Pinelabs Java SDK Integration

This document explains the integration process of the Pinelabs Java SDK for your Java Applications

By Pinelabs Team

Prerequisites

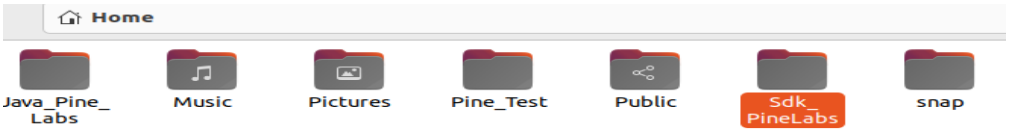
Open JDK Version >=17

Installation Process

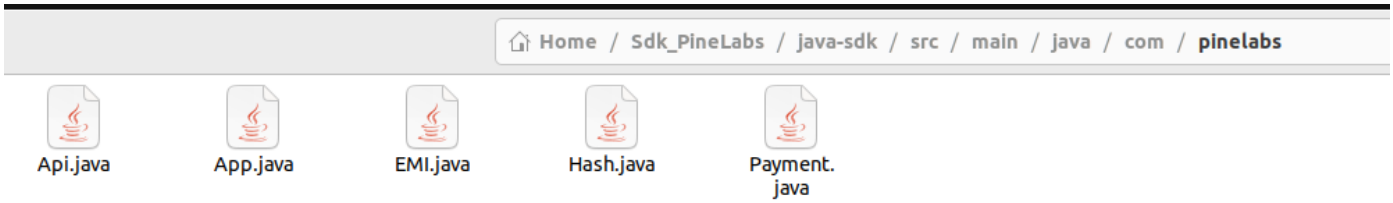
This section explains the installation flow of the Java SDK for dealing with Pine Lab Apis

Given below are the steps for installing the SDK in your project:

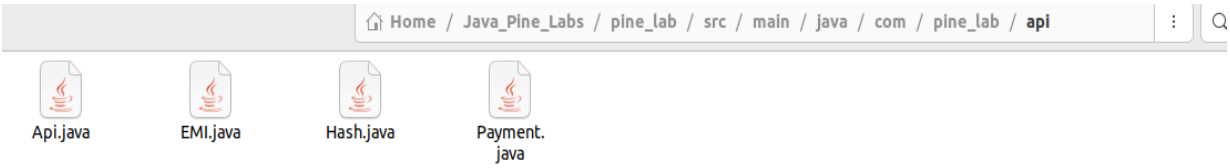
1. Extract the SDK (Zip File) in any location on your system. Here we have extracted it inside Sdk_PineLabs Folder (Sdk Folder - java-sdk)



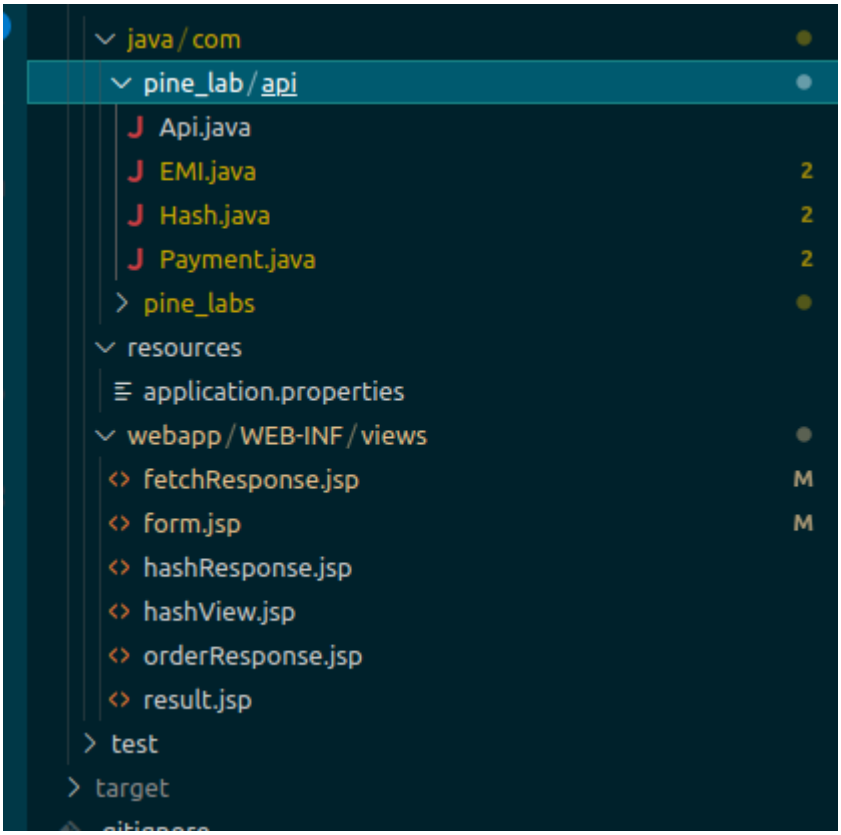
2. Now we need to come into this path src/main/java/com/pine_lab/api inside java-sdk folder and in that we'll be able to locate all the necessary classes, So Copy classes except App.java (Testing Class) into your existing project's repository or package.



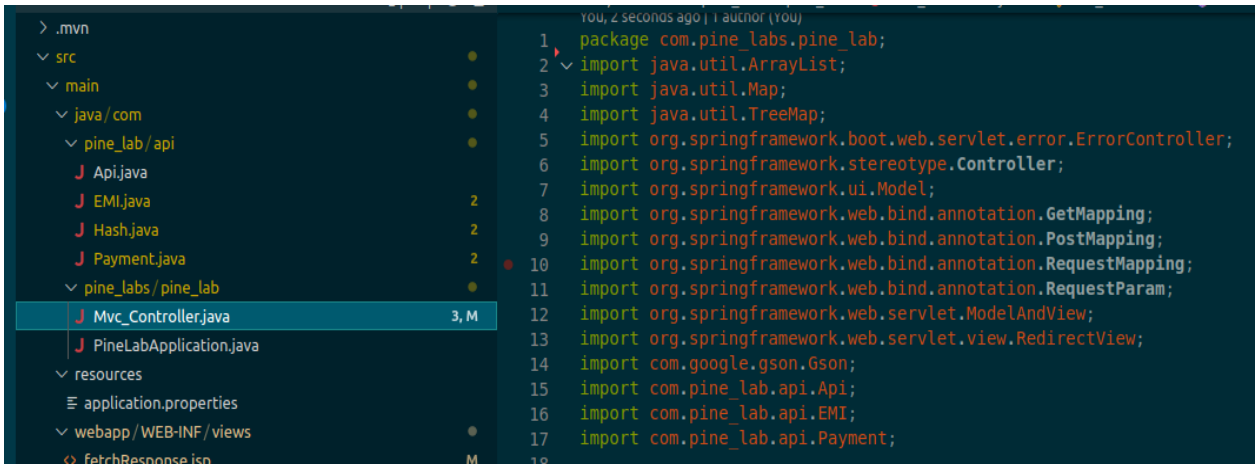
3. For sample we have imported files in our Sample Springboot Project (Java_Pine_Labs), and created a new package pine_lab/api and injected all classes of sdk into it.



4. So it'll look something like this in your IDE, where your main classes are also located in other packages. In this other main controller classes are located in com/pine_labs & sdk files have been added into com/pine_lab/api.



5. After that you need to import classes into your main controller class, like in our Sample Springboot Project we have included classes (import com.pine_lab.api.Api;) in that way you need to import other classes as well.



6. Then you need to copy all the dependencies in your current project's Pom.xml file from Sdk's Pom.xml file. For reference below image shows xml file of SDK.


```
    <dependency>
      <groupId>com.squareup.okhttp3</groupId>
      <artifactId>okhttp</artifactId>
      <version>4.9.1</version>
  </dependency>
    <dependency>
      <groupId>com.fasterxml.jackson.core</groupId>
      <artifactId>jackson-databind</artifactId>
      <version>2.13.0</version>
  </dependency>
  <dependency>
    <groupId>javax.xml.bind</groupId>
    <artifactId>jaxb-api</artifactId>
    <version>2.3.1</version>
</dependency>
    <dependency>
      <groupId>org.apache.httpcomponents</groupId>
      <artifactId>httpclient</artifactId>
      <version>4.5.13</version>
    </dependency>
    <dependency>
      <groupId>com.google.code.gson</groupId>
      <artifactId>gson</artifactId>
      <version>2.8.8</version>
    </dependency>
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>javax.servlet-api</artifactId>
      <version>4.0.1</version>
      <scope>provided</scope>
  </dependency>
```

Usage Process

This section explains the usage flow of the Java SDK

Now we have the classes of SDK imported into our Project's newly created Package named api in path com/pine_lab

1. Api Class- Used for initializing object containing Merchant details so we don't have to repeat process of passing these details into every class.
2. Payment Class – Used for creating payment links and fetching payment details for a transaction id
3. EMI Class – Used for calculation of EMI data for a product
4. Hash Class – Used for hash verification in your application for secure development

Create Order Process

In order for us to create an order and generate a URL for payment we need to use the "Payment Class",

1. Create an instance of the Api class first and pass the following parameters
    a. Merchant Id of the merchant provided by Pinelabs team (Integer)
    b. Merchant Access Code also provided by Pinelabs team (String)
    c. Merchant Secret also provided by Pinelabs team (String)
    d. Pg Mode which will be of type boolean (True for Testing and False for Production)

```
Integer mid= 106598;
String apiAccessCode="4a39a6d4-46b7-474d-929d-21bf0e9ed607";
String secret_key="55E0F73224EC458A8EC0B68F7B47ACAE";
boolean isTest=true; //false for production
```

```
Api api = new Api(mid, apiAccessCode, secret_key, isTest);
Payment apiPayment = api.payment();
EMI emiCalculaor=api.emi();
```

2. Now you need to call payment method with Api class's object and assign it to Payment class's instance and then you need to call the "create" method on the instance of the Payment class and pass the following arguments
    a. Txn Id- Required, String
    b. Amount- Long
    c. Callback Url- String
    d. Customer Data – Required: Customer information including the first name, last name, mobile number, customer id, email, Shipping Data, Billing Data etc. (Map Data Structure)
    e. Billing Data – Optional: Billing details of the customer (Map Data Structure)
    f. UDF Data – Optional: UDF data which can be anything extra you want to include in the order (Map Data Structure)
    g. Payment Modes – Required: Modes of payments you want to be enabled on the checkout using the payment (ArrayList of Strings containing mode names→ cards,netbanking,emi,cardless_emi,upi,debit_emi,prebooking,bnpl,wallet)

h. Product Details- Required: List of Map objects of product details.

Amount(Mandatory)

```
Long amount=1000l;
```

Unique Transaction Id(Mandatory)

```
String txnId="txnID17ass1aassdd6r6r";
```

Callback Url(Mandatory)

```
String merchant_return_url="http://localhost:8000/api/callback/result";
```

Customer Data(Optional/Null)

```
//If no data point it to null like Map<String,Object> customerData=null;
Map<String,Object> customerData=new TreeMap<String,Object>();
customerData.put("email_id","");customerData.put("first_name","");customerData.put("last_name","");customerData.
customerData.put("mobile_no","");
Map<String,Object> billing_data=new TreeMap<String,Object>();
billing_data.put("address1","");billing_data.put("address2","");billing_data.put("address3","");billing_data.put
billing_data.put("city","");billing_data.put("state","");billing_data.put("country","");
Map<String,Object> shipping_data=new TreeMap<String,Object>();
shipping_data.put("first_name","");shipping_data.put("last_name","");shipping_data.put("mobile_no","");shipping_
shipping_data.put("address2","");shipping_data.put("address3","");shipping_data.put("pincode","");shipping_data.
shipping_data.put("state","");shipping_data.put("country","");
customerData.put("billing_data",billing_data);customerData.put("shipping_data",shipping_data);
```

Udf Data(Optional)

```
//If no data point it to null like Map<String,Object> udf_data=null;
Map<String,Object> udf_data=new TreeMap<String,Object>();
udf_data.put("udf_field_1","");udf_data.put("udf_field_2","");udf_data.put("udf_field_3","");udf_data.put("udf_f
udf_data.put("udf_field_5","");
```

Payment Modes(Mandatory)

```
//Some modes might not be supported on keys that you'll provide for creating payment link!
String[] paymentModes=["cards","netbanking","emi","upi","cardless_emi","bnpl","debit_emi"];
```

Product Details(Optional)

```
ArrayList<HashMap<String,Object>> product_list=new ArrayList<>();
HashMap<String,Object> products=new HashMap<String,Object>();
products.put("product_code","testprod01");products.put("product_amount",500000);
product_list.add(products);
```

Create a order

```
ntResponse = apiPayment.create(txnId, amount,merchant_return_url,customerData,udf_data,paymentModes,product_list);
ntResponse.toString());
```

3. It will return the following once executed
    a. **Token**: Payment order token for processing of order (If it was success)
    b. **URL**: URL using which the user can be directly redirected to the payment page where he/she can make the payment. (If it was success)
    c. **Status:** Status of the order create api status if it worked successfully or not.

4. You need to extract response key's value which will be having all the value above from Map.
   There will be another key other than response so just don't include that and extract response key's value which will be having (Token, Redirect Url, Response Code, Response Message)

Success Response

```
{response={"token":"S01Nox%2f6%2bge3%2fZLU10i35LmRBFO8jX8RKaa5K1J%2fwkyg7s%3d","redirect_url":"https://uat.pinep
```

Failure Response :

```
{response={"respone_message":"SECURE HASH RETURNED FROM MERCHANT NOT MATCH","respone_code":-34}}
```

Fetch Order Process

In order for us to fetch an order detail we need to use the "Payment Class"

1. First create an instance of the class and refer it to Api class's object and pass parameters shown below into it's constructor:
    a. Merchant Id of the merchant provided by Pinelabs team (Integer)
    b. Merchant Access Code also provided by Pinelabs team (String)
    c. Merchant Secret also provided by Pinelabs team (String)
    d. Pg Mode which will be of type boolean (True for Testing and False for Production)

2. Create Payment class's object, call Api class's payment function and assign it to this payment class's object.

```
Api api = new Api(mid, apiAccessCode, secret_key, isTest);
Payment apiPayment = api.payment();
```

3. Now call fetch function using Payment class's reference variable (apiPayment) and pass below parameters.
    a. Transaction ID: Transaction id which was used during the order creation process explained at the top (string)
    b. Transaction Type: Type of the transaction which was passed at the order creation process. This needs to be the same otherwise it won't work. (Integer)

Transaction Id(Mandatory)

```
String txnId="txnID17ass1aassdd6r6r";
```

Transaction Type(Mandatory)

```
Integer txnType=3;//For Enquiry
```

Fetch Order Method Call

```
Map<String,Object> enquiryResponse = apiPayment.fetch("txnID17as1aassdd6r6r", 3);
```

4. Response

Fetch Order Method Call

```
Map<String,Object> enquiryResponse = apiPayment.fetch("txnID17as1aassdd6r6r", 3);
System.out.println(enquiryResponse.toString());
```

Success Response

```
{response={"ppc_MerchantID":"106598","ppc_MerchantAccessCode":"4a39a6d4-46b7-474d-929d-21bf0e9ed607","ppc_PinePG
```

EMI Details Process

In order for us to fetch EMI details for any product we need to use the "EMI Class"

1. First create an instance of the class and refer it to Api class's object and pass parameters shown below into it's constructor:
    a. Merchant Id of the merchant provided by Pinelabs team (Integer)
    b. Merchant Access Code also provided by Pinelabs team (String)
    c. Merchant Secret also provided by Pinelabs team (String)
    d. Pg Mode which will be of type boolean (True for Testing and False for Production)

2. Create Emi class's object, call Api class's emi function and assign it to this emi class's object.

3. Now call emiCalculation method using EMI class's object and then you need to pass product_details as list of map (For Multi or Single Cart), and in map it should have product_code,product_amount as keys as shown in image below and along with that you need to pass total amount as passed in amount_in_paisa variable.

Amount(Mandatory)

```
//Total Amount which should match total amount value of Products
Long amount=1000l;
```

Product Details (Mandatory)

```
ArrayList<TreeMap<String,Object>> product_details=new ArrayList<TreeMap<String,Object>>();
TreeMap<String,Object> products=new TreeMap<String,Object>();
products.put("product_code","testprod01");products.put("product_amount",500000);
product_details.add(products);
```

Emi Calculator Method Call

```
Map<String,Object> emiResponse = emiCalculaor.emiCalculation(amount,product_details);
System.out.println(emiResponse.toString());
```

4. Once executed it will return the following response

Success Response

```
{response={"issuer":[{"list_emi_tenure":[{"offer_scheme":{"product_details":[{"schemes":[],"product_code":"testp
```

Failure Response

```
{Exception:Invalid Data}
```

Hash Verification Process

In order for us to verify hash of all the responses we get from the SDK to make sure there was no tampering with the response we need to use the "Payment Class"

1. First create an instance of the class and refer it to Api class's object and pass parameters shown below into it's constructor:
    a. Merchant Id of the merchant provided by Pinelabs team (Integer)
    b. Merchant Access Code also provided by Pinelabs team (String)
    c. Merchant Secret also provided by Pinelabs team (String)
    d. Pg Mode which will be of type boolean (True for Testing and False for Production)

2. Create Payment class's object, call Api class's payment function and assign it to this payment class's object.
3. Now call verifyHash function through newly created Payment class's object and pass the following params to the method
    a. **Hash (secret_dia):** Pass hash you want to verify
    b. **Response Body**: Pass the entire response body **without the hash and hash type**

RequestObject(Mandatory)

```
TreeMap<String,Object> dataMap=new TreeMap<>();
dataMap.put("ppc_MerchantID", "106600");dataMap.put("ppc_MerchantAccessCode", "bcf441be-411b-46a1-aa88-c6e852a7d
dataMap.put("ppc_TransactionCompletionDateTime", "20/09/2023 04:07:52 PM");dataMap.put("ppc_UniqueMerchantTxnID"
dataMap.put("ppc_Amount", "1000");dataMap.put("ppc_TxnResponseCode", "1");dataMap.put("ppc_TxnResponseMessage",
```

Hash(Mandatory)

```
String hash="D640CFF0FCB8D42B74B1AFD19D97A375DAF174CCBE9555E40CC62369649288896";
```

Hash Verification Method Call

```
boolean hashVerification=apiPayment.verifyHash(hash,dataMap);
System.out.println(hashVerification);
```

4. Once Executed it'll return boolean response whether it was verified or not (True/False)

TDR/GST Information:

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

TLS 1.2 information:

Java 8 and above provide TLS 1.2 support.
Note: The exact version might depend on the specific Java vendor (Oracle, OpenJDK, etc.).