Strictly Confidential

PayU
Integration Document

7th Floor, Pearl Towers
Plot 51, Sector 32
Gurgaon, 122002
India
T: 0124-6749078
F: 0124-6749101

Table of Contents (Click on the topic for direct access)

OVERVIEW 4

PayU Payment Gateway 4

Payment Process Flow 4

SECTION I: WEBSITE INTEGRATION 5

Steps for Integration Process 6

Parameters to be posted by Merchant to PayU in Transaction Request 8

Seamless Integration – Parameters in Transaction Request 17

Additional Charges – Convenience Fee Model (To be used only if recommended by Account Manager at PayU) 18

Method 1: Enabled from backend at PayU 18

Method 2: Merchant Calculates and Posts Additional Charges to PayU 18

Important Things to remember: Characters allowed for parameters 20

Formula for hash (checksum) before transaction 20

Formula for hash (checksum) after transaction 20

Hash (Checksum) Algorithm Example codes 20

For PHP 20

For .NET 20

For JSP 21

Response Parameters posted by PayU to Merchant 22

Shopping Cart Integration Kits 26

Platform based Integration kits 26

SECTION II: WEB SERVICES – APIs 27

Web Service Request Format: 27

Web Service Response Format 28

LIST OF APIs AND THEIR DESCRIPTION 28

1) verify_payment 28

2) check_payment 29

3) cancel_refund_transaction 30

4) check_action_status 31

5) capture_transaction 32

6) update_requests 33

7) cod_verify 34

8) cod_cancel 35

9) cod_settled 36
10) get_TDR: 37
11) udf_update 37
12) create_invoice 38
13) check_offer_status (1st Usage) 40
14) check_offer_status (2nd Usage) 41
15) getNetbankingStatus 43
16) getIssuingBankStatus 44
20) get_user_cards 52
21) save_user_card 53
22) edit_user_card 54
23) delete_user_card 55

OVERVIEW

This document describes the **steps** for technical integration process between merchant website and PayU Payment Gateway for enabling online transactions. This document is covered in two sections. Section I covers **website integration** and Section II covers **APIs** provided to the merchants.

PayU Payment Gateway

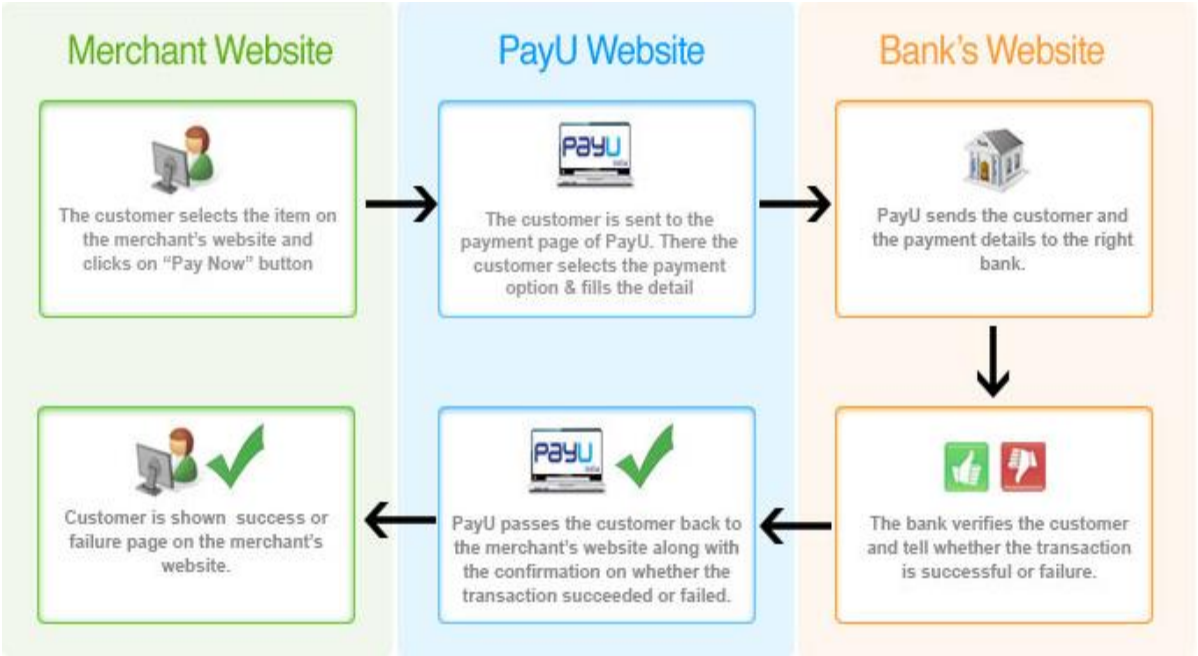
PayU offers electronic payment services to merchant website through its partnerships with various banks and payment instrument companies. Through PayU, the customers would be able to make electronic payments through a variety of modes which are mentioned below:

- Credit cards
- Debit cards
- Online net banking accounts
- EMI payments
- Cash Cards
- Email Invoicing
- IVR
- Cash on Delivery (COD)

PayU also offers an **online interface** (known as **PayU Dashboard**) where the merchant has access to various features like viewing all the transaction details, settlement reports, analytical reports etc. Through this interface, the merchant can also execute actions like capturing, cancelling and refunding the transactions. This online interface can be accessed through https://www.payu.in by using the username and password provided to you.

Payment Process Flow

The following diagram explains how the customer makes the payment and how the process flows:



SECTION I: WEBSITE INTEGRATION

The merchant can integrate with PayU by using one of the three different methods of Integration:

1) **Non-Seamless Integration** – In this mode during the transaction, the customer would be re-directed from merchant website to PayU payment page. On the PayU payment page, he would need to select the payment option and enter the respective card details. After this, PayU would re-direct the customer to the desired bank webpage for further authentication.

2) **Seamless Integration** - In this mode, the merchant needs to collect the customer card details on their own website and post them to PayU. Here, the customer would not be stopped at PayU payment page at all, as the payment option and card details are already received from the merchant. The merchant must be **PCI-DSS certified** in this case. For further information on PCI-DSS certification please contact your Account Manager at PayU.

Also, the merchant website can be based either on a **shopping cart** or can be developed by the merchant **(not based upon any shopping cart).** Based on the type (out of these two), PayU would provide integration kit (code) to the merchant which they needs to incorporate at their end. The list of Integration kits supported by PayU at present is mentioned in later sections of the document.

Steps for Integration Process

The steps for integrating with PayU can technically be described as below:

1) To start off the integration process, you would be provided a **test setup** by PayU where you would be given a test merchant account and test credit card credentials to have a first-hand experience of the overall transaction flow. Here, you need to make the transaction request on our **test server (and not the production server)**. Once your testing is complete, **then only** you will be ready to move to the PayU production server.

2) To initiate a transaction, the merchant needs to generate a **POST REQUEST** - which must consist of mandatory and optional parameters mentioned in the **later section**. This POST REQUEST needs to be hit on the below mentioned PayU URLs:

   For PayU Test Server:

   POST URL: **https://test.payu.in/_payment**

   For PayU Production (LIVE) Server:

   POST URL: **https://secure.payu.in/_payment**

3) In the merchant initiated **POST REQUEST**, one of the mandatory parameters is named as **hash.** The details of this hash parameter have been covered in the later section. But it is **absolutely critical** for the merchant to calculate the hash correctly and post to us in the request.

4) When the transaction **POST REQUEST** hits the PayU server, a new transaction entry is created in the PayU Database. To identify each new transaction in the PayU Database, a unique identifier is created every time at PayU's end. This identifier is known as the **PayU ID (or MihPayID).**

5) With the POST REQUEST, customer would be re-directed to PayU's payment page. Customer now selects the particular payment option on PayU's page (Credit Card/Debit Card/Net Banking etc) and clicks on 'Pay Now'. PayU re-directs the customer to the chosen bank. The customer goes through the necessary authorization/authentication process at bank's login page, and the bank gives the success/failure response back to PayU.

6) PayU marks the transaction status on the basis of response received from Bank. PayU provides the final transaction response string to the merchant through a **POST RESPONSE.** The parameters in this response are covered in the subsequent sections.

7) In the POST RESPONSE sent by PayU, you would receive the final status of the transaction. You will receive the **hash** parameter here also. Similar to step 3, it is **absolutely crucial** to verify this hash value at your end and then only accept/reject the invoice order. This is done to strictly avoid any tampering attempt by the user.

DISCLAIMER:

1. Test URL: The Test URL is provided to PayU merchants to test the integration of their server with that of PayU or Bank. It is understood that since this is merely a Test URL, the Merchant should not treat any transactions done on this Test server as live and should not deliver the products/services with respect to any such test transactions even in the case your server receive a successful transaction confirmation from PayU/Bank.

2. Merchants are herein forth requested to set up required control checks on their (merchant) systems/servers to ensure that only those transactions should get routed to the PayU test server which are initiated with sole intention of test the environment.

Parameters to be posted by Merchant to PayU in Transaction Request

Sr. No	Variable	Description
1)	key **(Mandatory)**	This parameter is the unique Merchant Key provided by PayU for your merchant account. The Merchant Key acts as the unique identifier (primary key) to identify a particular Merchant Account in our database. While posting the data to us, you need to put this Merchant Key value for you merchant account in this parameter. Also, please note that during integration with PayU, you would need to first integrate with our Test Server. PayU would be providing you the necessary Merchant Key for test server. Please do not use your live account's merchant key here. It would not work. Once testing is done, you are ready to move to live server. Here, you would need to replace the test Merchant Key with Live Merchant Key. This is a critical step for successfully moving to live PayU server. Example: C0Ds8q
2)	txnid **(Mandatory)**	This parameter is known as Transaction ID (or Order ID). It is the order reference number generated at your (Merchant's) end. It is an identifier which you (merchant) would use to track a particular order. If a transaction using a particular transaction ID has already been successful at PayU, the usage of same Transaction ID again would fail. Hence, it is essential that you post us a unique transaction ID for every new transaction. (Please make sure that the transaction ID being sent to us hasn't been successful earlier. In case of this duplication, the customer would get an error of 'duplicate Order ID'). Data Type – Varchar Character Limit – 25 characters Example: fd3e847h2
3)	amount **(Mandatory)**	This parameter should contain the payment amount of the particular transaction. Note: Please type-cast the amount to float type Example: 10.00
4)	productinfo **(Mandatory)**	This parameter should contain a brief product description. It should be a string describing the product (The description type is entirely your choice). Data type - Varchar Character Limit – 100 characters Example: tshirt100
5)	firstname **(Mandatory)**	Self-Explanatory (Must contain the first name of the customer) Data Type – Varchar

		Character Limit – 60 characters Example: Ankit
6)	email **(Mandatory)**	Self-explanatory (Must contain the email of the customer) Data type – Varchar Character Limit – 50 Example: test@gmail.com
7)	phone **(Mandatory)**	Self-explanatory (Must contain the phone number of the customer) Data type – Varchar Character Limit – 50 (numeric value only) Example: 9999999999
8)	lastname	Self-Explanatory (only alphabets a-z are allowed) Data Type – Varchar Character Limit – 20 characters Example: Verma
9)	address1	Self-Explanatory Data Type – Varchar Character Limit – 100 Characters allowed : A to Z, a to z, 0 to 9, @, - (Minus), _ (Underscore), / (Backslash), (Space), (Dot)
10)	address2	Self-explanatory Data Type – Varchar Character Limit – 100 (Allowed characters are same as for address1 parameter)
11)	city	Self-explanatory Data type – Varchar Character Limit – 50 (Allowed characters are same as for address1 parameter)
12)	state	Self-explanatory Data type – Varchar Character Limit – 50 (Allowed characters are same as in address parameter)
13)	country	Self-explanatory Data type – Varchar Character Limit – 50 (Allowed characters are same as in address parameter)
14)	zipcode	Self-explanatory Data type – Varchar Character Limit – 20 (Only numeric value allowed)

15)	udf1	User defined field 1 – This parameter has been made for you to keep any information corresponding to the transaction, which may be useful for you to keep in the database. UDF1-UDF5 fields are for this purpose only. It's completely for your usage and you can post any string value in this parameter. udf1-udf5 are optional parameters and you may use them only if needed Data type – Varchar Character Limit – 255
16)	udf2	User defined field 2 – Same description as UDF1 Data type – Varchar Character Limit – 255
17)	udf3	User defined field 3 – Same description as UDF1 Data type – Varchar Character Limit – 255
18)	udf4	User defined field 4 – Same description as UDF1 Data type – Varchar Character Limit – 255
19)	udf5	User defined field 5 – Same description as UDF1 Data type – Varchar Character Limit – 255
20)	surl (Mandatory)	Success URL - This parameter must contain the URL on which PayU will redirect the final response if the transaction is successful. The response handling can then be done by you after redirection to this URL
21)	furl (Mandatory)	Failure URL - This parameter must contain the URL on which PayU will redirect the final response if the transaction is failed. The response handling can then be done by you after redirection to this URL
22)	curl	Cancel URL - This parameter should contain the URL on which PayU will redirect the response if the transaction is cancelled by the customer on PayU page. The response handling can then be done by you after redirection to this URL
23)	hash (Checksum) **(Mandatory)**	Hash is a crucial parameter – used specifically to avoid any tampering during the transaction. There are two different methods to calculate hash. **Please follow method 1 only.** Method 2 is just there for the documentation and is not to be used. Method 1 - This is the simplest way of calculating the hash value. Here, please make sure that the **api_version** parameter is **NOT POSTED** from your end. For hash calculation, you need to generate a string using certain parameters and apply the sha512 algorithm on this string. Please note that you have to use pipe (|) character in between these parameters as mentioned below. The parameter order is mentioned below: **sha512(key|txnid|amount|productinfo|firstname|email|udf1|udf2|udf3|udf4|u**

		df5|||||SALT) All these parameters (and their descriptions) have already been mentioned earlier in this table. Here, SALT (to be provided by PayU), key, txnid, amount, productinfo, firstname, email are **mandatory** parameters and hence **can't be empty** in hash calculation above. But, udf1-udf5 are **optional** and hence you need to calculate the hash based upon the fact that whether you are posting a particular udf or not. For example, if you are NOT posting udf1. Then, in the hash calculation, udf1 field will be left empty. Following examples will clarify various scenarios of hash calculation: Case 1: If all the udf parameters (udf1-udf5) are posted by the merchant. Then, hash=sha512(**key|txnid|amount|productinfo|firstname|email|udf1|udf2|udf3|udf4|udf5|||||SALT**) Case 2: If only some of the udf parameters are posted and others are not. For example, if udf2 and udf4 are posted and udf1, udf3, udf5 are not. Then, hash=sha512(**key|txnid|amount|productinfo|firstname|email||udf2||udf4|||||SALT**) Case 3: If NONE of the udf parameters (udf1-udf5) are posted. Then, hash=sha512(**key|txnid|amount|productinfo|firstname|email||||||||||SALT**) Example: If key=**C0Dr8m**, txnid=**12345**, amount=**10**, productinfo=**Shopping**, firstname=**Test**, email=**test@test.com**, udf2=**abc**, udf4=**15**, SALT=**3sf0jURk** and udf1, udf3, udf5 are **not** posted. Then, hash would be calculated as Case 2 above: **sha512(C0Dr8m|12345|10|Shopping|Test|test@test.com||abc||15|||||||3sf0jURk)** (This value comes out to be *ffcdbf04fa5beefdcc2dd476c18bc410f02b3968e7f4f54e8f43f1e1a310bb32e3b4dec9305232bb89db5b1d0c009a53bcace6f4bd8ec2f695baf3d43ba730ce*) IMPORTANT: For details related to hash at the time of post back from PayU to the merchant, please refer to later section. This is also absolutely mandatory to avoid any tampering. Method 2- Second method for hash calculation (***Don't use this method. It is only for internal documentation***). Here, parameter **api_version** should be equal to 2. hash = sha512(**key|txnid|amount|offer_key|api_version|SALT**)

<table>
<tr><td>24)</td><td>pg</td><td>This parameter signifies the payment category (tab) that you want the customer to see by default on the PayU page. Hence if PG='NB', then after redirection to PayU's payment page, the Net Banking option would be opened by default.

(PG parameter may take different values like : NB for Net Banking tab, CC for Credit Card tab, DC for Debit Card tab, CASH for Cash Card tab and EMI for EMI tab)

Note: PG = CC, i.e. Credit Card tab is recommended. If PG is left empty, CC will be taken as default.</td></tr>
<tr><td>25)</td><td>codurl</td><td>Cash on delivery URL – This parameter is used when a transaction attempt fails. In this case, if retries have been enabled for you (done by PayU for your merchant account), our PayU page is shown (to provide another attempt to customer to complete the transaction) with the 'failed transaction message' to the customer and also 'Pay by COD' option. To handle this 'Pay by COD' option, you can fill the COD URL parameter with a URL which we will redirect to, when the customer selects this option. This way, you can then provide the customer another attempt at the transaction through this URL.</td></tr>
<tr><td>27)</td><td>drop_category</td><td>This parameter is used to customize the payment options for each individual transaction. For example, if we consider the categories Credit Card, Debit Card and Net Banking for a merchant. If there are 30 net banking options available and the merchant wants to drop 2 of those net banking options (i.e. do not display those 2 options on PayU page), then drop_category parameter can be used effectively. Below table denotes example of category and sub-categories at PayU

<table>
<tr><th>Category</th><th>Sub-category</th></tr>
<tr><td>Credit Card</td><td>MasterCard, Amex, Diners etc</td></tr>
<tr><td>Debit Card</td><td>Visa, Mastercard, Maestro etc</td></tr>
<tr><td>Net Banking</td><td>SBI Net Banking, HDFC Net Banking etc</td></tr>
<tr><td>EMI</td><td>CITI 3 Months EMI, HDFC 6 Months EMI etc</td></tr>
<tr><td>Cash Card</td><td>AirtelMoney, YPay, ITZ Cash card etc</td></tr>
</table>

Now, to drop the whole category, please use the following values:

<table>
<tr><th>Category</th><th>Value of 'drop_category' parameter</th></tr>
<tr><td>Credit Card</td><td>CC</td></tr>
<tr><td>Debit Card</td><td>DC</td></tr>
<tr><td>Net Banking</td><td>NB</td></tr>
<tr><td>EMI</td><td>EMI</td></tr>
</table>
</td></tr>
</table>

		Cash Card | CASH To drop sub-categories, please use the respective bank codes for them. Please contact PayU to get the respective bank codes. Also note that the delimiter for categories is comma (,) character and for sub-categories it is the pipe (|) character. Examples for usage: **drop_category - DC|VISA|MAST, NB|ICIB** : Here, for debit card category, only Visa and Master Card options would be dropped (and hence not displayed on the PayU page). In Net Banking option, only ICICI Net Banking would be dropped. All other active payment options would be displayed. **drop_category - CC|AMEX, DC|VISA, EMI|EMI6** : Here, for credit card category, only AMEX option would be dropped (and hence not displayed). In debit card category, only VISA option would be dropped. And in EMI category, only HDFC 6 months EMI option (bank code – EMI6) would be dropped. All the other active payment options would be displayed. **Note:** Please make sure to use this parameter only after testing properly as an incorrect string will lead to undesirable payment options being displayed.
28)	enforce_paymethod	This parameter allows you to customize the payment options for each individual transaction. For example, if we consider the categories Credit Card, Debit Card and Net Banking. If the merchant wants to display only 4 debit card options and only 2 Net Banking options for a transaction A and wants to display only 2 debit card option and 5 Net Banking options for another transaction B, the customization is needed and this parameter (enforce_paymethod) provides exactly that feature. The merchant needs to put the necessary payment options in this parameter and post it to us at the time of transaction. All the categories and sub-categories have specific values which need to be put in this string. The categories/subcategories are as follows: **Category** | **Sub-category** Credit Card | MasterCard, Amex, Diners etc Debit Card | Visa, Mastercard, Maestro etc Net Banking | SBI Net Banking, HDFC Net Banking etc EMI | CITI 3 Months EMI, HDFC 6 Months EMI etc Cash Card | AirtelMoney, YPay, ITZ Cash card etc Now, to enforce complete categories, please use the following values:

		<table><tr><th>Category</th><th>Value of enforced_paymethod</th></tr><tr><td>Credit Card</td><td>Creditcard</td></tr><tr><td>Debit Card</td><td>Debitcard</td></tr><tr><td>Net Banking</td><td>Netbanking</td></tr><tr><td>EMI</td><td>Emi</td></tr><tr><td>Cash Card</td><td>Cashcard</td></tr></table> To enforce sub-categories, please use the respective bank codes for them. Please contact PayU to get the respective bank codes. Please note that the delimiter is pipe (|) character here. Examples: **creditcard|debitcard|HDFB|AXIB** – Here, all the credit card and debit card options would be displayed (as the whole category is enforced). In Net Banking category, only HDFC and AXIS Net Banking would be displayed. Rest of the categories would not be displayed at all (EMI, Cash card etc – as they are not being mentioned in the string). **creditcard|VISA|SMAE|netbanking|EMI6|EMI9|cashcard** – Here, all the credit card options, net banking options and cash card options would be displayed (as the whole category is enforced for these). In Debit card category, Visa and SBI Maestro payment options would be displayed (as bank codes for only these options are mentioned in the string). In EMI category, only HDFC EMI (for 6 and 9 months) would be displayed. Note: Please make sure to use this parameter only after testing properly as an incorrect string will lead to undesirable payment options being displayed.
29)	custom_note	This parameter is useful when you want to display a message string on the PayU Payment page. For example, if for a particular product X, you want your customer to know that an extra amount of Rs 100 would be charged afterwards, you can show the corresponding message on payment page. For this, you need to post that message in this parameter – custom_note. The note would be displayed just below the payment tabs (Credit Card/Debit Cards/Net Banking) For Example: custom_note = **You will be charged an extra amount of Rs 100 on this transaction** **Characters allowed:** A to Z, a to z, 0 to 9, % (percentage), , (comma), . (decimal), ' (apostrophe)
30)	note_category	This parameter gives you an option of showing the message string passed in custom_note parameter for only the selected Payment categories. Hence, this parameter should contain the comma separated list of the payment options for

		which the **custom_note** will appear. For example: **note_category = CC,NB** will show the custom_note for Credit Card & Net banking only
31)	api_version	Please don't use this parameter while posting the data
32)	shipping_firstname	**This parameter has to used in case of COD (Cash on Delivery) Only** Self-Explanatory (Constraints same as firstname parameter). If this parameter is posted, the corresponding value would be filled up automatically in the form under COD tab on PayU payment page
33)	shipping_lastname	**This parameter has to used in case of COD (Cash on Delivery) Only** Self-Explanatory (Constraints same as lastname parameter). If this parameter is posted, the corresponding value would be filled up automatically in the form under COD tab on PayU payment page
34)	shipping_address1	**This parameter has to used in case of COD (Cash on Delivery) Only** Self-Explanatory (Constraints same as address1 parameter). If this parameter is posted, the corresponding value would be filled up automatically in the form under COD tab on PayU payment page
35)	shipping_address2	**This parameter has to used in case of COD (Cash on Delivery) Only** Self-Explanatory (Constraints same as address2 parameter). If this parameter is posted, the corresponding value would be filled up automatically in the form under COD tab on PayU payment page
36)	shipping_city	**This parameter has to used in case of COD (Cash on Delivery) Only** Self-Explanatory (Constraints same as city parameter). If this parameter is posted, the corresponding value would be filled up automatically in the form under COD tab on PayU payment page
37)	shipping_state	**This parameter has to used in case of COD (Cash on Delivery) Only** Self-Explanatory (Constraints same as **state** parameter). If this parameter is posted, the corresponding value would be filled up automatically in the form under COD tab on PayU payment page
38)	shipping_country	**This parameter has to used in case of COD (Cash on Delivery) Only** Self-Explanatory (constraints same as **country** parameter). If this parameter is posted, the corresponding value would be filled up automatically in the form under COD tab on PayU payment page
39)	shipping_zipcode	**This parameter has to used in case of COD (Cash on Delivery) Only** Self-Explanatory (constraints same as **zipcode** parameter). If this parameter is posted, the corresponding value would be filled up automatically in the form under COD tab on PayU payment page

40)	shipping_phone	**This parameter has to used in case of COD (Cash on Delivery) Only** Self-Explanatory (constraints same as **phone** parameter). If this parameter is posted, the corresponding value would be filled up automatically in the form under COD tab on PayU payment page
41)	offer_key	This parameter is useful when the merchant wants to give the customer a discount offer on certain transactions based upon a pre-defined combination. This combination can be based upon payment options/bins etc. For each new offer created, a unique **offer_key** is generated. At the time of a transaction, this offer_key needs to be posted by the merchant.

For your reference, please find sample code below which shows the basic set of parameters being posted. Please execute this piece of code in browser to observe the POST request being re-directed to PayU page and then you can form the complete transaction request in your code base (with the mandatory and optional parameters)

```
<html>
<head>
</head>
<body>
<form action='https://test.payu.in/_payment' method='post'>
<input type="hidden" name="firstname" value="Vikas Kumar" />
<input type="hidden" name="lastname" value="" />
<input type="hidden" name="surl" value="https://www.google.com" />
<input type="hidden" name="phone" value="9999999999" />
<input type="hidden" name="key" value="C0Dr8m" />
<input type="hidden" name="hash" value =
"c2522a8d561e7c52f7d6b2d46c96b924afac8554313af4b80edef3e237e179bd6e2020e8c5480
60306d9fa2cf5c75c35205bcc4b09bcf5b9a9bcec8de2952d0" />
<input type="hidden" name="curl" value="http://www.google.com" />
<input type="hidden" name="furl" value="https:/www.yahoo.in" />
<input type="hidden" name="txnid" value="PLS-10061-3" />
<input type="hidden" name="productinfo" value="SAU Admission 2014" />
<input type="hidden" name="amount" value="600.000" />
<input type="hidden" name="email" value="vikaskumarsre@gmail.com" />
<input type= "submit" value="submit">
</form>
</body></html>
```

Seamless Integration – Parameters in Transaction Request

For seamless mode, 7 extra parameters are required in the transaction Post Request from your end – along with the parameters mentioned in the above table. These are mentioned below:

S No	Variable	Description
1)	pg (**Mandatory**)	This parameter is the same as the one mentioned in the POST Parameters mentioned above. It must be set as the payment category. Please set its value to '**NB'** for Net Banking , '**CC'** for Credit Card , '**DC'** for Debit Card , '**CASH'** for Cash Card and '**EMI'** for EMI
2)	bankcode (**Mandatory**)	Each payment option is identified with a unique bank code at PayU. You would need to post this parameter with the corresponding payment option's bankcode value in it. For example, for ICICI Net Banking, the value of bankcode parameter value should be ICIB. For detailed list of bank codes, please contact PayU team
3)	ccnum (**Mandatory**)	This parameter must contain the card (credit/debit) number entered by the customer for the transaction.
4)	ccname (**Mandatory**)	This parameter must contain the name on card – as entered by the customer for the transaction.
5)	ccvv (**Mandatory**)	This parameter must contain the cvv number of the card – as entered by the customer for the transaction.
6)	ccexpmon (**Mandatory**)	This parameter must contain the card's expiry month - as entered by the customer for the transaction. Please make sure that this is always in 2 digits. For months 1-9, this parameter must be appended with 0 – like 01, 02…09. For months 10-12, this parameter must not be appended – It should be 10, 11 and 12 respectively.
7)	ccexpyr (**Mandatory**)	The customer must contain the card's expiry year – as entered by the customer for the transaction. It must be of 4 digits. For example - 2017, 2029 etc.

Additional Charges – Convenience Fee Model (To be used only if recommended by Account Manager at PayU)

There are 2 different methods to implement Additional Charges on PayU.

Method 1: Enabled from backend at PayU

The merchant would be posting the **transaction amount** of the product in the transaction request.

1) Once the customer lands on PayU payment page and clicks on '**Pay Now**' option, the **additional amount** would be added to the amount of the product by PayU (based upon the TDR values) and the **total amount** would be passed on to the bank's page while re-directing.

2) After PayU receives the status of transaction from the bank, it sends the response of back to the merchant. In this response, the **amount** and **additional amount** can be differentiated with the below parameters.

   - Original Transaction Amount - **amount**
   - Additional Amount - **additionalCharges**

3) Once you receive the response from PayU, you need to check for reverse hash. If you are verifying the reverse hash at your end (which is strictly recommended to avoid any tamper cases), its formula will also change in case additionalCharges value is sent.

   Here, if the additionalCharges parameter is posted in the transaction response, then hash formula is:

   sha512**(additionalCharges|SALT|status|||||udf5|udf4|udf3|udf2|udf1|email|firstname|productinfo|amount|txnid|key)**

4) If additionalCharges parameter is not posted in the transaction response, then hash formula is the generic reverse hash formula:

   sha512(**SALT|status|||||udf5|udf4|udf3|udf2|udf1|email|firstname|productinfo|amount|txnid|key)**

Method 2: Merchant Calculates and Posts Additional Charges to PayU

1) The merchant would be posting both the transaction amount and additional charges in the transaction request. The parameters used for these are **amount** and **additional_charges** respectively. The way to pass the additional_charges parameter is as below:

<bankcode1> :< additional charge value>, < bankcode2> :< additional charge value>

Example: **CC:12,AMEX:19,SBIB:98,DINR:2,DC:25,NB:55**

2) In this method of applying additional charges, hash sequence would be affected for both Pre-Transaction and Post-Transaction.

Pre-Transaction hash sequence:

Merchant needs to form the below hash sequence before posting the transaction to PayU:

sha512**(key|txnid|amount|productinfo|firstname|email|udf1|udf2|udf3|udf4|udf5||||||SALT|additional_charges)**

Where additional_charges value would be same as the value posted in transaction request. For example, **CC:12,AMEX:19,SBIB:98,DINR:2,DC:25,NB:55**

3) Now, once the transaction request hits PayU server and re-direction happens, the customer lands upon PayU payment page. Here, depending on the payment option selection by the customer, the additional charge value would be added to transaction amount. For example, for the above example, if the customer selects Credit Card, Rs 12 would be added to the transaction amount. If the customer selects AMEX option, Rs 19 would be added to the transaction amount. For SBI Net Banking, Rs 98 would be added to the transaction amount and so on. Please note that the additional charges would be added only once the customer clicks on 'Pay Now' option.

4) When PayU receives the response from Bank, a POST Response is sent to the merchant. Here also, the hash sequence needs to be changed.

Post-Transaction hash sequence:

Merchant needs to form the below hash sequence and verify it with the hash sent by PayU in the Post Response:

sha512**(additionalCharges|SALT|status||||||udf5|udf4|udf3|udf2|udf1|email|firstname|productinfo|amount|txnid|key)**

Where, **additionalCharges** value must be same as the value Posted from PayU to the merchant in the response.

5) This hash value must be compared with the hash value posted by PayU to the merchant. If both match, then only the order should be processed. If they don't match, then the transaction has been tampered with by the user and hence should not be processed further.

Important Things to remember: Characters allowed for parameters

- For parameters address1, address2, city, state, country, product info, email, and phone following characters are allowed:
- Characters: A to Z, a to z, 0 to 9
- -(Minus)
- _ (Underscore)
- @ (At the Rate)
- / (Slash)
- (Space)
- . (Dot)

If the merchant sends any other special characters then they will be automatically removed. The address parameter will consider only first 100 characters.

Formula for hash (checksum) before transaction

This has already been covered in the description of **hash** in the table containing the POST Parameters above.

Formula for hash (checksum) after transaction

This time the variables are in reverse order and status variable is added between salt and udf1.

sha512(SALT|status|||||udf5|udf4|udf3|udf2|udf1|email|firstname|productinfo|amount|txnid|key)

It is absolutely mandatory that the hash (or checksum) is computed again after you receive response from PayU and compare it with post back parameters below. This will protect you from any tampering by the user and help in ensuring safe and secure transaction experience.

Hash (Checksum) Algorithm Example codes

The Checksum algorithm used is SHA512 which is globally well known algorithm. To need help with implementation, feel free to call us, mail us or use Google to find the desired function library for your implementation. Some example codes are also mentioned below:

For PHP

Example code:

```
$output = hash ("sha512", $text);
```

For .NET

Link: http://msdn.microsoft.com/en- us/library/system.security.cryptography.sha512.aspx

Example code:

```
byte[] data = new byte[DATA_SIZE];
byte[] result;
SHA512 shaM = new SHA512Managed();
result = shaM.ComputeHash(data);
```

For JSP

Example code:

```
import java.io.FileInputStream;
import java.security.MessageDigest;
public class SHACheckSumExample
{
        public static void main(String[] args)throws Exception
        {
                MessageDigest md = MessageDigest.getInstance("SHA-512");
                FileInputStream fis = new FileInputStream("c:\\loging.log");
                byte[] dataBytes = new byte[1024];
                int nread = 0;
                while ((nread = fis.read(dataBytes)) != -1)
                {
                        md.update(dataBytes, 0, nread);
                };
                byte[] mdbytes = md.digest();
                //convert the byte to hex format method
                StringBuffer sb = new StringBuffer();
                for (int i = 0; i < mdbytes.length; i++)
                {
                        sb.append(Integer.toString((mdbytes[i] & 0xff) + 0x100,
                        16).substring(1));
                }
                System.out.println("Hex format : " + sb.toString());
                //convert the byte to hex format method 2
                StringBuffer hexString = new StringBuffer();
                for (int i=0;i<mdbytes.length;i++)
                hexString.append(Integer.toHexString(0xFF & mdbytes[i]));
        }
        System.out.println("Hex format : " + hexString.toString());
}
```

Response Parameters posted by PayU to Merchant

<table>
<tr><th>Sr.No</th><th>Variable Name</th><th>Description</th></tr>
<tr><td>1</td><td>mihpayid</td><td>It is a unique reference number created for each transaction at PayU's end. For every new transaction request that hits PayU's server (coming from any of our merchants), a unique reference ID is created and it is known as mihpayid (or PayU ID)</td></tr>
<tr><td>2</td><td>mode</td><td>This parameter describes the payment category by which the transaction was completed/attempted by the customer. The values are mentioned below:
<table>
<tr><th>Category used by Customer</th><th>Value of Mode Parameter</th></tr>
<tr><td>Credit Card</td><td>CC</td></tr>
<tr><td>Debit Card</td><td>DC</td></tr>
<tr><td>NetBanking</td><td>NB</td></tr>
<tr><td>Cash Card</td><td>CASH</td></tr>
<tr><td>EMI</td><td>EMI</td></tr>
<tr><td>IVR</td><td>IVR</td></tr>
<tr><td>Cash On Delivery</td><td>COD</td></tr>
</table></td></tr>
<tr><td>3</td><td>status</td><td>This parameter gives the status of the transaction. Hence, the value of this parameter depends on whether the transaction was successful or not. You must map the order status using this parameter only. The values are as below:

If the transaction is successful, the value of 'status' parameter would be 'success'.

The value of 'status' as 'failure' or 'pending' must be treated as a failed transaction only.</td></tr>
<tr><td>4</td><td>key</td><td>This parameter would contain the merchant key for the merchant's account at PayU. It would be the same as the key used while the transaction request is being posted from merchant's end to PayU.</td></tr>
<tr><td>5</td><td>txnid</td><td>This parameter would contain the transaction ID value posted by the merchant during the transaction request.</td></tr>
<tr><td>6</td><td>amount</td><td>This parameter would contain the original amount which was sent in the transaction request by the merchant.</td></tr>
</table>

7	discount	This parameter would contain the discount given to user - based on the type of offer applied by the merchant.
8	offer	This parameter would contain the offer key which was sent in the transaction request by the merchant.
9	productinfo	This parameter would contain the same value of **productinfo** which was sent in the transaction request from merchant's end to PayU
10	firstname	This parameter would contain the same value of **firstname** which was sent in the transaction request from merchant's end to PayU
11	lastname	This parameter would contain the same value of **lastname** which was sent in the transaction request from merchant's end to PayU
12	address1	This parameter would contain the same value of **address1** which was sent in the transaction request from merchant's end to PayU
13	address2	This parameter would contain the same value of **address2** which was sent in the transaction request from merchant's end to PayU
14	city	This parameter would contain the same value of **city** which was sent in the transaction request from merchant's end to PayU
15	state	This parameter would contain the same value of **state** which was sent in the transaction request from merchant's end to PayU
16	country	This parameter would contain the same value of **country** which was sent in the transaction request from merchant's end to PayU
17	zipcode	This parameter would contain the same value of **zipcode** which was sent in the transaction request from merchant's end to PayU
18	email	This parameter would contain the same value of **email** which was sent in the transaction request from merchant's end to PayU
19	phone	This parameter would contain the same value of **phone** which was sent in the transaction request from merchant's end to PayU
20	udf1	This parameter would contain the same value of **udf1** which was sent in the transaction request from merchant's end to PayU
21	udf2	This parameter would contain the same value of **udf2** which was sent in the transaction request from merchant's end to PayU
22	udf3	This parameter would contain the same value of **udf3** which was sent in the transaction request from merchant's end to PayU
23	udf4	This parameter would contain the same value of **udf4** which was sent in the transaction request from merchant's end to PayU
24	udf5	This parameter would contain the same value of **udf5** which was sent in the transaction request from merchant's end to PayU

25	Hash	This parameter is absolutely crucial and is similar to the hash parameter used in the transaction request send by the merchant to PayU. PayU calculates the hash using a string of other parameters and returns to the merchant. The merchant must verify the hash and then only mark a transaction as success/failure. This is to make sure that the transaction hasn't been tampered with. The calculation is as below: **sha512(SALT|status|||||udf5|udf4|udf3|udf2|udf1|email|first name|productinfo|amount|txnid|key)** The handling of udf1 – udf5 parameters remains similar to the hash calculation when the merchant sends it in the transaction request to PayU. If any of the udf (udf1-udf5) was posted in the transaction request, it must be taken in hash calculation also. If none of the udf parameters were posted in the transaction request, they should be left empty in the hash calculation too.
26	Error	For the failed transactions, this parameter provides the reason of failure. Please note that the reason of failure depends upon the error codes provided by different banks and hence the detailing of error reason may differ from one transaction to another. The merchant can use this parameter to retrieve the reason of failure for a particular transaction.
27	bankcode	This parameter would contain the code indicating the payment option used for the transaction. For example, in Debit Card mode, there are different options like Visa Debit Card, Mastercard, Maestro etc. For each option, a unique bankcode exists. It would be returned in this bankcode parameter. For example, Visa Debit Card – **VISA,** Master Debit Card – **MAST.**
28	PG_TYPE	This parameter gives information on the payment gateway used for the transaction. For example, if SBI PG was used, it would contain the value **SBIPG.** If SBI Netbanking was used for the transaction, the value of PG_TYPE would be **SBINB**. Similarly, it would have a unique value for all different type of payment gateways.
29	bank_ref_num	For each **successful** transaction – this parameter would contain the **bank reference number** generated by the bank.
30	shipping_firstname	This parameter would contain the same value of **shipping_firstname** which was sent in the transaction request from merchant's end to PayU

31	shipping_lastname	This parameter would contain the same value of **shipping_lastname** which was sent in the transaction request from merchant's end to PayU
32	shipping_address1	This parameter would contain the same value of **shipping_address1** which was sent in the transaction request from merchant's end to PayU
33	shipping_address2	This parameter would contain the same value of **shipping_address2** which was sent in the transaction request from merchant's end to PayU
34	shipping_city	This parameter would contain the same value of **shipping_city** which was sent in the transaction request from merchant's end to PayU
35	shipping_state	This parameter would contain the same value of **shipping_state** which was sent in the transaction request from merchant's end to PayU
36	shipping_country	This parameter would contain the same value of **shipping_country** which was sent in the transaction request from merchant's end to PayU
37	shipping_zipcode	This parameter would contain the same value of **shipping_zipcode** which was sent in the transaction request from merchant's end to PayU
38	shipping_phone	This parameter would contain the same value of **shipping_phone** which was sent in the transaction request from merchant's end to PayU
39	unmappedstatus	This parameter contains the status of a transaction as per the internal database of PayU. PayU's system has several intermediate status which are used for tracking various activities internal to the system. Hence, this status contains intermediate states of a transaction also - and hence is known as **unmappedstatus**. For example: dropped/bounced/captured/auth/failed/usercancelled/pending

Shopping Cart Integration Kits

Shopping Cart Kits currently available with PayU are:

- Interspire
- Opencart
- Jhoomla Virtue Mart
- Magento
- Prestashop
- Tomatocart
- Zencart
- CS-Cart
- OSCommerce
- Wordpress ecommerce
- WordPress Woo-commerce
- Wordpress - Paid Membership Pro
- Drupal Ubercart
- X-Cart

Platform based Integration kits

PayU Integration Kits are available in the following environments:

- PHP
- JSP
- .NET
- ROR

NOTE: *Kindly contact your account manager in case you are using some other shopping cart and want us to develop a kit for the same.*

NOTE: In case of any integration queries, please drop a mail at tech@payu.in

SECTION II: WEB SERVICES – APIs

PayU has made many web-services for you. Each web-service has a specific function and hence can be used to automate different features. The basic format and execution of all web-services remains the same. Each web-service is a server-to-server call from your server to PayU's server.

Web services can be accessed by making a **server to server call** on the below mentioned PayU URLs:

URL to be used:

For Production Server:

https://info.payu.in/merchant/postservice.php?form=1
(form=1 shall return output in array form)

https://info.payu.in/merchant/postservice.php?form=2
(form=2 shall return output in json form)

For Test Server:

https://test.payu.in/merchant/postservice.php?form=1
(form=1 shall returns output in array form)

https://test.payu.in/merchant/postservice.php?form=2
(form=2 shall return output in json form)

Web Service Request Format:

The input request format for executing a web-service is as follows:

Mandatory Input Parameters

Parameter	Description	Sample Value
key	Merchant key provided by PayU. Please refer to the first entry in the Post Parameters table for detailed description of this parameter	Ibibo
command	This parameter must have **name of the web-service.** The names and definitions of all web-services will be covered later in detail	verify_payment
hash	This parameter **must** contain the hash value to be calculated at your end. The string used for calculating the hash is mentioned below: **sha512(key|command|var1|salt)** sha512 is the encryption method used here.	ajh84ba8abvav

var1, var2, var3 ... up to var15	These are the variable parameters, whose values depend on the particular web-service. The definition of these parameters will be covered in the (Read command explanations mentioned later for this)	Abc

Web Service Response Format

Web Service API responds back in PHP serialized string by default.

Parameter	Description	Sample Value
status	Status of web service call	0 if web service call failed 1 if web service call succeeded
msg	Reason String	Parameter missing or token is empty or amount is empty or transaction not exists
transaction_details	May or may not be returned depending on the web service being called	mihpayid,request_id, bank_ref_num etc
request_id	PayU Request ID for a request in a Transaction. eg. A transaction can have a refund request.	7800456
bank_ref_num	Bank Reference Number. If bank provides after a successful action.	204519474956

LIST OF APIs AND THEIR DESCRIPTION

1) verify_payment

This web-service is used to reconcile the transaction with PayU. When we **post back the final response** to you (merchant), we provide a list of parameters (including the status of the transaction – For example, **success, failed** etc). On a few occasions, the transaction response is initiated from our end, but it doesn't reach you due to network issues or user activity (like refreshing the browser etc).

This API is helpful to tackle such cases - where you can execute it to get the status of the transaction. Since you already have the **txnID (Order ID generated at your end)** value for such cases, you simply need to execute the verify_payment API with the necessary input parameters. The output would return you the transaction status and various other parameters also.

Another usage of this API is to provide an additional layer of verification of the transaction (in addition to checksum). You can verify the status and other parameters received in the post response via this API.

We strongly recommend that this API is used to reconcile with PayU's database once you receive the response. This will protect you from any tampering by the user and help in ensuring safe and secure transaction experience.

The return parameters are MIHPayID, Amount, Discount, Mode and Status of transaction.

Input Variables Description:

Parameter	Description	Sample Value
var1	In this parameter, you can put all the txnid(Your transaction ID/order ID) values in a pipe separated form.	100123|100124|100125|100126

Web Service Responses:

- **if Merchant transaction ID is missing**
  array('status' => 0, 'msg' => 'Parameter missing')

- **if Merchant transaction ID isn't found**
  array('status' => '1', 'msg' => 'Transaction Fetched Successfully',
  'transaction_details' => array('mihpayid' => 'Not Found', 'status' => 'Not Found'));

- **if successfully fetched**

```
array('status' => '1',
        'msg' => 'Transaction Fetched Successfully',
        'transaction_details' =>array('mihpayid' => Transaction ID,
                                      'request_id' => Request ID,
                                      'bank_ref_num' => Bank Reference Number,
                                      'amt' => Amount
                                      'disc' => Discount
                                      'mode' => Transaction Mode (NB for Net banking, CC
                                      for credit card, DC for Debit card, "-" for unknown)
                                      'status' => Transaction Status
                                )
                        );
```

2) check_payment

This API functions similar to verify_payment API mentioned above. The only difference is that the input parameter in this API is the PayUID (MihpayID) generated at PayU's end whereas the input parameter in verify_payment API is the TxnID (Transaction ID generated at your end). It returns all the parameters for a given transaction.

Input Variables Description:

Parameter	Description	Sample Value
var1	In this parameter, you need to pass the Payu id (mihpayid) of the transaction.	8000123

Web Service Responses:

- **if mihpayid is missing**

  array('status' => 0, 'msg' => 'Parameter missing')

- **if mihpayid isn't found**

  array('status' => '1', 'msg' => 'Transaction Fetched Successfully',
  'transaction_details' => array('mihpayid'=> 'Not Found', 'status' => 'Not Found')
  );

- **if successfully fetched**

  array('status' => '1', 'msg' => 'Transaction Fetched Successfully',
  'transaction_details' =>array(
  'mihpayid' => Transaction ID,
  'request_id' => Request ID,
  'bank_ref_num' => Bank Reference Number,
  'amt' => Amount
  'disc' => Discount
  'mode' => Transaction Mode (NB for Net banking, CC for credit card, DC for Debit card, "-" for unknown)
  'status' => Transaction Status
  'txnid' => Request ID
  'amount' => Transaction Fees Amount
  'amount_paid' => Transaction Amount
  'discount' => Transaction Discount
  'net_amount'=>Amount received by merchant after all deductions
  .... Some more fields
)
  );

3) cancel_refund_transaction

This command can be used for 2 different purposes:

- To cancel a transaction which is in '**auth'** state at the moment
- To refund a transaction which is in '**captured'** state at the moment

Input Variables Description:

Parameter	Description	Sample Value
var1	Payu ID (mihpayid) of transaction	8000123
var2	This parameter should contain the Token ID (unique token from merchant) for the refund request. Token ID has to be generated at your end for each new refund request. It is an identifier for each new refund request which can be used for tracking it. It must be unique for every new refund request generated – otherwise the refund request would not be generated successfully.	7800456

var3	This parameter should contain the amount which needs to be refunded. Please note that both partial and full refunds are allowed. Hence, for partial refund, this var3 value would be less than the amount with which the transaction was made. For full refund, var3 value would be equal to the amount with which the transaction was made.	500

Web Service Responses:

- **if token is missing**
  array('status' => 0, 'msg' => 'token is empty');

- **if amount is missing**
  array('status' => 0, 'msg' => 'amount is empty')

- **if transaction isn't found**
  array('status' => 0, 'msg' => 'transaction not exists');

- **on successful processing at our end**
  array('status' => '1', 'msg' => Cancel Request Queued',
  'txn_update_id' => Request ID, 'bank_ref_num' => Bank Reference Number,' mihpayid'=> PayU Transaction id);

- **on successful processing on our end for captured transactions**
  array('status' => '1', 'msg' => Refund Request Queued', 'request_id' => Request ID,
  'bank_ref_num' => Bank Reference Number, 'mihpayid'=> PayU Transaction id);

- **if failed to refund**
  array('status' => 0, 'msg' => Refund request failed');

- **if capture is done on the same day**
  array('status' => '1', 'msg' => ' Capture is done today, please check for refund status tomorrow ', 'request_id' => Request ID, 'bank_ref_num' => Bank Reference Number, 'mihpayid' => PayU ID);

- **if invalid token**
  array('status' => 0, 'msg' => 'token already used or request pending.');

- **on successful processing at PayU end for** auth **transactions**
  array('status' => '1', 'msg' => ' Cancel Request Queued', 'txn_update_id' => Request ID, 'bank_ref_num' => Bank Reference Number);

- **if failed to cancel a transaction**
  array('status' => 0, 'msg' => 'Cancel request failed');

4) check_action_status

This API is used to check the status of refund/cancel requests. Whenever the cancel_refund_transaction API is executed successfully, a **Request ID** is returned in the output parameters for that particular request. In check_action_status API, you need to input this Request ID to get the current status of the request. The return parameters are MIHPayID, Amount, Discount, Mode and Status of transaction.

Input Variables Description:

Parameter	Description	Sample Value
var1	request_id	7800456

Web Service Responses:

- **if mihpayid is missing**

  array('status' => 0, 'msg' => 'Parameter missing')

- **if mihpayid isn't found**

```
array('status' => '1', 'msg' => 'Transaction Fetched Successfully',
        'transaction_details' => array(
                search ID => 'No action status found'
            )
        )
 );
```

- **if successfully fetched**

```
array('status' => '1', 'msg' => 'Transaction Fetched Successfully',
        'transaction_details' =>array(Search ID => array (
                                        Request ID => array (
                                                'mihpayid' => Transaction ID,
                                                'bank_ref_num' => Bank Reference Number,
                                                'request_id' => Request ID,
                                                'amt' => Amount
                                                'disc' => Discount
                                                 'mode' => Transaction Mode
                    (NB for Netbanking, CC for credit card, DC for Debit card, "-" for unknown)
                                                'action' => Requested Action
                                            'status' => Request Status
                                    )
                                )
                            )
        );
```

5) capture_transaction

This command is used to update the status of a transaction which is in auth (authorized) state at the moment. Please note that this API is applicable only for transactions in 'auth' status and nothing else. 'auth' and 'capture' model is followed for transactions routed through CITI PG. So, if a transaction through CITI PG is successful, it will be marked as 'auth' at that instant. The merchant has an option of capturing or cancelling this transaction till end of that day. This API can be executed

to capture the transaction. For cancelling, please refer to cancel_refund_transaction API mentioned above. Please note that we automatically capture all the 'auth' transactions at end of day (if you haven't already 'cancelled' or 'captured' it). Also, this auth and capture model is strictly applicable for CITI PG only. The return parameters are: request_id, bank_ref_num.

Input Variables Description:

Parameter	Description	Sample Value
var1	Payu ID (mihpayid) of transaction	8000123
var2	token ID(unique token from merchant)	7800456

Web Service Responses:

- **if token is missing**
  array('status' => 0, 'msg' => 'token is empty');

- **if transaction isn't found**
  array('status' => 0, 'msg' => 'transaction not exists');

- **on successful processing at our end**
  array('status' => '1', 'msg' => ' Capture Request Queued', 'request_id' => Request ID, 'bank_ref_num' => Bank Reference Number);

- **if invalid token**
  array('status' => 0, 'msg' => 'token already used or request pending.');

- **if failed to refund**
  array('status' => 0, 'msg' => 'Capture request failed');

6) update_requests

This command is used to update a requested refund, cancel, or capture transaction. The return parameters are status and msg. For example, in case of COD transaction, if a refund is initiated its status goes to **'requested'** state. Once the refund is done, then its status can be changed to **'refund'** by calling this API.

Input Variables Description:

Parameter	Description	Sample Value
var1	Payu id (mihpayid) of transaction	8000123
var2	Request ID (unique id given to merchant) provided when cancel_transaction or refund_transaction or capture_transaction was called)	7800456
var3	Bank Ref Id for the requested transaction	Abc123

var4	Amount of the requested transaction	5000
var5	Action (cancel/capture/refund)	refund
var6	New Status to be set	Success/failure

Web Service Responses:

- **If bank_ref_no is missing**
  array('status' => 0, 'msg' => 'bank_ref_no is empty');
- **if amount is missing**
  array('status' => 0, 'msg' => 'amount is empty')
- **if transaction isn't found**
  array('status' => 0, 'msg' => 'transaction not exists');
- **if action is not valid**
  array('status' => 0, 'msg' => 'action is not valid');
- **if status is not correct**
  array('status' => 0, 'msg' => status is not correct');
- **On success**
  array('status' => 1, 'msg' => 'Status updated to success.');
- **On failure**
  array('status' => 0, 'msg' => 'Status could not be updated. Please verify the parameters.);

7) cod_verify

This command is used to **verify** a COD request. When a transaction is successful through PayU, it is marked as **'in progress'** at that moment. The reason is that the money hasn't been received yet and hence we mark it in this intermediary state. Once you verify the transaction with the customer, you can execute this API to update the status in PayU Database from '**in progress'** to '**pending'**. The return parameters are status, message and transaction ID.

Input Variables Description:

Parameter	Description	Sample Value
var1	Payu ID (mihpayid) of transaction	8000123
var2	Token ID(unique token from merchant)	7800456
var3	Amount	500

Web Service Responses:

- **if token is missing**
  array('status' => 0, 'msg' => 'token is empty');

- **if amount is missing**
  array('status' => 0, 'msg' => 'amount is empty')

- **if amount is invalid**
  array('status' => 0, 'msg' => 'Invalid amount')

- **if transaction isn't found**
  array('status' => 0, 'msg' => 'transaction not exists');

- **on successful processing at PayU end**
  array('status' => '1', 'msg' => 'Queued', transaction_id' => $mihpayid);

- **if failed to verify a request**
  array('status' => 0, 'msg' => 'Failed', 'error_code' => $verifyReturn['status']);

8) cod_cancel

This command is used to **cancel** a cod request. When a COD transaction is successful at PayU's end in real time, its status is marked as **'in progress'** at that moment. This API can be executed to change the transaction status from **'in progress'** to **'cancelled'** in the PayU database. It is suggested to execute this API only when you are sure you want to cancel the transaction. Updating this way in PayU Database would help you in tracking such orders for future purpose – through the merchant panel provided to you. The return parameters are status message and transaction ID.

Additional Variables Description:

Parameter	Description	Sample Value
var1	Payu ID (mihpayid) of transaction	8000123
var2	Token ID(unique token from merchant)	7800456
var3	Amount	500

Web Service Responses:

- **if token is missing**
  array('status' => 0, 'msg' => 'token is empty');

- **if amount is missing**
  array('status' => 0, 'msg' => 'amount is empty')

- **if amount is invalid**
  array('status' => 0, 'msg' => 'Invalid amount')

- **if transaction isn't found**
  array('status' => 0, 'msg' => 'transaction not exists');

- **on successful processing at PayU end**
  array('status' => '1', 'msg' => 'Queued', transaction_id' => $mihpayid);

- **if failed to cancel a request**
  array('status' => 0, 'msg' => 'Failed', 'error_code' => $cancelReturn['status']);

9) cod_settled

This command is used to **settle** a COD request. cod_settled API should be executed on a transaction only when cod_verify has already been executed. cod_settled updates the transaction status from 'pending' to 'captured'. It is suggested, that you execute this API only when you are sure that money has been successfully received from the customer at your end. Doing it this way would ensure you can track such orders in the future through the merchant panel provided to you. The return parameters are status message and Transaction ID.

Input Variables Description:

Parameter	Description	Sample Value
var1	Payu id (mihpayid) of transaction	8000123
var2	token ID(unique token from merchant)	7800456
var3	amount	500

Web Service Responses:

- **if token is missing**
  array('status' => 0, 'msg' => 'token is empty');

- **if amount is missing**
  array('status' => 0, 'msg' => 'amount is empty')

- **if amount is invalid**
  array('status' => 0, 'msg' => 'Invalid amount')

- **if transaction isn't found**
  array('status' => 0, 'msg' => 'transaction not exists');

- **on successful processing at PayU end**
  array('status' => '1', 'msg' => 'Queued', transaction_id' => $mihpayid);

- **if failed to settled a request**
  array('status' => 0, 'msg' => 'Failed', 'error_code' => $settledReturn['status']);

10) get_TDR:

This command is used to get the TDR value of a transaction with PayU. It is a simple API for which you need to provide the PayU ID of the transaction as input and the TDR value is returned in the output.

Input Variables Description:

Parameter	Description	Sample Value
var1	Payu id (mihpayid) of transaction	8000123

Web Service Responses

- **if mihpayid is missing**

  array('status' => 0, 'msg' => Invalid PayU Id)

- **if successfully fetched**

```
array('status' =>1', 'msg' => 'Transaction Fetched Successfully',
                'transaction_details' =>array(
                                                TDR=> Value
                                                )
                                        );
```

11) udf_update

This command is used to update the UDF1-UDF5 values of transaction with PayU. UDFs are the user-defined fields which are posted from the merchant to PayU. This API is specifically used to update the values in these fields in PayU Database. The return parameters are the **updated UDF** values of transaction.

Input Variables Description:

Parameter	Description	Sample Value
var1	transaction ID(txnid)	80001abcd
var2	udf1 of transaction	8000123
var3	udf2 of transaction	4334343
var4	udf3 of transaction	434343
var5	udf4 of transaction	Abcd123
var6	udf5 of transaction	Efgh1234

Web Service Responses

- **if mihpayid is missing**

array('status' => 0, 'msg' => 'Parameter missing')

- **if transaction id is missing**
  array('status' => 0, 'msg' => Invalid TXN Id)

- **if successfully fetched**
  array([status] => UDF values updated
  [transaction_id] => 80001abcd
  [udf1] => 8000123
  [udf2] => 4334343
  [udf3] => 434343
  [udf4] => Abcd123
  [udf5] => Efgh1234
)

12) create_invoice

This API is provided to the merchant to create an email invoice for a customer and gives the merchant an option of sending the email invoice immediately to the customer or it can be automated to be sent later.

Input Variables Description:

Parameter	Sample Value
var1	{"**amount**":"10","**txnid**":"abc3332","**productinfo**":"jnvjrenv","**firstname**":"test","**email**":"test@test.com","**phone**":"1234567890","**address1**":"testaddress","**city**":"test","**state**":"test","**country**":"test","**zipcode**":"122002","**template_id**":"14","**validation_period**":6,"**send_email_now**":"1"}

Here, the input var1 parameter has to be generated in the json string format mentioned in the sample value string above. This string shows each parameter and its corresponding value separated by the delimiter colon (:). The parameters are also separated by the comma delimiter (,)

Following is the description of the parameters in the above mentioned string:

Parameter	Description
amount **(Mandatory)**	Payment Amount
txnid **(Mandatory)**	Merchant generated transaction number which is used to track a particular order. (Must be unique every time if already successful, otherwise you get an error of duplicate transaction)
productinfo **(Mandatory)**	Product Description
firstname **(Mandatory)**	Self-Explanatory (only alphabets a-z are allowed)

email **(Mandatory)**	Self-explanatory
phone **(Mandatory)**	Self-explanatory (Numeric Value only)
address1	Self-Explanatory (Length of Address1 must not be more than 100 characters and the allowed characters are only) A TO Z, a to z, 0 to 9, @, - (Minus), _ (Underscore), / (Backslash), (Space), (Dot)
city	Self-explanatory (allowed characters are same as in address1)
State	Self-explanatory (allowed characters are same as in address1)
country	Self-explanatory (allowed characters are same as in address1)
zipcode	Self-explanatory (numeric value only)
template_id	Template ID to be provided in case of more than one email invoice templates. Merchant can decide which template to use and provide that particular template ID in this parameter
validation_period	Number of days for which the email invoice usage is valid (If this field is left empty, then default value will be taken as 7 days)
send_email_now	1 - If the merchant wants to automatically send the email invoice request to the customer at the time of creation of email invoice itself 0 - If the merchant doesn't want to send the email invoice request to the customer at the creation time itself. In this case, the email would be sent later automatically

Web Service Responses

- **if successfully executed**

array
(
[Transaction Id] => abfcee3332
[Email Id] => test.example@ibibogroup.com
[Phone] => 1234567890
[Status] => Success
*[URL]=>*https://secure.payu.in/processInvoice?invoiceId=8d495559881c4011e2a882*d3e1d0f1e37081c387837f77d5b651172d5cc00b80*
)

- **if duplicate transaction id is used**

Invoice for this transaction ID already exists.

- **If invalid parameter is sent***

Invalid `<parameter>`

Note: Here <parameter> value displayed would be the incorrect parameter provided*

13) check_offer_status (1st Usage)

This API is used to check the status of an offer for a particular merchant when all the details are passed. The return parameters are status, msg, discount/error_code, category, offer_key, offer_type(instant/ cashback) , offer_availed_count, offer_remaining_count.

Input Variables Description:

Parameter	Description	Sample Value
var1	Offer Key(mandatory)	offer@123
var2	Amount	100
var3	Category	CC
var4	Bank Code	CC
var5	Card Number(mandatory)	5432112345678901
var6	Name on Card	Nitesh
var7	Phone Number	91234567890
var8	Email Id	abc@xyz.com

Error Codes:

- 'INVALID_OFFER'=>'E001',
- 'INVALID_PAYMENT_METHOD'=>'E002'

In the Output:

- Parameter 'status' = 1, means offer is valid
- Parameter 'status' = 0, means offer is invalid.

Web Service Responses:

Note: In the response, category will be the passed Category.

- **If the offer is a valid offer**

```
array('status'=>1,
        'msg'=>'Valid offer',
        'discount'=> 5,
        'category'=> creditcard,
        'offer_key'=>offerKey,
        'offer_type' => 'Instant',
        'offer_availed_count' => 5,
        'offer_remaining_count' => 10
      )
```

- **If the offer is expired**

```
array('status'=>0,
        'msg'=> 'Offer Expired',
        'error_code'=>E001,
        'category'=> creditcard,
        'offer_key'=>offerKey,
        'offer_type' => 'Unknown',
        'offer_availed_count' => 'Unknown',
        'offer_remaining_count' => 'Unknown'
      )
```

- **If the card limit is exhausted**

```
array('status'=>0,
        'msg'=> 'Offer Exhausted',
        'error_code'=>E001,
        'category'=> creditcard,
        'offer_key'=>offerKey,
        'offer_type' => 'Unknown',
        'offer_availed_count' => 'Unknown',
        'offer_remaining_count' => 'Unknown'
 )
```

- **if offerKey is invalid**

```
array('status'=>0,
        'msg'=>'Invalid offer',
        'error_code'=>E001,
        'category'=> creditcard,
        'offer_key'=>offerKey,
        'offer_type' => 'Unknown',
        'offer_availed_count' => 'Unknown',
        'offer_remaining_count' => 'Unknown'
 )
```

14) check_offer_status (2nd Usage)

This API is used to check the status of an offer when only the parameters Offer Key and card number are passed as input. This API can be used to check the offer status when offer is created using bin only. In this case we can depict that the offer has been created for which category (like CC/DC/NB/EMI). Hence, for using this API, you need to pass the Offer Key and Card Number in var1 and var5 field as inputs and leave the rest field empty.

The return parameters are status, msg, error_code (In case of error), category, offer_key, offer_type (instant/cashback), offer_availed_count, 'offer_remaining_count'.

Input Variables Description:

Parameter	Description	Sample Value
var1	Offer Key(mandatory)	offer@123

var2	Empty	-
var3	Empty	-
var4	Empty	-
var5	Card Number(mandatory)	5432112345678901

Error Codes:

- 'INVALID_OFFER'=>'E001',
- 'INVALID_PAYMENT_METHOD'=>'E002'

Output:

- Parameter 'Status' = 1, means offer is valid
- Parameter 'Status' = 0, means offer is invalid

Web Service Responses:

- **If the offer is a valid offer for the given card number(bin)**

```
array('status'=>1,
    'msg'=>Valid offer,
     'category'=> creditcard,
    'offer_key'=>abc@123,
    'offer_type' => instant,
    'offer_availed_count' => 5,
    'offer_remaining_count' => 10
  )
```

- **If the offer is expired**

```
array('status'=>0,
    'msg'=> 'Offer Expired',
    'error_code'=>E001,
     'category'=> Unknown
    'offer_key'=>offerKey,
    'offer_type' => 'Unknown',
    'offer_availed_count' => 'Unknown',
    'offer_remaining_count' => 'Unknown'
  )
```

- **If the card limit is exhausted**

```
array('status'=>0,
    'msg'=> 'Offer Exhausted',
    'error_code'=>E001,
     'category'=> Unknown
    'offer_key'=>offerKey,
    'offer_type' => 'Unknown',
    'offer_availed_count' => 'Unknown',
    'offer_remaining_count' => 'Unknown'
  )
```

- **If the offer is a invalid offer for the given card number(bin)**

```
array('status'=>0,
        'msg'=>Invalid offer,
         'error_code'=> E001/E002,
        'offer_key'=>abc@123,
        'offer_type' => Unknown,
        'offer_availed_count' => Unknown,
        'offer_remaining_count' => Unknown
       )
```

15) getNetbankingStatus

This API is used to help you in handling the NetBanking Downtime. A few times, one or more Net Banking options may be facing downtime due to issues observed at Bank's end. This API is used to tell the status of one or all the net banking options. The status can be either up or down. If you want to know the status of a specific Net Banking option, the input parameter should contain the corresponding ibibo_code. If you want to know the status of all the Net Banking options, the input parameter should contain the value 'default'.

Input variable description:

Parameter	Description	Sample Value
var1	ibibo_code or "default"	AXIB, "default"

Web Service Responses:

Case a: To get status of one Net Banking Option **(The specific ibibo_code is passed in input)**

Response**:**

```
array
(
   [AXIB] => array
     (
        [ibibo_code] => AXIB
        [title] => AXIS Bank NetBanking
        [up_status] => 0
     )
)
```

Note:

- up_status = 0 signifies that the particular Bank option is **down** at the moment.
- up_status=1 signifies that the particular Bank Banking option is **up** at the moment.

Case b: To get status of all Net Banking options. (**The value "default" is passed in input)**

Web Service Responses:

```
array
(
   [AXIB] => array
      (
         [ibibo_code] => AXIB
         [title] => AXIS Bank NetBanking
         [up_status] => 1
      )

   [BOIB] => array
      (
         [ibibo_code] => BOIB
         [title] => Bank of India
         [up_status] => 1
      )

   [BOMB] => array
      (
         [ibibo_code] => BOMB
         [title] => Bank of Maharashtra
         [up_status] => 1
      )

   [CABB] => array
      (
         [ibibo_code] => CABB
         [title] => Canara Bank
         [up_status] => 1
      )
.
.
.
.<All the other banks and their status>
```

Note:

- up_status = 0 signifies that the particular Bank option is **down** at the moment.
- up_status= 1 signifies that the particular Bank Banking option is **up** at the moment.

16) getIssuingBankStatus

This API is used to help you in handling the Credit Card/Debit Card Issuing Bank Downtime. It allows you get the present status of an Issuing Bank using the specific Bank Identification Number (BIN). BIN is identified as the first 6 digits of a credit/debit card. You need to provide the bin number as input and the corresponding issuing bank's status would be returned in the output (whether up or down).

Input variable description:

Parameter	Description	Sample Value
var1	Bank Identification Number(First 6 digits of a card)	512345

Web Service Responses:

```
array
(
   [issuing_bank] => HDFC
   [up_status] => 1
)
```

Note:

- up_status = 0 signifies that the particular Bank option is **down** at the moment.
- up_status= 1 signifies that the particular Bank Banking option is **up** at the moment.

17) get_Transaction_Details

This API is used to extract the transaction details between two given time periods. The API takes the input as two dates (initial and final), between which the transaction details are needed. The output would consist of the status of the API (success or failed) and all the transaction details in an array format.

Input variable description:

Parameter	Description	Sample Value
var1	Starting Date (From when the transaction details are needed) in yyyy-mm-dd format	2014-01-12
var2	Starting Date (Till when the transaction details are needed) in yyyy-mm-dd format	2014-01-13

Web Service Responses:

The status variable would be 1 for successful web-service execution and would be 0 in case of unsuccessful web-service execution. Output would be returned in the following array format:

a) For Successful Response, status=1:

```
array
   (
      [status] => 1
          [msg] => Transaction Fetched Successfully
          [Transaction_details] => Array
      (
        [0] => array
         (
```

```
    [id] => 403993715508970248
    [status] => failed
    [key] => C0Dr8m
    [merchantname] => test payu
    [txnid] => e1e8a8f4ace8506043e1
    [firstname] => John
    [lastname] => Moses
    [addedon] => 2014-02-04 01:25:38
    [bank_name] => Visa Debit Cards (All Banks)
    [payment_gateway] => AXIS
    [phone] => 9585475883
    [email] => y.johnmoses@gmail.com
    [amount] => 100.00
    [discount] => 0.00
    [additional_charges] => 0.00
    [productinfo] => CSIIT Conference Registration
    [error_code] => E312
    [bank_ref_no] => 2000112693
    [ibibo_code] => VISA
    [mode] => DC
    [ip] => 117.206.82.90
    [card_no] => 414367XXXXXX0250
    [cardtype] => international
    [offer_key] =>
    [field2] => 403506432293
    [udf1] =>
    [pg_mid] => TESTIBIBOWEB
    [offer_type] =>
    [failure_reason] =>
    [mer_service_fee] =>
    [mer_service_tax] =>
  )

[1] => Array
  (
    [id] => 403993715508970268
    [status] => captured
    [key] => C0Dr8m
    [merchantname] => test payu
    [txnid] => 8613914632655135
    [firstname] => Hans Wurst
    [lastname] =>
    [addedon] => 2014-02-04 03:03:06
    [bank_name] => Credit Card
    [payment_gateway] => HDFC
    [phone] =>
    [email] => f606f938f64b499aa3fd952d6338aa54@example.com
    [amount] => 30.00
    [discount] => 0.00
    [additional_charges] => 0.00
```

```
        [productinfo] => 3752946
        [error_code] => E000
        [bank_ref_no] => 1953525040340351
        [ibibo_code] => CC
        [mode] => CC
        [ip] => 217.6.59.133
        [card_no] => 512345XXXXXX2346
        [cardtype] => domestic
        [offer_key] =>
        [field2] => 999999
        [udf1] =>
        [pg_mid] => 90000970
        [offer_type] =>
        [failure_reason] =>
        [mer_service_fee] => 0.70
        [mer_service_tax] => 0.09
    )

[2] => Array
    (
        [id] => 403993715508970270
        [status] => captured
        [key] => C0Dr8m
        [merchantname] => test payu
        [txnid] => 8813914632908201
        [firstname] => Hans Wurst
        [lastname] =>
        [addedon] => 2014-02-04 03:03:30
        [bank_name] => Credit Card
        [payment_gateway] => HDFC
        [phone] =>
        [email] => 89163cd22823449d89e6d5cd2346fea3@example.com
        [amount] => 30.00
        [discount] => 0.00
        [additional_charges] => 0.00
        [productinfo] => P172
        [error_code] => E000
        [bank_ref_no] => 261662040340351
        [ibibo_code] => CC
        [mode] => CC
        [ip] => 217.6.59.133
        [card_no] => 512345XXXXXX2346
        [cardtype] => domestic
        [offer_key] =>
        [field2] => 999999
        [udf1] =>
        [pg_mid] => 90000970
        [offer_type] =>
        [failure_reason] =>
        [mer_service_fee] => 0.70
```

[mer_service_tax] => 0.09

)

)

)

b) For successful web-service execution, but empty response (i.e. No transactions found):

```
array
(
   [status] => 1
   [msg] => Transaction Fetched Successfully
   [Transaction_details] => Array
      (
      )
)
```

c) Failed case:

In case of invalid input date format, output would be of the following form:

```
array
(
   [status] => 0
   [msg] => Invalid Date Entered. Date format should be yyyy-mm-dd
)
```

18) get_transaction_info

This API works exactly the same way as **get_Transaction_Details** API. The only enhancement is that this API can take input as the exact time in terms of minutes and seconds also. Output would be in the same format as get_Transaction_Details API output.

Input variable description:

Parameter	Description	Sample Value
var1	Starting Time (From when the transaction details are needed) in yyyy-mm-dd hh:mm:ss format	2014-01-12 16:00:00
var2	Starting Time (Till when the transaction details are needed) in yyyy-mm-dd hh:mm:ss format	2014-01-12 16:15:00

Web Service Responses:

The status variable would be 1 for successful web-service execution and would be 0 in case of unsuccessful web-service execution. Output would be returned in the following array format:

a) For Successful Response, status=1:

```
array
    (
        [status] => 1
            [msg] => Transaction Fetched Successfully
            [Transaction_details] => Array
        (
          [0] => array
            (
                [id] => 403993715508970248
                [status] => failed
                [key] => C0Dr8m
                [merchantname] => test payu
                [txnid] => e1e8a8f4ace8506043e1
                [firstname] => John
                [lastname] => Moses
                [addedon] => 2014-02-04 01:25:38
                [bank_name] => Visa Debit Cards (All Banks)
                [payment_gateway] => AXIS
                [phone] => 9585475883
                [email] => y.johnmoses@gmail.com
                [amount] => 100.00
                [discount] => 0.00
                [additional_charges] => 0.00
                [productinfo] => CSIIT Conference Registration
                [error_code] => E312
                [bank_ref_no] => 2000112693
                [ibibo_code] => VISA
                [mode] => DC
                [ip] => 117.206.82.90
                [card_no] => 414367XXXXXX0250
                [cardtype] => international
                [offer_key] =>
                [field2] => 403506432293
                [udf1] =>
                [pg_mid] => TESTIBIBOWEB
                [offer_type] =>
                [failure_reason] =>
                [mer_service_fee] =>
                [mer_service_tax] =>
            )

        [1] => Array
            (
                [id] => 403993715508970268
                [status] => captured
                [key] => C0Dr8m
                [merchantname] => test payu
                [txnid] => 8613914632655135
```

```
        [firstname] => Hans Wurst
        [lastname] =>
        [addedon] => 2014-02-04 03:03:06
        [bank_name] => Credit Card
        [payment_gateway] => HDFC
        [phone] =>
        [email] => f606f938f64b499aa3fd952d6338aa54@example.com
        [amount] => 30.00
        [discount] => 0.00
        [additional_charges] => 0.00
        [productinfo] => 3752946
        [error_code] => E000
        [bank_ref_no] => 1953525040340351
        [ibibo_code] => CC
        [mode] => CC
        [ip] => 217.6.59.133
        [card_no] => 512345XXXXXX2346
        [cardtype] => domestic
        [offer_key] =>
        [field2] => 999999
        [udf1] =>
        [pg_mid] => 90000970
        [offer_type] =>
        [failure_reason] =>
        [mer_service_fee] => 0.70
        [mer_service_tax] => 0.09
    )

[2] => Array
    (
        [id] => 403993715508970270
        [status] => captured
        [key] => C0Dr8m
        [merchantname] => test payu
        [txnid] => 8813914632908201
        [firstname] => Hans Wurst
        [lastname] =>
        [addedon] => 2014-02-04 03:03:30
        [bank_name] => Credit Card
        [payment_gateway] => HDFC
        [phone] =>
        [email] => 89163cd22823449d89e6d5cd2346fea3@example.com
        [amount] => 30.00
        [discount] => 0.00
        [additional_charges] => 0.00
        [productinfo] => P172
        [error_code] => E000
        [bank_ref_no] => 261662040340351
        [ibibo_code] => CC
        [mode] => CC
```

```
            [ip] => 217.6.59.133
            [card_no] => 512345XXXXXX2346
            [cardtype] => domestic
            [offer_key] =>
            [field2] => 999999
            [udf1] =>
            [pg_mid] => 90000970
            [offer_type] =>
            [failure_reason] =>
            [mer_service_fee] => 0.70
            [mer_service_tax] => 0.09

                    )
            )
    )
```

b) For successful web-service execution, but empty response (i.e. No transactions found):

```
array
(
   [status] => 1
   [msg] => Transaction Fetched Successfully
   [Transaction_details] => Array
      (
      )
)
```

c) Failed case:

In case of invalid input date format, output would be of the following form:

```
array
(
   [status] => 0
   [msg] => Invalid Date Entered. Date format should be yyyy-mm-dd hh:mm:ss
)
```

19) check_isDomestic

This API is used to detect whether a particular bin number is international or domestic. It is also useful to determine the card's issuing bank, the card type brand – i.e, Visa, Master etc and also the Card Category – i.e. Credit/Debit etc. Bin number is the first 6 digits of a Credit/Debit card.

Input Variables description:

Parameter	Description	Sample Value
var1	Card Number/Bin(First 6 digits of a card)	512345

Web Service Responses:

Case a: If the card is domestic

array
(
[isDomestic] => Y
[issuingBank] => HDFC
[cardType] => MAST
[cardCategory] => CC
)

Case b: If the card is international

array
(
[isDomestic] => N
[issuingBank] => UNKNOWN
[cardType] => UNKNOWN
[cardCategory] => CC
)

Here in the output,

- isDomestic = Y signifies that the particular bin is domestic.
- isDomestic = N signifies that the particular bin is International.
- cardType = <value> which can be be ['MAST','VISA','MAES','AMEX', 'DINR','Unknown']
- [issuingBank] = The issuing bank of the card used for transaction
- [cardCategory] = CC signifies that the particular bin is a Credit Card Bin
- [cardCategory] = DC signifies that the particular bin is a Debit Card Bin

Note: This API would give the output based upon PayU's bin list which may not be completely exhaustive.

API's 20-23 are related to PayU's Store Card Feature

20) get_user_cards

This API is used to fetch all the cards corresponding to the user. In this API, card number and other sensitive information is not returned.

Input Variables description:

Parameter	Description	Sample Value
var1	user_credentials (In the format- MerchantKey:UserIdentifier)	JQBlG:abc

Web Service Responses:

Case a: Cards are found in the vault.

Response:

```
array
(
   [status] => 1
   [msg] => Cards fetched Succesfully
   [user_cards] => Array
      (
         [745d72e2fd9b7e88824fef4e7ed7dac1fe624b74] => Array
            (
               [name_on_card] => {name}
              [card_name] => nickname but if sent empty then (cardType****last 4 digits of card) e.g. mastercard****2346
               [card_type] => CC(ibibo_code)
               [card_token] => 745d72e2fd9b7e88824fef4e7ed7dac1fe624b74
               [is_expired] => 1(1 when card is expired , 0 when not)
               [card_mode] => CC(card Category)
               [card_no] => 412345xxxxxx2356(masked Card Number)
               [card_brand] => VISA
               [card_bin] => 412345
                        [expiry_year] => 2017
                        [expiry_month] => 10
            )
      )
)
```

Case b: No cards are found for the user

```
array
(
   [status] => 0
   [msg] => Card not found.
)
```

21) save_user_card

This API is used for saving a card to the vault. On successful storing of the card, it returns the cardToken.

Input Variables description:

Parameter	Description	Sample Value
var1	user_credentials - merchantKey:userId	JQBlG:abc
var2	cardName(nickname of the card)	My_card
var3	cardMode	CC
var4	cardType	AMEX
var5	nameOnCard	Nitesh Jindal
var6	cardNo	5123456789012345
var7	cardExpMon	9
var8	cardExpYr	2014

Case a: When card is stored successfully

Web Service Responses:

array
(
[status] => 1
[msg] => Card Stored Successfully.
[cardToken] => 745d72e2fd9b7e88824fef4e7ed7dac1fe624b74
)

Case b: Any of the field is invalid

If card Number is invalid:

Web Service Response:

array
(
[status] => 0
[msg] => CardNumber is invalid
)

22) edit_user_card

This API is used to edit the details of an existing stored card in the vault. In this case, along with all the parameters that are required to save to the card, cardToken of the card to edit is also required to be passed. On successfully editing the card, it returns the cardToken of the card.

Input Variables description:

Parameter	Description	Sample Value
var1	User Credentials - MerchantKey:UserId	JQBlG:abc
var2	cardToken(card token of the card to edit)	745d72e2fd9b7e88824fef4e7ed7dac1f
var3	cardName(nickname of the card)	My_card
var4	cardMode	CC
var5	cardType	AMEX
var6	nameOnCard	Nitesh Jindal
var7	cardNo	5123456789012345
var8	cardExpMon	9
var9	cardExpYr	2014

Case a: On successful editing of card

Web Service Response:

array
(
[status] => 1
[msg] => {cardName} Edited Successfully.
[cardToken] => 745d72e2fd9b7e88824fef4e7ed7dac1fe624b74
)

Case b: If the wrong card token is given to edit

Web Service Response:

array
(
[status] => 0
[msg] => Card not found to edit
)

23) delete_user_card

This API is used to delete a card.

Input Variables description:

Parameter	Description	Sample Value
var1	user_credentials - merchantKey:userId	JQBlG:abc
var2	cardToken (cardtoken of the card to delete)	745d72e2fd9b7e88824fef4e7ed

Web Service Responses:

Case a: On successful deletion of card

array
(
'status' => 1,
'msg' => {cardName} deleted successfully'
)

Case b: on failure of deletion

array
(
'status' => 0,
'msg' => error reason
)