**Mapping the payload parameters to actual Database tables.**

Green color: methods and classes

Red color: tables

Bold: important points

**BILLREQUEST:**

* Here **BillRequest** comes to Bill.java. So flow starts in this class.
* Here OPERATOR\_NAME is predefinely set to **Paytm**. We get the connection from billdata.CommonUtility.*getUn*(),billdata.CommonUtility.*getPwd*() [connecting to database using username and password].
* Here payment gateway [pg = paytm] is defined. If(pg.equals(pg.equalsIgnoreCase("PayTM") || pg.equalsIgnoreCase("Cashfree") then OPERATOR\_NAME is again changed to pg value.
* Here we taken a string variable ie., **wallet\_billing\_provider** which gets the value from BillUtility.*getSetting*(con,"wallet\_billing\_provider");. This method is in **BillUtility class**. In this method, we pass wallet\_billing\_provider to the s\_field then a query is executed where s\_field column in **settings table** is equal to “wallet\_billing\_provider” then we fetch the corresponding **s\_value** column where s\_field = “wallet\_billing\_provider”, and we assign this s\_value to wallet\_billing\_provider. And the **s\_value is either PayTM or Cashfree.**
* Then we assign this wallet\_billing\_provider value to OPERATOR\_NAME. Next here we declare **bp\_group\_id** [Bllingprovider id] which we assign the value using GroupDAO.*getGroupId*(con, OPERATOR\_NAME); method.
* bp\_group\_id value can be fetched by using group\_Id column in **groups table** where group\_name = “PayTM” and returns the **group\_id[for paytm groupid is 2]** to calling method.
* After that inorder to get the bill parameters we will call a method PayTMWebUtility.*getPrebillBeanForPayTM();* by passing some values of request, response, operator\_name, bp\_group\_id. In this method, we set the prebillBean by using request.getParameter(value); as we passed the request as argument for this method.
* Here if we want **customerEmail** we set the customerEmail in prebillBean using GroupDAO.*getDefaultCustomerEmail()*. In this method, we set the **gmail** by using **groups table** where group\_id = merchant\_id[we pass this mid as argument to this method] and set the customerEmail to gmail.
* As well as we set the **customerName** by calling the method GroupDAO.*getDefaultCustomerName()* by passing mid, paytm\_group\_id. Here also we set the customerName by using **groups table**. Here we select the customerName[in groups table it is referred to **group\_name**] by executing a query where group\_id= paytm\_group\_id and operator='A'. Then we select the corresponding group\_name which is assigned to customerName. As well as **customerPhoneNumber()** can be found based on the same process but here group\_id= mid [passed as argument] and sets customerPhoneNumber[g\_phone\_number in groups table].
* When all these parameters are set to prebillBean then we store the all these parameters in a table by calling a method ParameterTrackingDAO.*storepayTMRequest()* by passing prebillBean, paytm\_group\_id, BillRequest etc., In this method, we store all these parameter values[usually called as BillRequest Parameters] in **wipay\_request\_response\_w table** in wipaydb database. Along with that we also store these values in another table by calling WiPayActionDAO.*storePTMCBRDBillResponse()* method by passing same above parameters and store them in **wipay\_action table** in the same database.
* Finally this method returns prebillBean to calling method in Bill.java class and assigns to prebillbean object. Here group\_name gets its values from by calling GroupDAO.*getAgencyName(*) method. In this method we use a query to find the group\_name in **groups table**. We select **group\_name[Paytm]** corresponding to group\_id[for paytm group\_id is 2] and operator=’**A**’ and returns the group\_name to the calling method.
* **if**(groupName==**null**) then we set proceed to false and we release the connection and passes the status message as {status\_code:" B105 ", status\_message: Billing Failure Your request cannot be processed due to Invalid Request, The requested Merchant Doesn't Exists or not configured, Please contact Administrator"} to browser.

**API VALIDATION :**

* After that next step is **API validation**. For this we call a method PayTMAPIValidation.*validateParams*(preBillBean); by passing preBillBean. Then we go to this method which is in **PayTMAPIValidation.java** class where we validate parameters. All these parameters that are validated are get by using prebillBean.get() method.
* Here the parameters we valid are
  + - **MerchantId**, is MerchantId numeric or not,
    - **OrderId**, if OrderId value less than 25 or not [orderid value must be less than 25,
    - **contentId**,
    - **sign**, if sign is valid or not[sign value must be valid],
    - **title**, checking if any prohibited words are is in the title by calling the method billdata.Utility.*isProhibitedWord()*. If any prohibited word is present then it will return false, again if title value less than 500 or not[title value must be less than 500],
    - **price**,if price is numeric or not[price must be numeric], price length less than 5 or not[price length must be less than 5],
    - if **BillType** is event or valuepack[BillType must be event or valuepack],
    - **minPrice**, if minPrice is less than price or not[minPrice must be less than price.

All these above validations are done in validateParams() method. If any one of these validation is failed the following errorcode is set based on the parameter, along with that proceed value is set to false returns prebillBean to calling method.

* If any one of the validation is failed. Then **isProceed** value is set to **false**.

**CALLBACK:**

* After that checks the condition **if(!(preBillBean.isProceed()))**, if its true it sends the **merchant callback and redirection** by stating **validation failed** and passing the status codes and status message[These are taken from the class WipayErrorCodes.java by calling the method *getError()* and gets assigned to them].
* This error response is stored in a table by calling WiPayActionDAO.*storePTMCBRDBillResponse*() method where we store these response parameters in **wipay\_action table**.
* After that, we need a **complete\_orderUrl** inorder to send merchant callback status. So we define a string variable complete\_orderUrl where we assign a method PayTMDAO.getMerchantCopleteOrderURL() where we pass merchantID as a parameter.
* So, here the flow goes to PayTMDAO.java class. Here in getMerchantCopleteOrderURL() method, we execute a query to get the url. We have to select the s\_value from settings table where **s\_field='merchant\_complete\_order\_url'** and **group\_id="+aGid+"**[Here ‘aGid’ is **merchantId**. For Samsung it is 6]. So we assign the s\_value(url) corresponding to these values to callbackUrl and returns to calling method[Bill.java].
* if(complete\_orderUrl!=null && !complete\_orderUrl.equals("")), if this condition is true, then it will send the merchant callback using PayTMHelper.sendMerchantCallback() method by passing the parameters preBillBean, STATUS, RESPMSG, RESPCODE, status\_code, status\_msg, Via, complete\_orderUrl.
* **Callback parameters** we send to merchant are [usually we call them as **callback status parameters**]:

mid

CURRENCY

TXNID

PAYMENTMODE

ORDERID

TXNAMOUNT

RESPMSG

STATUS

RESPCODE

status\_code

status\_msg

* we send these parameters using json object and sends them by using the method PayTMTLSFireURL() by passing the url[ie., complete\_order\_url,cbparams(callback params as json)].
* After that we return to Bill.java and stores these values in a table by calling ParameterTrackingDAO.storePTMCBRDBillResponse() by passing prebillBean other callback parameters. In this method, we store these parameters in **wipay\_request\_response\_w table** in wipaydb. After storing we return to Bill.java. Along with that we also store these parameters in another table by calling WiPayActionDAO.*storePTMCBRDBillResponse()* method and stores in **wipay\_action table** in the same database.

**REDIRECTION:**

* For sending Redirection, we need **merchant\_redirect\_url**. So, here we define a string variable merchant\_redirect\_url where we assign a method PayTMDAO.getMerchantRedirectURL() where we pass merchantID as a parameter.
* So, here the flow goes to PayTMDAO.java class. Here in getMerchantRedirectURL() method, we execute a query to get the url. We have to select the **s\_value** from settings table where **s\_field=’re\_ip'**[which is called redirection ip address, for **eg:** **http://dev.wicore.in/WiPayTest/redirection.jsp**] and **group\_id="+aGid+"**[Here ‘aGid’ is **merchantId**. For Samsung it is 6]. So we assign the s\_value(url) corresponding to these values to redirectUrl and returns to calling method[Bill.java].
* **if(merchant\_redirect\_url!=null && ! merchant\_redirect\_url.equals("")),** if this condition is true, then it will assign PayTMHelper.getMerchantRedirectionURL() method to merchant\_redirect\_url by passing preBillBean, STATUS, RESPMSG, RESPCODE, status\_code, status\_msg, Via, merchant\_redirect\_url which returns url in **json format**. Then it will send the merchant redirection using response.sendRedirect(merchant\_redirect\_url); method.
* After we store these parameters in **wipay\_request\_response\_w table** by calling a method ParameterTrackingDAO.*storePTMCBRDBillResponse()* by passing redirection parameters. As well as we store these parameters in **wipay\_action table** by calling WiPayActionDAO.*storePTMCBRDBillResponse()* method by passing redirecton parameters.
* **Redirection parameters we send to merchant are:**

**Redirection url:**

**1.for event**

http:////callbackURL?mid=182&ORDERID=182- 180620141453&CURRENCY=INR&TXNID=358335039&TXNAMOUNT=100&PAYMENTMODE=DC&RESPMSG=Cancel Request by Customer&STATUS=TXN\_FAILURE&RESPCODE=141&status\_code=B105&status\_msg=Billing%20Status%20%28Billing%20Failure%29&sign=

**2. for subscription**

http:////callbackURL?mid=182&cid=wipay\_product\_id&ORDERID=182180620 141453&CURRENCY=INR&TXNID=358335039&TXNAMOUNT=100&PAYMENTMODE=DC &action=NEW&RESPMSG=CancelRequestbyCustomer&STATUS=TXN\_FAILURE&RESPCODE= 141&status\_code=B105&status\_msg=Billing%20Status%20%28Billing%20Failure%29&sign=

* Database Name: wipaydb

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Parameters** | **Column\_name in table** | **Table name** | **Type of value stored in table** | **Paremeter Description** |
| 1.mid | mid,  wa\_mid | wipay\_request\_response\_w & Wipay\_action | Eg:6[for samsung it is 6]  Eg:193 | It is merchant ID. We have separate merchant id for each merchants. Wipay[We] assigned them in groups table |
| 2.CURRENCY | CURRENCY | Paytm\_history | Eg: INR | Type of currency. |
| 3.TXNID | TXNID | Paytm\_history | Eg: 32333346 |  |
| 4.ORDERID | order\_id,  wa\_order\_id | wipay\_request\_response\_w & Wipay\_action | 20211118060332G78909967X | Order\_id is given by merchant. It is unique |
| 5.TXNAMOUNT | TXNAMOUNT | Paytm\_history | Eg: 64 | Cost of the product given by merchant for payment |
| 6. RESPMSG | RESPMSG | Paytm\_history | Eg: CancelRequestbyCustomer | It describes the response message during payment either payment is success or not. |
| 8. STATUS | STATUS,wa\_status | Paytm\_history | Eg: TXN\_FAILURE | Status of the billing |
| 9. RESPCODE | RESPCODE | Paytm\_history | Eg: 141 | Code |
| 10. status\_code | status\_code, | wipay\_request\_response\_w | Eg: V035  Eg: B107 | Error code which we predefinely assigned for each status message in WipayErrorCodes.java class |
| 11. status\_msg | status\_message | wipay\_request\_response\_w | Eg: Invalid sign  Eg: order\_id is not validated from Server (Billing Failure) | Status message also predefinely assigned for each status code in WipayErrorCodes.java class |
| 12.sign | sign | wipay\_request\_response\_w | Eg: 1bfa83d9ad01b812353daa50afb7407e  Eg: 45abd63dffa5d703ef355b2bd06a772d | Sign is specially generated by wipay in a MD5Sign.getMD5() method |
| 13. cid | cid, wa\_cid | wipay\_request\_response\_w & wipay\_action | Eg:000003023578  Eg: 000002685412 | cid is content id based on the content user is selected. When we perform only event billing it is not required.For subscription it is required. |
| 14. action | action, wa\_action | wipay\_request\_response\_w & wipay\_action | Eg: New  Eg: RENEWAL | It may be called as subscription status.It is also required only for subscription. |

**MERCHANT VALIDATION:**

* In this step, we do merchant validation, so we call this PayTMDAO.getMerchantValidateURL() method and assigned to **validateURL**. In this method we execute a query which selects **s\_value** from **settings table** based on the **s\_field='bp\_validate\_ip'**[billing provider validate ip, for eg: **http://dev.wicore.in/WiPay/test/paytm/validateMerchantOrder**] and **group\_id='"+aGid+"'**"[merchant id] and assigns the s\_value to paytm\_validate\_ip and returns the value to calling method.
* After that we call a method PayTMWebUtility.validateOrderForWEB() by passing the parameters like request, groupName, ds, Via, validateURL, preBillBean and assigns it to a variable **isValidOrder.** In this method, we set all these parameters to a **json object named orderParam.**
* **if( validateURL != null )** , if this condition is true then it will call URLUtility.PayTMReadFireURLRetry(validateURL, "POST", orderJSON, false, Via) method which **opens a connection between merchant & wipay** and validates the request and returns response in **string format**.
* After that we execute a condition **if( validateWebRespone(orderJSON, resJSON, Via, preBillBean))** which calls this method and validates the parameters in validateWebRespone() method using the response got from merchant.
* If all the parameters validations return true, then **isvalid** is assigned to true; and returns this isvalid value to calling method[Bill.java].
* After that **if(isValidOrder),** if this condition is true, then OrderBean sets its parameters by using prebillBean. After that **ctype\_id** is set to call a method ContentTypeDAO.getContentTypeId() which returns ctype\_id. In this method, we execute a query which selects content\_id from **content\_types table** where content\_names\_ocg= '"+ctype\_id+"'"; then this content\_id is assigned to ctype\_id.
* After that these merchant validation parameters are stored in a table by calling WiPayActionDAO.storePTMCBRDBillResponse() method which stores these parameters in wipay\_action table.

**Merchant validation parameters are:**

**Merchant validation request and response data:**

**1.for event and valuepack billing**

Request {"mid":"182","price":"1","order\_id":"123","ctype":"games","cid":"123"}

Response Data: {"result":{"mid":"182","price":"1","order\_id":"123","ctype":"games","cid":"123"},"error\_message":""," error\_code":"-1"} {"result":null,"error\_message":"Invalid request","error\_code":"101"}

**2. for subscription**

Request {"mid":"182","price":"49","order\_id":"123","title":"MT+Chella+Std+Medium+Latin+FlipFont","ctype":"games",bill\_type":”SUBSCRIPTION”, "cid":"Subscrion\_Product\_ID","sign":"bf57af02456fad02e2eef2cd6da6030c" }

Response Data for subscription: {"result": {"mid":"182","price":"1","order\_id":"123","ctype":"SUBSCRIPTION","cid":" Subscrion\_Product\_ID ",”bill\_type”:”SUBSCRIPTION”,”title”:” MT+Chella+Std+Medium+Latin+FlipFont”},"error\_message":"","error\_code":"-1",”sign”:”generated sign”} {"result":null,"error\_message":"Invalid request","error\_code":"101"}

* Database Name: wipaydb

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Parameters** | **Column\_name in table** | **Table name** | **Type of value stored in table** | **Parameter Description** |
| 1.mid | mid, Wa\_mid | wipay\_request\_response\_w & Wipay\_action | Eg:6[for samsung it is 6]  Eg:193 | It is merchant ID. We have separate merchant id for each merchants. Wipay[We] assigned them in groups table |
| 2. cid | cid, Wa\_cid | wipay\_request\_response\_w & Wipay\_action | Eg:000003023578  Eg: 000002685412 | cid is content id based on the content user is selected. When we perform event billing cid is sent by merchant.For subscription it is shared by wicore. |
| 3.ctype | ctype, Wa\_ctype | wipay\_request\_response\_w & Wipay\_action | Eg: VALUEPACK | Here ctype gets its value from bill\_type in which its default value is VALUEPACK during merchant validation |
| 4.price | price | wipay\_request\_response\_w | Eg:1 | Price of the content purchased by user |
| 5.ORDERID | order\_id, Wa\_order\_id | wipay\_request\_response\_w & Wipay\_action | 20211118060332G78909967X | Order\_id is given by merchant. It is unique |
| 6. title | title | wipay\_request\_response\_w | Eg: PKrosemary%E2%84%A2 | It describes the title of the product. It is sent by merchant. |
| 7. error\_code |  |  | Eg: -1 | Error codes are sent by merchant based on validation failure or success |
| 8. error\_msg |  |  |  | error\_messages are also sent by merchant |
| 9.sign | sign | wipay\_request\_response\_w | Eg: 1bfa83d9ad01b812353daa50afb7407e  Eg: 45abd63dffa5d703ef355b2bd06a772d | Sign is specially generated by wipay in a MD5Sign.getMD5() method |
| 10. bill\_type | bill\_type | wipay\_request\_response\_w | Eg: SUBSCRIPTION | Bill\_type is only required for subscription billing. For event billing it is not required. Bill\_type |

Redirect and merchant validation parameters are completed.

**BILLING:**

* **if(OPERATOR\_NAME.equalsIgnoreCase("PayTM"))**, if this condition is true then Billing process will continue.