}

    // Shipping Data ( Optional )
const shipping_data = {
    first_name: "", // String
    last_name: "", // String
    mobile_no: "", // String
    address1: "", // String
    address2: "", // String
    address3: "", // String
    pincode: "", // String
    city: "", // String
    state: "", // String
    country: "", // String
}

    // UDF data ( Optional )
const udf_data = {
    udf_field_1: "", // String
    udf_field_2: "", // String
    udf_field_3: "", // String
    udf_field_4: "", // String
    udf_field_5: "", // String
}

    // Payment Modes That Needs To Be Shown ( Mandatory )
const payment_mode = {
    netbanking: true, // Boolean
    cards: true, // Boolean
```

```
    emi: true, // Boolean
    upi: true, // Boolean
    cardless_emi: true, // Boolean
    wallet: true, // Boolean
    debit_emi: true, // Boolean
    prebooking: true, // Boolean
    bnpl: true, // Boolean
    paybypoints: false, // Boolean
}

    // Product Details (SKU is mandatory during emi)
const product_details = [
    {
        "product_code": "testSKU1", // String
        "product_amount": 500000 // Integer
    },
    {
        "product_code": "testSKU1", // String
        "product_amount": 500000 // Integer
    }
]
```

Order Creation

Using the instance of the SDK we created above we will call the `.create()` method on the `payment` interface for creating an order with the provided parameters. It takes the following positional arguments

1. Transaction Data
2. Payment Modes
3. Customer Data
4. Billing Data
5. Shipping Data
6. UDF Data
7. Product Details

The `create()` method returns a promise with the response or else throws an error if something went wrong.

```
// Create Order
pinelabs.payment.create(txn_data, payment_mode, customer_data, billing_data,
shipping_data, udf_data, product_details).then((data) => {
    console.log(data)
});
```

Success Response


```
{
  "status": true,
  "redirect_url": "https://uat.pinepg.in/pinepg/v2/process/payment?token=S01w
PSlIH%2bopelRVif7m7e4SgrTRIcKYx25YDYfmgtbPOE%3d"
}
```


Failure Response

```
Fatal error: Uncaught Exception: MERCHANT PASSWORD DOES NOT MATCH
```

Fetch Order

Using the instance of the SDK we created above we will call the `.fetch()` method on the `payment` interface for fetching an order details with the provided transaction id and transaction type. It takes the following positional arguments

1. Transaction ID
2. Transaction Type

```
pinelabs.payment.fetch("650acb67d3752", 3).then((data) => {
    console.log(data)
});
```

Success Response


```
{
  "ppc_MerchantID": "106600",
  "ppc_MerchantAccessCode": "bcf441be-411b-46a1-aa88-c6e852a7d68c",
  "ppc_PinePGTxnStatus": "7",
  "ppc_TransactionCompletionDateTime": "20\/09\/2023 04:07:52 PM",
  "ppc_UniqueMerchantTxnID": "650acb67d3752",
  "ppc_Amount": "1000",
  "ppc_TxnResponseCode": "1",
  "ppc_TxnResponseMessage": "SUCCESS",
  "ppc_PinePGTransactionID": "12069839",
  "ppc_CapturedAmount": "1000",
  "ppc_RefundedAmount": "0",
  "ppc_AcquirerName": "BILLDESK",
  "ppc_DIA_SECRET": "D640CFF0FCB8D42B74B1AFD19D97A375DAF174CCBE9555E40CC62369
64928896",
  "ppc_DIA_SECRET_TYPE": "SHA256",
  "ppc_PaymentMode": "3",
  "ppc_Parent_TxnStatus": "4",
  "ppc_ParentTxnResponseCode": "1",
  "ppc_ParentTxnResponseMessage": "SUCCESS",
  "ppc_CustomerMobile": "7737291210",
  "ppc_UdfField1": "",
```

```
  "ppc_UdfField2": "",
  "ppc_UdfField3": "",
  "ppc_UdfField4": "",
  "ppc_AcquirerResponseCode": "0300",
  "ppc_AcquirerResponseMessage": "NA"
}
```


Failure Response


```
{
  "ppc_MerchantID": "106600",
  "ppc_MerchantAccessCode": "bcf441be-411b-46a1-aa88-c6e852a7d68c",
  "ppc_PinePGTxnStatus": "-6",
  "ppc_TransactionCompletionDateTime": "21\/09\/2023 11:29:48 PM",
  "ppc_UniqueMerchantTxnID": "106600_2109202323294890763",
  "ppc_TxnResponseCode": "-40",
  "ppc_TxnResponseMessage": "INVALID DATA",
  "ppc_CapturedAmount": "0",
  "ppc_RefundedAmount": "0",
  "ppc_DIA_SECRET": "4B9DD62C1CE94C354E368A2DA1C51C2E8ED16ABDC46414B8AAA60F378CDCE390",
  "ppc_DIA_SECRET_TYPE": "SHA256"
}
```


Incorrect Merchant Details

```
"IP Access Denied"
```

EMI Calculator

Using the instance of the SDK we created above we will call the `.calculate()` method on the `emi` interface for fetching offers for EMI with the provided product details. It takes the following positional arguments

1. Transaction Data
2. Product Details

```
const txn_data = {
    amount_in_paisa: '1000',
}

const productsDetail = [
    {
        "product_code": "testproduct02",
        "product_amount": 10000
    }
];

const res = pinelabs.emi.calculate(txn_data, productsDetail).then((data) => {
```

```
    console.log(data)
});
```

Success Response


```
{
  "issuer": [
    {
      "list_emi_tenure": [
        {
          "offer_scheme": {
            "product_details": [
              {
                "schemes": [],
                "product_code": "testproduct02",
                "product_amount": 10000,
                "subvention_cashback_discount": 0,
                "product_discount": 0,
                "subvention_cashback_discount_percentage": 0,
                "product_discount_percentage": 0,
                "subvention_type": 3,
                "bank_interest_rate_percentage": 150000,
                "bank_interest_rate": 251
              }
            ],
            "emi_scheme": {
              "scheme_id": 48040,
              "program_type": 105,
              "is_scheme_valid": true
            }
          },
          "tenure_id": "3",
          "tenure_in_month": "3",
          "monthly_installment": 3417,
          "bank_interest_rate": 150000,
          "interest_pay_to_bank": 251,
          "total_offerred_discount_cashback_amount": 0,
          "loan_amount": 10000,
          "auth_amount": 10000
        },
        {
          "offer_scheme": {
            "product_details": [
              {
                "schemes": [],
                "product_code": "testproduct02",
                "product_amount": 10000,
                "subvention_cashback_discount": 0,
```

```
                "product_discount": 0,
                "subvention_cashback_discount_percentage": 0,
                "product_discount_percentage": 0,
                "subvention_type": 3,
                "bank_interest_rate_percentage": 150000,
                "bank_interest_rate": 440
              }
            ],
            "emi_scheme": {
              "scheme_id": 48040,
              "program_type": 105,
              "is_scheme_valid": true
            }
          },
          "tenure_id": "6",
          "tenure_in_month": "6",
          "monthly_installment": 1740,
          "bank_interest_rate": 150000,
          "interest_pay_to_bank": 440,
          "total_offerred_discount_cashback_amount": 0,
          "loan_amount": 10000,
          "auth_amount": 10000
        }
      ],
      "issuer_name": "HDFC",
      "is_debit_emi_issuer": false
    }
  ],
  "response_code": 1,
  "response_message": "SUCCESS"
}
```


Failure Response

```
Fatal error: Uncaught Exception: INVALID DATA,MISMATCH_IN_TOTAL_CART_AMOUNT_AND_TOTAL_PRODUCT_AMOUNT
```

Verify Hash

Using the instance of the SDK we created above we will call the `.verify()` method on the `hash` interface for verifying a hash received in the response of callback and webhooks. It takes the following positional arguments

1. Hash Received in Response (DIA_SECRET)
2. Response Received (Not including DIA_SECRET and DIA_SECRET_TYPE)

```
const isVerified = pinelabs.hash.verify("D640CFF0FCB8D42B74B1AFD19D97A375DAF174CCBE9555E40CC6236964928896", response);
console.log(isVerified);
```

React Js :-

React.js itself does not handle TLS directly, as it's a JavaScript library for building user interfaces.

The TLS support is typically handled by the underlying server or service your React.js application communicates with.

Note :-

Please note no additional charges like TDR, GST etc are handled in our Plugins and the same need to be manually handled at merchant end