

solution Document of Merchant portal

April 2025

|  |  |
| --- | --- |
| **Document Control Sheet** | |
| **Current Version** | 0.4 |
| **Project Code** | NS\_SBI\_04\_08\_2024 |
| **Project Name** | SBI ePay Implementation |
| **Document Type** | ePay Merchant Portal |
| **Author** | Vikram Deshpande |
| **First Reviewed By** | Ranu Jain |
| **Final Reviewed By** |  |
| **Approved By** |  |
| **Contact Person** | Ranu Jain |
| **Frequency of Document Review** | Next Revision of Standard Documents |
| **Document Creation Date** | 14-03-2025 |
| **Last Updated Date** |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision History** | | | |
| **Version** | **Date** | **Name** | **Comments** |
| 0.1 | 14-03-2025 | Vikram Deshpande | Draft Version |
| 0.2 | 09-04-2025 |  |  |

**Contents**

[1. Document Purpose 24](#_Toc195091184)

[2. Technical Design 25](#_Toc195091185)

[2.1 Technology Stack 25](#_Toc195091186)

[2.2 Generic Class Flow 26](#_Toc195091187)

[2.3 Generic Header 26](#_Toc195091188)

[2.4 Generic Request Body 26](#_Toc195091189)

[2.5 Generic Response Body 26](#_Toc195091190)

[2.6 Error Codes 27](#_Toc195091191)

[3. SD\_MS 1 – Merchant Portal – Login 28](#_Toc195091192)

[3.1 Captcha 28](#_Toc195091193)

[3.1.1 Class Flow Diagram 28](#_Toc195091194)

[3.1.1.1 API Definition / Implementation 28](#_Toc195091195)

[3.1.1.2 Validation 30](#_Toc195091196)

[3.1.1.2.1 Request Data Validation 30](#_Toc195091197)

[3.1.1.2.2 Business Validation 30](#_Toc195091198)

[3.1.1.3 Business Logic 30](#_Toc195091199)

[3.1.1.4 GemFire cache region – Merchant\_Captcha\_Management 30](#_Toc195091200)

[3.1.1.5 Information 31](#_Toc195091201)

[3.1.2 User Validation 31](#_Toc195091202)

[3.1.2.1 API Definition / Implementation 31](#_Toc195091203)

[3.1.2.2 Validation 32](#_Toc195091204)

[3.1.2.3 Business logic 32](#_Toc195091205)

[3.1.2.4 Database Table – MERCHNT\_USER 33](#_Toc195091206)

[3.1.2.5 Information 34](#_Toc195091207)

[3.1.3 Merchant Login 34](#_Toc195091208)

[3.1.3.1 Class Diagram 34](#_Toc195091209)

[3.1.3.2 API Definition / Implementation 34](#_Toc195091210)

[3.1.3.3 Validation 37](#_Toc195091211)

[3.1.3.4 Business Logic 37](#_Toc195091212)

[3.1.3.5 Database Table – MERCHANT\_USER 37](#_Toc195091213)

[3.1.3.6 Information 38](#_Toc195091214)

[3.1.4 OTP Generation 38](#_Toc195091215)

[3.1.4.1 Class Diagram 38](#_Toc195091216)

[3.1.4.2 API Definition / Implementation 38](#_Toc195091217)

[3.1.4.3 Validation 41](#_Toc195091218)

[3.1.4.3.1 Request Data Validation 41](#_Toc195091219)

[3.1.4.3.2 Business Validation 41](#_Toc195091220)

[3.1.4.4 Business Logic 41](#_Toc195091221)

[3.1.4.5 GemFire cache region – OTP\_MANAGEMENT 41](#_Toc195091222)

[3.1.4.6 Information 42](#_Toc195091223)

[3.1.5 OTP Regeneration 43](#_Toc195091224)

[3.1.5.1 Class Diagram 43](#_Toc195091225)

[3.1.5.2 API Definition / Implementation 43](#_Toc195091226)

[3.1.5.3 Validation 46](#_Toc195091227)

[3.1.5.4 Business Logic 46](#_Toc195091228)

[3.1.5.5 GemFire Cache Region – OTP\_MANAGEMENT 46](#_Toc195091229)

[3.1.5.6 Information 47](#_Toc195091230)

[3.1.6 OTP Validation 47](#_Toc195091231)

[3.1.6.1 API Definition / Implementation 47](#_Toc195091232)

[3.1.6.2 Validation 49](#_Toc195091233)

[3.1.6.2.1 Request Data Validation 49](#_Toc195091234)

[3.1.6.2.2 Business Validation 49](#_Toc195091235)

[3.1.6.3 Business Logic 49](#_Toc195091236)

[3.1.6.3.1 GEMFIRE Region – Merchant\_OTP\_MANAGEMENT 49](#_Toc195091237)

[3.1.6.3.2 GEMFIRE Region – Merchant\_Token\_MANAGEMENT 50](#_Toc195091238)

[3.1.6.4 Information 50](#_Toc195091239)

[3.1.7 Change Password 51](#_Toc195091240)

[3.1.7.1 Class Diagram 51](#_Toc195091241)

[3.1.7.2 API Definition / Implementation 51](#_Toc195091242)

[3.1.7.3 Validation 53](#_Toc195091243)

[3.1.7.4 Business Logic 53](#_Toc195091244)

[3.1.7.5 Database Table – Update MERCHANT\_USER and PASSWORD\_MANAGEMENT 54](#_Toc195091245)

[3.1.7.6 Information 55](#_Toc195091246)

[3.1.8 Reset Password 55](#_Toc195091247)

[3.1.8.1 API Definition / Implementation 55](#_Toc195091248)

[3.1.8.2 Validation 57](#_Toc195091249)

[3.1.8.2.1 Request Data Validation 57](#_Toc195091250)

[3.1.8.3 Business Logic 58](#_Toc195091251)

[3.1.8.4 Database table – Update MERCHANT\_USER and PASSWORD\_MANAGEMENT 58](#_Toc195091252)

[3.1.8.5 Information 58](#_Toc195091253)

[3.1.9 Token Validation 59](#_Toc195091254)

[3.1.9.1 API Definition / Implementation 59](#_Toc195091255)

[3.1.9.2 Validation 60](#_Toc195091256)

[3.1.9.3 Business Logic - NA 60](#_Toc195091257)

[3.1.9.4 Information - NA 60](#_Toc195091258)

[3.1.10 Merchant User Captcha Validation 61](#_Toc195091259)

[3.1.10.1 API Definition / Implementation 61](#_Toc195091260)

[3.1.10.2 Validation 62](#_Toc195091261)

[3.1.10.3 Business Logic 62](#_Toc195091262)

[3.1.10.4 Information 62](#_Toc195091263)

[3.1.10.5 Validate logged in user has access to active mId 64](#_Toc195091264)

[3.1.10.6 API Definition / Implementation 64](#_Toc195091265)

[3.1.10.7 Validation 65](#_Toc195091266)

[3.1.10.8 Business Logic 65](#_Toc195091267)

[3.1.10.9 Information 65](#_Toc195091268)

[3.1.11 Validate logged in user has access to particular mId 66](#_Toc195091269)

[3.1.11.1 API Definition / Implementation 66](#_Toc195091270)

[3.1.11.2 Validation 67](#_Toc195091271)

[3.1.11.3 Business Logic 67](#_Toc195091272)

[3.1.11.4 Information 67](#_Toc195091273)

[4. SD\_MS - Merchant Portal – Merchant Panel 67](#_Toc195091274)

[4.1 Menu Bar 67](#_Toc195091276)

[4.1.1 Theme button – UI implementation. Users can select dark, light themes. 68](#_Toc195091277)

[4.1.2 Merchant Portal – User Alerts. 68](#_Toc195091278)

[4.1.2.1 Class Diagram 68](#_Toc195091279)

[4.1.2.2 Merchant Portal - Update User Alerts (TBD) 68](#_Toc195091280)

[4.1.2.2.1 API Definition and Implementation 68](#_Toc195091281)

[4.1.2.2.2 Validation 69](#_Toc195091282)

[4.1.2.2.3 Business Logic 69](#_Toc195091283)

[4.1.2.2.4 Database table – ALERT\_MANAGEMENT 70](#_Toc195091284)

[4.1.2.2.5 Information 70](#_Toc195091285)

[4.1.2.2.6 Merchant Portal – Update User Alerts 70](#_Toc195091286)

[4.1.2.2.7 API Definition and implementation 70](#_Toc195091287)

[4.1.2.2.8 Validation 71](#_Toc195091288)

[4.1.2.2.9 Business Logic 72](#_Toc195091289)

[4.1.2.2.10 Database Table – ALERT\_MANAGEMENT 72](#_Toc195091290)

[4.1.2.2.11 Information 72](#_Toc195091291)

[4.1.2.3 User Profile – Login User Profile, API defined in section 2.6.1.1 73](#_Toc195091292)

[4.1.2.4 Merchant Portal - User Logout 73](#_Toc195091293)

[4.1.2.4.1 Class diagram 73](#_Toc195091294)

[4.1.2.4.2 API Definition and Implementation 73](#_Toc195091295)

[4.1.2.4.3 Validation 74](#_Toc195091296)

[4.1.2.4.4 Business logic 75](#_Toc195091297)

[4.1.2.4.5 Database table – TOKEN\_MANAGEMENT 75](#_Toc195091298)

[4.1.2.4.6 Information 75](#_Toc195091299)

[4.2 Dashboard 75](#_Toc195091300)

[4.2.1 Class Diagram 76](#_Toc195091301)

[4.2.2 Transaction Trends 76](#_Toc195091302)

[4.2.2.1 API Definition and Implementation 76](#_Toc195091303)

[4.2.2.1.1 Validation 78](#_Toc195091304)

[4.2.2.1.2 Business Logic 78](#_Toc195091305)

[4.2.2.1.3 Information 79](#_Toc195091306)

[4.2.2.1.4 Database View 79](#_Toc195091307)

[4.2.3 Transaction Summary 80](#_Toc195091308)

[4.2.3.1 API Definition and Implementation 80](#_Toc195091309)

[4.2.3.1.1 Validation 82](#_Toc195091310)

[4.2.3.1.2 Business Logic 82](#_Toc195091311)

[4.2.3.1.3 Information 82](#_Toc195091312)

[4.2.3.1.4 Database Table 83](#_Toc195091313)

[4.2.4 Transaction – Refund Details (TBD) 83](#_Toc195091314)

[4.2.5 Transaction – Details (TBD) 83](#_Toc195091315)

[4.2.6 Transaction – Chargeback Details (TBD) 83](#_Toc195091316)

[4.2.7 Transaction – Daily Settlement Summary 83](#_Toc195091317)

[4.2.7.1 API Definition and Implementation 83](#_Toc195091318)

[4.2.7.2 Validation 84](#_Toc195091319)

[4.2.7.2.1 Business logic 84](#_Toc195091320)

[4.2.7.2.2 Information 85](#_Toc195091321)

[4.2.7.2.3 Materialized View 85](#_Toc195091322)

[4.2.8 Transaction – Daily Refund Summary 85](#_Toc195091323)

[4.2.8.1.1 API Definition and implementation 85](#_Toc195091324)

[4.2.8.1.2 Validation 86](#_Toc195091325)

[4.2.8.1.3 Business logic 86](#_Toc195091326)

[4.2.8.1.4 Information 87](#_Toc195091327)

[4.2.8.1.5 Materialized View 87](#_Toc195091328)

[4.2.9 Transaction – Daily Transaction Summary 87](#_Toc195091329)

[4.2.9.1.1 API Definition and implementation 87](#_Toc195091330)

[4.2.9.1.2 Validation 88](#_Toc195091331)

[o Check if a particular mId is available and valid, otherwise return an error message. 88](#_Toc195091332)

[4.2.9.1.3 Business logic 88](#_Toc195091333)

[4.2.9.1.4 Information 88](#_Toc195091334)

[4.2.9.1.5 Materialized View 89](#_Toc195091335)

[4.2.10 Transaction - Payment Mode Health 89](#_Toc195091336)

[4.2.10.1 Downtime API - Listing 89](#_Toc195091337)

[4.2.10.1.1 API Definition and implementation 89](#_Toc195091338)

[4.2.10.1.2 Validation - None 92](#_Toc195091339)

[4.2.10.1.3 Business Logic 92](#_Toc195091340)

[4.2.10.2 Downtime API - Listing 92](#_Toc195091341)

[4.2.10.2.1 API Definition and Implementation 92](#_Toc195091342)

[4.2.10.2.2 Business logic 105](#_Toc195091343)

[4.2.11 Transaction - Recent Transaction 106](#_Toc195091344)

[4.2.11.1 API Implementation 106](#_Toc195091345)

[4.2.11.1.1 Validation 107](#_Toc195091346)

[4.2.11.1.2 Business Logic 108](#_Toc195091347)

[4.2.11.1.3 Information 108](#_Toc195091348)

[4.2.11.1.4 Materialized View 108](#_Toc195091349)

[4.2.12 Transaction – Download file for recent transaction 108](#_Toc195091350)

[4.2.12.1 API Implementation 108](#_Toc195091351)

[4.2.12.1.1 Validation 110](#_Toc195091352)

[4.2.12.1.2 Business Logic 110](#_Toc195091353)

[4.2.12.1.3 Information 110](#_Toc195091354)

[4.2.12.1.4 Materialized View 110](#_Toc195091355)

[4.3 Reports 111](#_Toc195091356)

[4.3.1 Class Diagram 111](#_Toc195091357)

[4.3.1.1 Reports - Get All Reports 111](#_Toc195091358)

[4.3.1.1.1 API Definition / Implementation 111](#_Toc195091359)

[4.3.1.1.2 Validation - NA 113](#_Toc195091360)

[4.3.1.1.3 Business Logic 113](#_Toc195091361)

[4.3.1.1.4 Database table - REPORT\_MASTER 113](#_Toc195091362)

[4.3.1.1.5 Information 113](#_Toc195091363)

[4.3.2 Reports – Create a Report 113](#_Toc195091364)

[4.3.2.1 API Definition / Implementation 113](#_Toc195091365)

[4.3.2.2 Validation 115](#_Toc195091366)

[4.3.2.3 Business Logic 116](#_Toc195091367)

[4.3.2.4 Database Table - REPORT\_MANAGEMENT 116](#_Toc195091368)

[4.3.2.5 Information 116](#_Toc195091369)

[4.3.3 Reports - Get All Reports 117](#_Toc195091370)

[4.3.3.1 API Definition / Implementation 117](#_Toc195091371)

[4.3.3.2 Validation 119](#_Toc195091372)

[4.3.3.3 Business Logic 119](#_Toc195091373)

[4.3.3.4 Database Table - REPORT\_MANAGEMENT 119](#_Toc195091374)

[4.3.3.5 Information 119](#_Toc195091375)

[4.3.4 Reports - Schedule a report 120](#_Toc195091376)

[4.3.4.1 API Definition / Implementation 120](#_Toc195091377)

[4.3.4.2 Validation 122](#_Toc195091378)

[4.3.4.3 Business Logic 122](#_Toc195091379)

[4.3.4.4 Database Table – REPORT\_SCHEDULER\_MANAGEMENT 122](#_Toc195091380)

[4.3.4.5 Information 122](#_Toc195091381)

[4.3.5 Reports – Update Scheduled Report 123](#_Toc195091382)

[4.3.5.1 API Definition / Implementation 123](#_Toc195091383)

[4.3.5.2 Validation 125](#_Toc195091384)

[4.3.5.3 Business Logic 125](#_Toc195091385)

[4.3.5.4 Database table – REPORT\_SCHEDULER\_MANAGEMENT 125](#_Toc195091386)

[4.3.5.5 Information 125](#_Toc195091387)

[4.3.6 Reports - Get All Schedule Report Request 126](#_Toc195091388)

[4.3.6.1 API Definition / Implementation 126](#_Toc195091389)

[4.3.6.2 Validation - None 129](#_Toc195091390)

[4.3.6.3 Business Logic 129](#_Toc195091391)

[4.3.6.4 Database table 129](#_Toc195091392)

[4.3.6.5 Information 130](#_Toc195091393)

[4.3.7 Reports – Download generated report for Merchant 130](#_Toc195091394)

[4.3.7.1 API Definition / Implementation 130](#_Toc195091395)

[4.3.7.2 Validation 131](#_Toc195091396)

[4.3.7.3 Business Logic 131](#_Toc195091397)

[4.3.7.4 Business logic - NA 131](#_Toc195091398)

[4.3.7.5 Information 131](#_Toc195091399)

[4.3.8 Reports – Cancel Report Scheduler 131](#_Toc195091400)

[4.3.8.1 API Definition / Implementation 131](#_Toc195091401)

[4.3.8.2 Validation 133](#_Toc195091402)

[4.3.8.3 Business logic 133](#_Toc195091403)

[4.3.8.4 Database table – REPORT\_SCHEDULER\_MANAGEMENT 133](#_Toc195091404)

[4.3.8.5 Information 133](#_Toc195091405)

[4.3.9 Reports – Scheduler 134](#_Toc195091406)

[4.3.9.1 Recent Transaction 134](#_Toc195091407)

[4.3.9.2 Business logic 134](#_Toc195091408)

[4.3.9.3 Database tables: 134](#_Toc195091409)

[4.4 Invoice 134](#_Toc195091410)

[4.4.1 Report Download - Merchant Fees 135](#_Toc195091411)

[4.4.1.1 Class Diagram 135](#_Toc195091412)

[4.4.1.2 API Definition / Implementation 135](#_Toc195091413)

[4.4.1.3 Validation 136](#_Toc195091414)

[4.4.1.4 Business logic 136](#_Toc195091415)

[4.4.1.5 Database View 136](#_Toc195091416)

[4.4.1.6 Information 137](#_Toc195091417)

[4.4.2 Report Download - GST Invoice 137](#_Toc195091418)

[4.4.2.1 API Definition / Implementation 137](#_Toc195091419)

[4.4.2.2 Validation 138](#_Toc195091420)

[4.4.2.3 Business Logic 139](#_Toc195091421)

[4.4.2.4 Database View - VIEW\_TRANSACTION\_GST\_INVOICE 139](#_Toc195091422)

[4.4.2.5 Information 139](#_Toc195091423)

[4.5 User Management - User Management 139](#_Toc195091424)

[4.5.1 User Management – Merchant User Profile 139](#_Toc195091425)

[4.5.1.1 View User Profile 139](#_Toc195091426)

[4.5.1.2 API Definition and Implementation 139](#_Toc195091427)

[4.5.1.3 Validation 141](#_Toc195091428)

[4.5.1.4 Business logic 141](#_Toc195091429)

[4.5.1.5 Database table – MERCHANT\_USER 142](#_Toc195091430)

[4.5.1.6 Information 142](#_Toc195091431)

[4.5.2 Update User profile 142](#_Toc195091432)

[4.5.2.1 API Definition and implementation 142](#_Toc195091433)

[4.5.2.2 Validation 145](#_Toc195091434)

[4.5.2.3 Business Logic 145](#_Toc195091435)

[4.5.2.4 Database Table – MERCHANT\_USER 146](#_Toc195091436)

[4.5.2.5 Information 146](#_Toc195091437)

[4.5.3 Merchant Portal – Merchant User Access and Menu 146](#_Toc195091438)

[4.5.3.1 Get User Access and Menu 146](#_Toc195091439)

[4.5.3.2 API Definition and Implementation 146](#_Toc195091440)

[4.5.3.3 Validation 150](#_Toc195091441)

[4.5.3.4 Business Logic 150](#_Toc195091442)

[4.5.3.5 Database Table – USER\_MENU\_PERMISSION 151](#_Toc195091443)

[4.5.3.6 Information 151](#_Toc195091444)

[4.5.4 Assign User Access and Menu 152](#_Toc195091445)

[4.5.4.1 API Definition and Implementation 152](#_Toc195091446)

[4.5.4.2 Validation 153](#_Toc195091447)

[4.5.4.3 Business Logic 153](#_Toc195091448)

[4.5.4.4 Database Table – USER\_MENU\_PERMISSION 155](#_Toc195091449)

[4.5.4.5 Information 155](#_Toc195091450)

[4.5.5 Get User Access and Menu 156](#_Toc195091451)

[4.5.5.1 API Definition and Implementation 156](#_Toc195091452)

[4.5.5.2 Validation 159](#_Toc195091453)

[4.5.5.3 Business Logic 159](#_Toc195091454)

[4.5.5.4 Database Table – USER\_MENU\_PERMISSION 159](#_Toc195091455)

[4.5.5.5 Information 159](#_Toc195091456)

[4.5.6 Merchant Portal: User Management View 160](#_Toc195091457)

[4.5.6.1 Profile Tab – Create Users 160](#_Toc195091458)

[4.5.6.1.1 API definition and implementations 160](#_Toc195091459)

[4.5.6.1.2 Validation 162](#_Toc195091460)

[4.5.6.1.3 Business Logic 163](#_Toc195091461)

[4.5.6.1.4 Database table – MERCHANT\_USER 163](#_Toc195091462)

[4.5.6.1.5 Information 163](#_Toc195091463)

[4.5.6.2 Profile Tab Update Users 164](#_Toc195091464)

[4.5.6.2.1 API definition and Implementations 164](#_Toc195091465)

[4.5.6.2.2 Validation 166](#_Toc195091466)

[4.5.6.2.3 Business Logic 167](#_Toc195091467)

[4.5.6.2.4 Database Table – MERCHANT\_USER 167](#_Toc195091468)

[4.5.6.2.5 Information 167](#_Toc195091469)

[4.5.7 Manage Users 168](#_Toc195091470)

[4.5.7.1 API definition and Implementations 168](#_Toc195091471)

[4.5.7.2 Validation 171](#_Toc195091472)

[4.5.7.2.1 Request Data Validation 171](#_Toc195091473)

[4.5.7.2.2 Business data validation 171](#_Toc195091474)

[4.5.7.2.3 Business Logic 171](#_Toc195091475)

[4.5.7.2.4 Database Table – MERCHANT\_USER 172](#_Toc195091476)

[4.5.7.2.5 Information 172](#_Toc195091477)

[4.5.8 User Status Update 173](#_Toc195091478)

[4.5.8.1 API Definition and Implementations 173](#_Toc195091479)

[4.5.8.2 Validation 175](#_Toc195091480)

[4.5.8.3 Business Logic 175](#_Toc195091481)

[4.5.8.4 Database table – MERCHANT\_USER 176](#_Toc195091482)

[4.5.8.5 Information 176](#_Toc195091483)

[4.5.9 Assign - Merchant User and MIDs mapping 177](#_Toc195091484)

[4.5.9.1 API definition and Implementations 177](#_Toc195091485)

[4.5.9.2 Validation 178](#_Toc195091486)

[4.5.9.2.1 Request Data Validation 178](#_Toc195091487)

[4.5.9.2.2 Business data validation 178](#_Toc195091488)

[4.5.9.2.3 Business Logic 178](#_Toc195091489)

[4.5.9.2.4 Database table – MERCHANT\_INFO 179](#_Toc195091490)

[4.5.9.2.5 Information 179](#_Toc195091491)

[4.5.10 Get - Merchant User and MIDs Mapping 180](#_Toc195091492)

[4.5.10.1 API Definition and Implementations 180](#_Toc195091493)

[4.5.10.2 Validation 181](#_Toc195091494)

[4.5.10.2.1 Request Data Validation 181](#_Toc195091495)

[4.5.10.2.2 Business logic 181](#_Toc195091496)

[4.5.10.2.3 Database table – MERCHANT\_INFO 182](#_Toc195091497)

[4.5.10.2.4 Information 182](#_Toc195091498)

[4.5.11 Merchant Portal – Get Merchant User Roles 183](#_Toc195091499)

[4.5.11.1 API definition and implementation 183](#_Toc195091500)

[4.5.11.1.1 Validation 184](#_Toc195091501)

[4.5.11.1.2 Business logic 184](#_Toc195091502)

[4.5.11.1.3 Information 184](#_Toc195091503)

[4.5.11.1.4 Database table – USER\_ROLES 185](#_Toc195091504)

[4.5.12 Merchant Portal – Get Users List for Merchant Identifier. 185](#_Toc195091505)

[4.5.12.1 API definition and implementation 185](#_Toc195091506)

[4.5.12.2 Validation 187](#_Toc195091507)

[4.5.12.2.1 Request Data Validation 187](#_Toc195091508)

[4.5.12.2.2 Business data validation 187](#_Toc195091509)

[4.5.12.2.3 Business logic 187](#_Toc195091510)

[4.5.12.2.4 Information 187](#_Toc195091511)

[4.5.12.2.5 Database table – MERCHANT\_USER 188](#_Toc195091512)

[4.5.13 Merchant Portal – Merchant Access Info 188](#_Toc195091513)

[4.5.13.1 API implementation 188](#_Toc195091514)

[4.5.13.2 Validation 189](#_Toc195091515)

[4.5.13.2.1 Business logic 190](#_Toc195091516)

[4.5.13.2.2 Information 190](#_Toc195091517)

[4.5.13.2.3 Database table MERCHANT\_INFO 191](#_Toc195091518)

[4.6 Account and Settings 191](#_Toc195091519)

[4.6.1 API Key and Encryption Key Management 192](#_Toc195091520)

[4.6.1.1 Class diagram 192](#_Toc195091521)

[4.6.1.2 Generate API and encryption key for Merchant 192](#_Toc195091522)

[4.6.1.2.1 API definition and implementations 192](#_Toc195091523)

[4.6.1.2.2 Validation 194](#_Toc195091524)

[4.6.1.2.3 Business Logic 195](#_Toc195091525)

[4.6.1.2.4 Database table - MERCHANT\_KEY\_MANAGEMENT 195](#_Toc195091526)

[4.6.1.2.5 Information 195](#_Toc195091527)

[4.6.1.3 Validate API key for Merchant 196](#_Toc195091528)

[4.6.1.3.1 API definition and implementations 196](#_Toc195091529)

[4.6.1.3.2 Validation 197](#_Toc195091530)

[4.6.1.3.3 Business Logic 198](#_Toc195091531)

[4.6.1.3.4 Database table - NA 198](#_Toc195091532)

[4.6.1.3.5 Information 198](#_Toc195091533)

[4.6.1.4 Get Merchant Encryption Keys 198](#_Toc195091534)

[4.6.1.4.1 API definition and implementations 198](#_Toc195091535)

[4.6.1.4.2 Validation 200](#_Toc195091536)

[4.6.1.4.3 Business Logic 200](#_Toc195091537)

[4.6.1.4.4 Database table - NA 200](#_Toc195091538)

[4.6.1.4.5 Information 200](#_Toc195091539)

[4.6.1.5 List of keys for a Merchant 200](#_Toc195091540)

[4.6.1.5.1 API definition and implementations 200](#_Toc195091541)

[4.6.1.5.2 Validation - None 207](#_Toc195091542)

[4.6.1.5.3 Business Logic 207](#_Toc195091543)

[4.6.1.5.4 Database table - NA 207](#_Toc195091544)

[4.6.1.5.5 Information 207](#_Toc195091545)

[4.6.1.6 Download keys for a merchant 207](#_Toc195091546)

[4.6.1.6.1 API definition and implementations 207](#_Toc195091547)

[4.6.1.6.2 Validation 208](#_Toc195091548)

[4.6.1.6.3 Business Logic 208](#_Toc195091549)

[4.6.1.6.4 Database table - NA 208](#_Toc195091550)

[4.6.1.6.5 Information 208](#_Toc195091551)

[4.6.2 Payment Page Theme 209](#_Toc195091552)

[4.6.2.1 Class diagram 209](#_Toc195091553)

[4.6.2.2 Create configured page theme 209](#_Toc195091554)

[4.6.2.2.1 API Definition and implementation 209](#_Toc195091555)

[4.6.2.2.2 Validation 210](#_Toc195091556)

[4.6.2.2.3 Business logic 210](#_Toc195091557)

[4.6.2.2.4 Database table: THEME 211](#_Toc195091558)

[4.6.2.2.5 Information 211](#_Toc195091559)

[4.6.2.3 Update Configured Page Theme 211](#_Toc195091560)

[4.6.2.3.1 API Definition and implementation 211](#_Toc195091561)

[4.6.2.3.2 Validation 214](#_Toc195091562)

[4.6.2.3.3 Business logic 214](#_Toc195091563)

[4.6.2.3.4 Database table: THEME 214](#_Toc195091564)

[4.6.2.3.5 Information 214](#_Toc195091565)

[4.6.2.4 Get Payment theme 215](#_Toc195091566)

[4.6.2.4.1 API Definition and implementation 215](#_Toc195091567)

[4.6.2.4.2 Validation 217](#_Toc195091568)

[4.6.2.4.3 Business logic 217](#_Toc195091569)

[4.6.2.4.4 Database Table: THEME 218](#_Toc195091570)

[4.6.2.4.5 Information 218](#_Toc195091571)

[4.6.2.5 Update Payment Page Theme 218](#_Toc195091572)

[4.6.2.5.1 API Definition and implementation 218](#_Toc195091573)

[4.6.2.5.2 Validation 219](#_Toc195091574)

[4.6.2.5.3 Business logic 219](#_Toc195091575)

[4.6.2.5.4 Database Table: THEME 220](#_Toc195091576)

[4.6.2.5.5 Information 220](#_Toc195091577)

[4.6.2.6 Reset Payment Page Theme 220](#_Toc195091578)

[4.6.2.6.1 API Definition and implementation 220](#_Toc195091579)

[4.6.2.6.2 Validation 221](#_Toc195091580)

[4.6.2.6.3 Business logic 221](#_Toc195091581)

[4.6.2.6.4 Database Table: THEME 222](#_Toc195091582)

[4.6.2.6.5 Information 222](#_Toc195091583)

[4.6.2.7 Get Payment Page Theme 222](#_Toc195091584)

[4.6.2.7.1 API Definition and implementation 222](#_Toc195091585)

[4.6.2.7.2 Validation 224](#_Toc195091586)

[4.6.2.7.3 Business logic 225](#_Toc195091587)

[4.6.2.7.4 Database Table: THEME 225](#_Toc195091588)

[4.6.2.7.5 Information 225](#_Toc195091589)

[4.6.3 Merchant Portal – Bank Accounts 226](#_Toc195091590)

[4.6.3.1 Class diagram 226](#_Toc195091591)

[4.6.3.2 Bank Accounts Listing 226](#_Toc195091592)

[4.6.3.2.1 API Definition and implementation 226](#_Toc195091593)

[4.6.3.2.2 Validation 228](#_Toc195091594)

[4.6.3.2.3 Business logic 228](#_Toc195091595)

[4.6.3.2.4 Database table – MERCHANT\_BANK\_ACCOUNT 229](#_Toc195091596)

[4.6.3.2.5 Information 229](#_Toc195091597)

[4.6.3.3 Bank accounts add / link 229](#_Toc195091598)

[4.6.3.3.1 API Definition and implementation 229](#_Toc195091599)

[4.6.3.3.2 Validation 231](#_Toc195091600)

[4.6.3.3.3 Business Logic 232](#_Toc195091601)

[4.6.3.3.4 Database Table – MERCHANT\_BANK\_ACCOUNT 232](#_Toc195091602)

[4.6.3.3.5 Information 233](#_Toc195091603)

[4.6.3.4 Bank Accounts Update 233](#_Toc195091604)

[4.6.3.4.1 API Definition and implementation 233](#_Toc195091605)

[4.6.3.4.2 Validation 235](#_Toc195091606)

[4.6.3.4.3 Business logic 235](#_Toc195091607)

[4.6.3.4.4 Database Table – MERCHANT\_BANK\_ACCOUNT 236](#_Toc195091608)

[4.6.3.4.5 Information 236](#_Toc195091609)

[4.7 Help & Support 236](#_Toc195091610)

[4.7.1 Get Help & Support 237](#_Toc195091611)

[4.7.1.1 Class diagram 237](#_Toc195091612)

[4.7.1.2 API Definition and implementation 237](#_Toc195091613)

[4.7.1.2.1 Validation 239](#_Toc195091614)

[4.7.1.2.2 Business logic 239](#_Toc195091615)

[4.7.1.2.3 Database Table – HELP\_SUPPORT\_TEAM 240](#_Toc195091616)

[4.7.1.2.4 Information 240](#_Toc195091617)

[4.7.2 Merchant Portal - Footer info 240](#_Toc195091618)

[4.7.2.1 Class diagram 240](#_Toc195091619)

[4.7.2.2 API Definition and implementation 241](#_Toc195091620)

[4.7.2.2.1 Validation 242](#_Toc195091621)

[4.7.2.2.2 Business logic 243](#_Toc195091622)

[4.7.2.2.3 Database Table - FOOTER 243](#_Toc195091623)

[4.7.2.2.4 Information 243](#_Toc195091624)

[5. SD\_MS – SBI Admin Portal – Integration 243](#_Toc195091625)

[5.1 Class diagram 243](#_Toc195091627)

[5.2 Merchant Portal – Merchant Admin Entity Creation 244](#_Toc195091628)

[5.2.1 API Definition and implementation 244](#_Toc195091629)

[5.2.1.1 Validation 245](#_Toc195091630)

[5.2.1.2 Business logic 245](#_Toc195091631)

[5.2.1.3 Database Table: MERCHANT\_ENTITY\_USER 246](#_Toc195091632)

[5.2.1.4 Information 246](#_Toc195091633)

[5.3 Merchant Portal – Admin Onboarding 246](#_Toc195091634)

[5.3.1 API Definition and implementation 246](#_Toc195091635)

[5.3.1.1 Validation 256](#_Toc195091636)

[5.3.1.2 Business L 256](#_Toc195091637)

[5.3.1.3 Database Table - MERCHANT\_USER and MERCHANT\_INFO 257](#_Toc195091638)

[5.3.1.4 Information 257](#_Toc195091639)

[5.4 Merchant Portal: Account Setting - Admin User Entity Mapping 258](#_Toc195091640)

[5.4.1 API Definition and implementation 258](#_Toc195091641)

[5.4.1.1 Validation 259](#_Toc195091642)

[5.4.1.1.1 Request Data Validation 259](#_Toc195091643)

[5.4.1.2 Business Logic 259](#_Toc195091644)

[5.4.1.3 Database Table - MERCHANT\_ENTITY\_USER 260](#_Toc195091645)

[5.4.1.4 Information - None 260](#_Toc195091646)

[5.5 Merchant Portal - List of User for Merchant for SBI Admin 260](#_Toc195091647)

[5.5.1 API Definition and Implementation 260](#_Toc195091648)

[5.5.2 Validation 262](#_Toc195091649)

[5.5.3 Business Logic 263](#_Toc195091650)

[5.5.4 Database Table – MERCHANT\_USER 263](#_Toc195091651)

[5.5.4.1 Information 263](#_Toc195091652)

[5.6 Update Help & Support 264](#_Toc195091653)

[5.6.1 API Definition and Implementation 264](#_Toc195091654)

[5.6.2 Validation 266](#_Toc195091655)

[5.6.3 Business logic 266](#_Toc195091656)

[5.6.4 Database Table - HELP\_SUPPORT\_TEAM 267](#_Toc195091657)

[5.6.5 Information 267](#_Toc195091658)

[5.7 Merchant Portal - Footer 267](#_Toc195091659)

[5.7.1 API Definition and Implementation 267](#_Toc195091660)

[5.7.2 Validation 269](#_Toc195091661)

[5.7.3 Business Logic 269](#_Toc195091662)

[5.7.4 Database Table – FOOTER 270](#_Toc195091663)

[5.7.5 Information 270](#_Toc195091664)

[5.8 Merchant Portal – Bank Accounts Listing 270](#_Toc195091665)

[5.8.1 API Definition and Implementation 270](#_Toc195091666)

[5.8.2 Validation 272](#_Toc195091667)

[5.8.3 Business Logic 272](#_Toc195091668)

[5.8.4 Database Table - MERCHANT\_BANK\_ACCOUNT 273](#_Toc195091669)

[5.8.5 Information 273](#_Toc195091670)

[6. Scheduler Service 273](#_Toc195091671)

[6.1 Password Expiry Scheduler 273](#_Toc195091673)

[6.1.1 Information 274](#_Toc195091674)

[6.2 Account Expiration Scheduler 274](#_Toc195091675)

[6.2.1 Business Logic 274](#_Toc195091676)

[6.2.2 Information 274](#_Toc195091677)

[7. Admin Services 275](#_Toc195091678)

[7.1 Merchant – Current Downtime for Paymodes 275](#_Toc195091680)

[7.1.1 API Implementation 275](#_Toc195091681)

[7.1.2 Validation 276](#_Toc195091682)

[7.1.3 Business Logic 276](#_Toc195091683)

[7.2 Merchant - Paymode Details 276](#_Toc195091684)

[7.2.1 API Implementation 277](#_Toc195091685)

[7.2.2 Validation 283](#_Toc195091686)

[7.2.3 Business Logic 284](#_Toc195091687)

[7.3 Merchant – Notification Details 284](#_Toc195091688)

[7.3.1 API Implementation 284](#_Toc195091689)

[7.3.2 Validation 286](#_Toc195091690)

[7.3.3 Business Logic 286](#_Toc195091691)

[7.4 Merchant – Save Paymode Details 286](#_Toc195091692)

[7.4.1 API Implementation 286](#_Toc195091693)

[7.4.2 Validation - None 289](#_Toc195091694)

[7.4.3 Business Logic 289](#_Toc195091695)

[8. Key Management Service 289](#_Toc195091696)

[8.1 Generates API Key ID and Key Secret 289](#_Toc195091698)

[8.1.1 API Implementation 290](#_Toc195091699)

[8.1.2 Validation 291](#_Toc195091700)

[8.1.3 Business Logic 291](#_Toc195091701)

[8.1.4 Information 291](#_Toc195091702)

[8.2 Generates Encryption Keys for a given Merchant 291](#_Toc195091703)

[8.2.1 API Implementation 291](#_Toc195091704)

[8.2.2 Validation 292](#_Toc195091705)

[8.2.3 Business Logic 292](#_Toc195091706)

[8.2.4 Information 293](#_Toc195091707)

[8.3 Generates Encryption Keys for a given Merchant 293](#_Toc195091708)

[8.3.1 API Implementation 293](#_Toc195091709)

[8.3.2 Validation 294](#_Toc195091710)

[8.3.3 Business logic 294](#_Toc195091711)

[8.3.4 Information 294](#_Toc195091712)

[8.4 Get Encryption Keys for a given Merchant ID 294](#_Toc195091713)

[8.4.1 API Implementation 294](#_Toc195091714)

[8.4.2 Validation: No 295](#_Toc195091715)

[8.4.3 Business Logic 295](#_Toc195091716)

[8.4.4 Information 295](#_Toc195091717)

[8.5 Download Encryption Keys 295](#_Toc195091718)

[8.5.1 API Implementation 295](#_Toc195091719)

[8.5.2 Validation: No 296](#_Toc195091720)

[8.5.3 Business Logic 296](#_Toc195091721)

[8.5.4 Information 296](#_Toc195091722)

[8.6 Fetches list of Keys 296](#_Toc195091723)

[8.6.1 API implementation 296](#_Toc195091724)

[8.6.2 Validation: No 297](#_Toc195091725)

[8.6.3 Business Logic 297](#_Toc195091726)

[8.6.4 Information 297](#_Toc195091727)

[9. Bank Service 297](#_Toc195091728)

[9.1 List active banks 297](#_Toc195091730)

[9.1.1 API Implementation 298](#_Toc195091731)

[9.1.2 Validation - None 300](#_Toc195091732)

[9.1.3 Business Logic 300](#_Toc195091733)

[9.2 Get all Branches 300](#_Toc195091734)

[9.2.1 API Implementation 300](#_Toc195091735)

[9.2.2 Business Logic 301](#_Toc195091736)

[9.3 Get bank details 301](#_Toc195091737)

[9.3.1 API Implementation 302](#_Toc195091738)

[9.3.2 Validation 303](#_Toc195091739)

[9.3.3 Business Logic 304](#_Toc195091740)

[9.4 Validate IFSC 304](#_Toc195091741)

[9.4.1 API Implementation 304](#_Toc195091742)

[9.4.2 Validation 305](#_Toc195091743)

[9.4.3 Business Logic 305](#_Toc195091744)

# Document Purpose

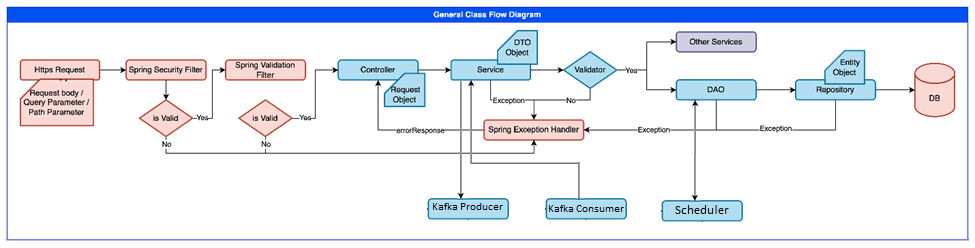
The purpose of this document is to outline the design of the Merchant Portal. This will include a view of the high-level architecture and the breakdown of the internal subsystems.

# Technical Design

## 2.1 Technology Stack

|  |  |
| --- | --- |
| Dependencies | Version |
| Java | 21.x.x |
| Gradle | 8.9 |
| Spring Boot | 3.3.x |
| Spring Schedular | 3.3.x |
| Spring Security | 6.3.x |
| Swagger (API Documentation) | 3.1.x |
| JUnit | 5 |
| Gemfire | 2.x |
| Liquibase | 4.29 |
| Slf4j | 1.7.x |
| Oracle Driver | 19.x |
| Apache Kafka | 2.7.X |

## 2.2 Generic Class Flow



## 2.3 Generic Header

|  |
| --- |
| {  [HTTP METHOD TYPE] [API\_END\_POINT]  HTTP/1.1 Host: [HOST NAME]  Authorization: Bearer <your-access-token>  API-Key: <your-api-key>  Content-Type: application/json  Accept: application/json  User-Agent: [CLIENT\_DETAILS]  Request-ID: [Unique identifier for request to track]  Origin: [Request Originator]  } |

## 2.4 Generic Request Body

|  |
| --- |
| {  "fieldName":"value",  } |

## 2.5 Generic Response Body

|  |
| --- |
| {  "data": [GENERIC\_RESPONSE\_OBJECT],  “count”: Total Record Count for data Object,  “total”: Total Record Count in current data Object,  "errors": [  {  "errorCode": "ERROR\_CODE",  "errorMessage":"ERROR\_MSG"  },  {  "errorCode": "ERROR\_CODE",  "errorMessage": "ERROR\_MSG"  }  ],  "status": "STATUS\_CODE"  } |

## 2.6 Error Codes

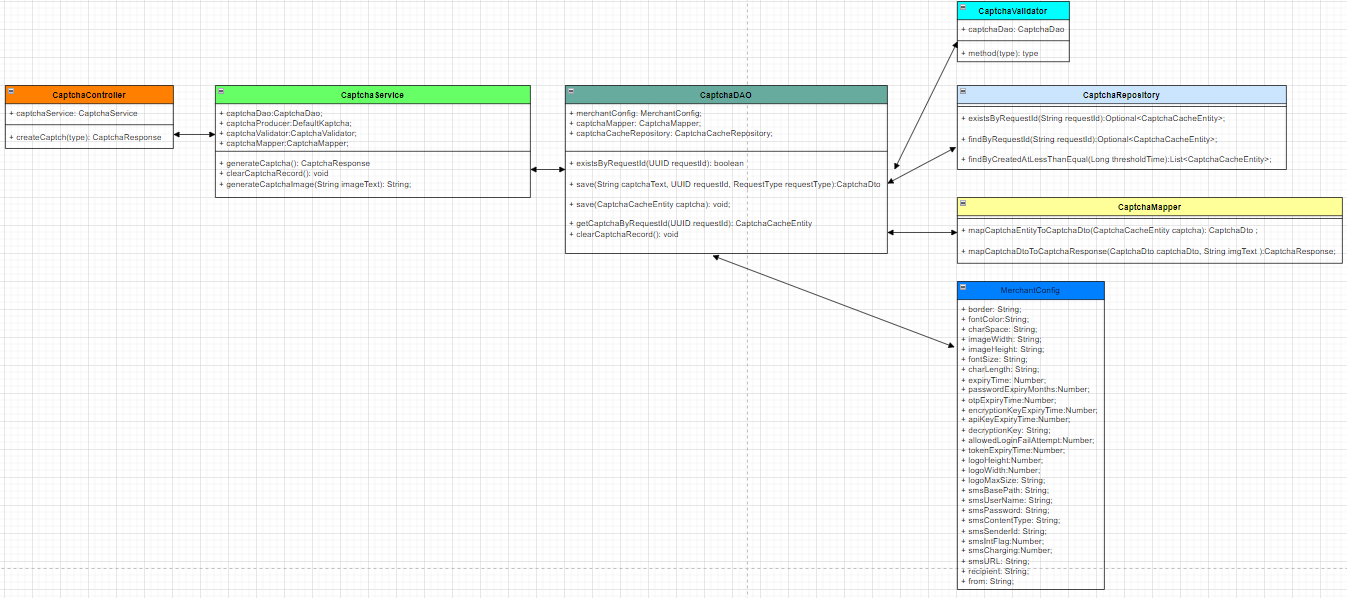
|  |  |
| --- | --- |
| Error Code | Error Description |
| 7001 | {field Name} is mandatory. |
| 7002 | {field Name} is invalid. {Reason} |
| 7003 | {field Name} is not found. |
| 7004 | {field Name} is already present. |
| 7005 | {field Name} is not active. |
| 7006 | {field Name} state not accurate. |
| 7031 | {field Name} generation failed. |
| 7032 | Something wrong |
| 7033 | {field Name} is expired |

# 3. SD\_MS 1 – Merchant Portal – Login

The Login functionality is used to deliver a secure and seamless user experience for accessing the Merchant Portal. This section presents the backend technical design for the core features including login, password reset, OTP handling, and supplementary mechanisms such as password expiry management, user validation, and captcha verification. The design emphasizes security, reliability, and ease-of-use to support a robust authentication process.

## Captcha

### Class Flow Diagram



#### API Definition / Implementation

* **Endpoint**: /merchant/v1/captcha/generate
* **Method**: POST
* **Description:** This API generates a captcha for the client to display during login or other actions requiring validation. The captcha ensures an additional layer of security by preventing automated login attempts.
* **Request Details**
  + **Content-Type**: application/json
  + **Request Object**

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Description** | **Mandatory** | **Data Type** |
| requestId | Unique identifier for the request. | Yes | UUID |
| requestType | The context for captcha generation. Ex (LOGIN, RESET\_PASSWORD, CHANGE\_PASSWORD) | Yes | string |

**Example Request**

|  |
| --- |
| {  "requestId": "57882ab1-b884-49cd-91d0-35199a44f880",  "requestType": "LOGIN" } |

**Response:**

* + **Content-Type**: application/json
  + **Response Object**

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Description** | **Mandatory** | **Data Type** |
| requestId | Same which is in request | Yes | UUID |
| captchaText | Captcha hash text | Yes | string |
| expiryTime | Captcha will be expired on this given time | Yes | long |
| requestType | Same which is in request body. | Yes | string |

**Example Response:**

|  |
| --- |
| {  “requestId": "57882ab1-b884-49cd-91d0-35199a44f880",  "captchaImage": "base64\_encoded\_image\_string",  " expiryTime": 1735614452395,  "requestType": "LOGIN" captchaAudio": "UklGRpS3AgBXQVZFZm10IBAAAAABAAEAgD4AAAB9AAACABAAZGF0YXC3AgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAD4  } |

#### Validation

##### Request Data Validation

* **RequestId**

The field must be unique for each request.

* **RequestType**

The field must contain a valid type, such as login, reset\_ password.

##### Business Validation

* Each captcha request must be generated uniquely for every individual request. The requestId should correspond to a single captcha challenge and should not be reused across multiple requests.
* The captcha solution must remain valid only for a predefined period, for example, five minutes. After this time, the captcha will expire, and the user will be required to request a new one.
* Duplicate requestId must not be allowed. If a request with the same requestId is received, it will be rejected.
* Captcha details including hash text will be stored in GemFire and will get deleted after predefined period.

#### Business Logic

* **Step-1: Captcha controller**

Request reaches controller and then calls the captcha service.

* **Step-2:** **Captcha service**

Calls the validator to check for request type andid and then generates the captcha image.

* **Step-3:** **Captcha validator**

Validates for the mandatory fields, i.e., requestId and requesttype. Also checks for the uniqueness of the requestId.

* **Step-4: Captcha service**

On successful validation, generates captcha text and image.

* **Step-5: Captcha DAO**

Calls the cache repository to save the captcha image, expiry time, and captcha Id in cache.

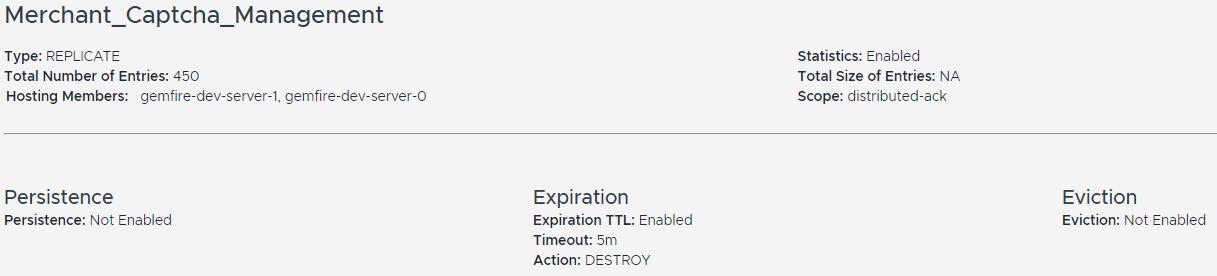
* **Step-6: Captcha Cache Repository**

Save captcha information in GemFire cache.

* **Step-7: Captcha service**

Response back with captcha image, expiry time and captcha Id.

#### GemFire cache region – Merchant\_Captcha\_Management



#### Information

* Log Request

Log all incoming requests, including headers and body parameters (excluding sensitive information) to the database.

* Log Response

Log all responses, including generated token details and any error messages except the token.

### User Validation

#### API Definition / Implementation

* **Endpoint:** /merchant/v1/validation/user
* **Method:** POST
* **Description:** This API does the user validation during login or other actions that require a validation. If a valid user exists, this API display the user details, otherwise an error message is displayed.
* **Request Details:**
* **Content-Type:** application/json

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Description** | **Mandatory** | **Data Type** |
| userName | Merchant userName / emailId / mobile | Yes | string |
| requestType | The context for login type request. | Yes | string |

**Example request:**

{

"requestType": "LOGIN”,

" userName": "mer\_user\_01”

}

* **Response:** Success: 200 – Ok

{

"status": 1,

"data": [

"User validated Successfully"

]

}

* **Response:** Error: 200 – Ok

|  |
| --- |
| {  "errors": [  {  "errorCode":"ERROR\_CODE ",  "errorMessage":"ERROR\_MSG",  },  ],  "status":0  } |

#### Validation

* **username:** Merchant user id or email id or mobile number.
* **requestType:** The field must contain a valid type, such as login.
* Validate user provided user id / email / mobile number with merchant user data.
* Validate **requestType** values
* If merchant user has isFirstLogin as true then send validation, user needs to change the password.

#### Business logic

**Step 1: Validation controller**

Request reaches validation controller and calls the validation service.

**Step 2: Validation service**

Service validates request for mandatory fields, calls the DAO to check user.

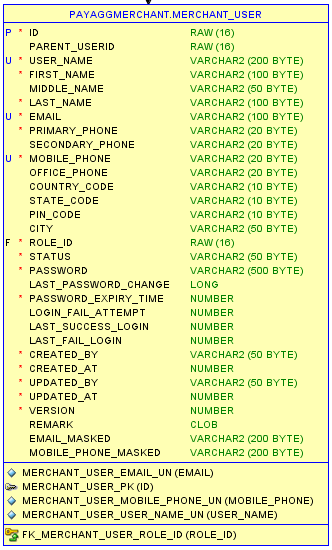
**Step 3: Validation DAO**

Check if user exists, then return value should be boolean.

**Step 4: Validation service**

For existing valid user, send success response otherwise send custom exception.

#### Database Table – MERCHNT\_USER

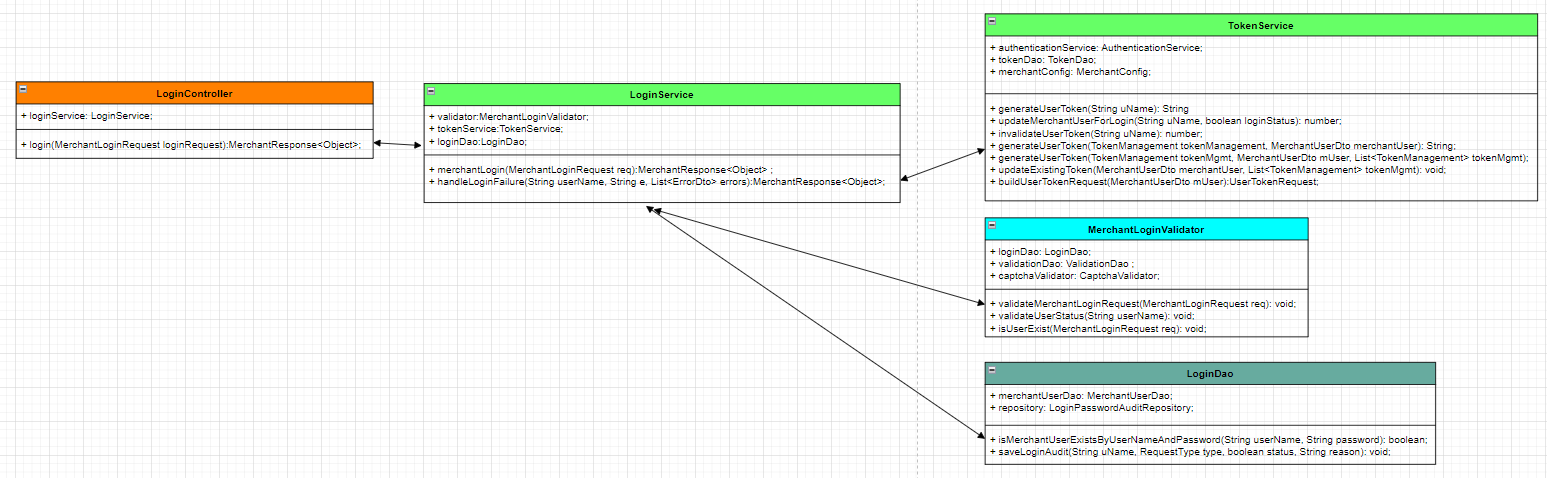


#### Information

* If valid, extract user details and then set them in the security context.
* If invalid, reject the request with an unauthorized response.
* Log Request
* Log all incoming requests, including headers and body parameters (excluding sensitive information) to them.
* Log Response:
* Log all responses, including generated token details and any error messages except the token.

### Merchant Login

#### Class Diagram



#### API Definition / Implementation

* **Endpoint:** /merchant/v1/login
* **Method:** POST
* **Description:** This API allows a merchant user to log in. It validates the merchant details such as user Id, email Id, and phone number. Upon successful validation, the merchant user is allowed to log in, otherwise the account is locked after multiple unsuccessful attempts.
* **Request Details:**
* **Content-Type:** application/json
* **Request Object**

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Description** | **Mandatory** | **Data Type** |
| userName | Merchant username or email id or mobile | Yes | string |
| password | Hashed password. | Yes | string |
| requestId | Same as during captcha request | Yes | string |
| captchaText | captcha Hashed Text | Yes | string |

* **Example request**

{

"userName": "SBIEPAY",

"password": "6dWTsEgPZ2bpCIQqITJY3sZ6WrRTtuNYvWGf5R9ZVwzfhSj4Kiw/n8q3WXdFM/Gpd6whChRkvh+8BV0jpz1WMA==",

"requestId": "30529ddf-ce3f-4547-a4c4-d741c893b81e",

"captchaText": "thE85w+Ua893+/HBOanEBbFA84LZhSN7nnhBtSy0BwP5x4gcqu6/Fgh5j3tuuTlXXSzIPmx0c6tjE+5IramRRA=="

}

* **Response: Success: 200 – Ok**

{  
 "data": [  
 {  
 "message": ""Login User Found Successfully”  
 }  
 ],  
 "status":1,  
}

* **Response:** Error: 200 – Ok

{

"errors": [

{

"errorCode":"ERROR\_CODE ",

"errorMessage":"ERROR\_MSG"

},

],

"status":0

}

|  |  |
| --- | --- |
| **Error Code** | **Error Description** |
| 7001 | Username / Request Type / Request id is Mandatory |
| 7002 | Invalid merchant user. Reason: User is Inactive. |
| 7003 | Invalid User -> Please check Username and Password. |

#### Validation

* **userName** - Validates merchant user id or email id or mobile number.
* **password –** Encrypted hashed password.
* **captchaText –** Check if value is blank.
* **requestId -** The field must contain a valid type, such as login, reset\_ password.
* Validate user provided user id / email id / mobile number with merchant user data.
* Validate **requestType** values, it should be LOGIN.
* Validate fail count and mark the user as LOCKED after multiple unsuccessful attempts.

#### Business Logic

* **Step 1: Login controller**

Request reaches login controller and calls the login service.

* **Step 2: Login service**

Service validates request for the mandatory fields and then calls DAO to check user.

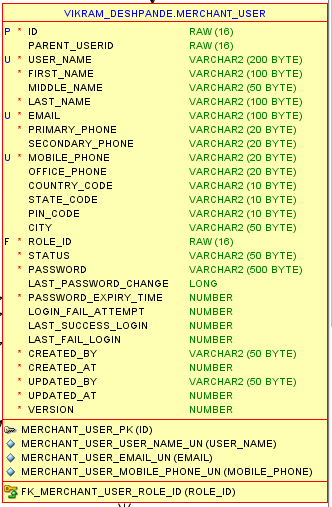
* **Step 3: Login DAO**

Check for the user existence, captcha expiry, and captcha string.

* **Step 4: Login service**

For existing valid user, send success response otherwise send custom exception.

#### Database Table – MERCHANT\_USER



#### Information

* + If valid, extract user details and then set them in the security context.
  + If invalid, reject the request with an unauthorized response.
  + Log Request

Log all incoming requests, including headers and body parameters (excluding sensitive information) to the database.

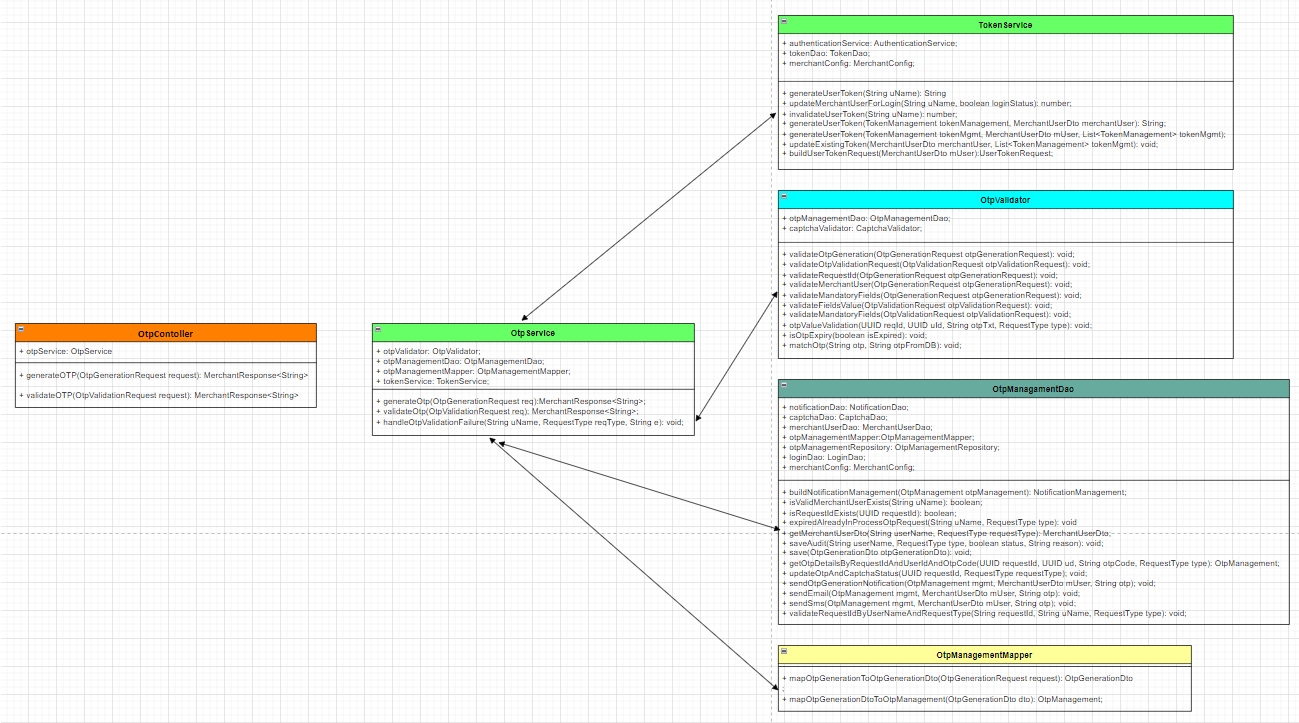
* + Log Response

Log all responses, including generated token details and any error messages except the token.

* + Manage audits in DB using MERCHANT\_USER\_AUDIT & LOGIN\_PASSWORD\_AUDIT tables.

### OTP Generation

#### Class Diagram



#### API Definition / Implementation

* **Endpoint**: /merchant/v1/otp/generation
* **Method**: POST
* **Description:** This API generates an OTP for the client to display during login or other actions requiring validation. The OTP ensures an additional layer of security by preventing automated login attempts. Upon successful validation of the mandatory details including user name, requestId, and requestType, an OTP is generated, otherwise an error message is displayed.
* **Request Details**
  + **Content-Type**: application/json
  + **Request Object**

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Description** | **Mandatory** | **Data Type** |
| userName | Merchant user id. | Yes | string |
| requestType | The context for login generation. | Yes | string |
| requestId | Unique request id | Yes | string |

* + **Example Request**

{

"userName": " merchantUser01",

“requestId”: “UUID”,  
 "requestType": "login"  
}

* **Response**
  + **Content-Type**: application/json
  + **Response Object**

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Description** | **Mandatory** | **Data Type** |
| otpId | Unique identifier for the OTP generated. | Yes | string |
| otp | Hashed value for generated OTP. | Yes | string |
| requestId | Echo of the requestId from the client request. | Yes | string |

* + **Example Response – Success 200 Ok**

{

"status": 1,

"data": [ "Otp Generated Successfully"]

}

* + **Example request – error 200 Ok**

{

"status":0,

"error": [

{

"errorCode": String,

"errorMessage": String

}

]

}

|  |  |
| --- | --- |
| Error Code | Error Description |
| 7001 | User name / Request Type / RequestId is Mandatory. |

#### Validation

##### Request Data Validation

* **Username:** The field value must be a valid merchant user id or email or mobile number.
* **RequestType:** The field must contain a valid type, such as login, reset\_ password.
* **RequestId:** The field must contain a unique request id.

##### Business Validation

#### Business Logic

* **Step1: OTP controller**

Request reaches OTP controller and calls the OTP service.

* **Step 2: OTP service**

Service validates OTP, and then calls DAO to save OTP.

* **Step 3: OTP validator**

Validate OTPrequest for mandatory fields and uniqueness of OTP.

* **Step 4: OTP service**

On successful validation generate OTP.

* **Step 5: OTP DAO**

Call GemFire repository to save OTP code, expiry and id.

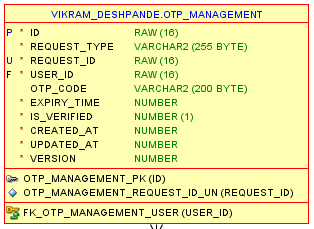
* **Step 6: OTP Repository**

Save OTP information.

* **Step 7: OTP service**

Save Notification and send success response.

#### GemFire cache region – OTP\_MANAGEMENT



#### Information

* If valid, extract user details and then set them in the security context.
* If invalid, reject the request with an unauthorized response.
* Audit all login success & failed attempt, along with error messages.
* Log Request

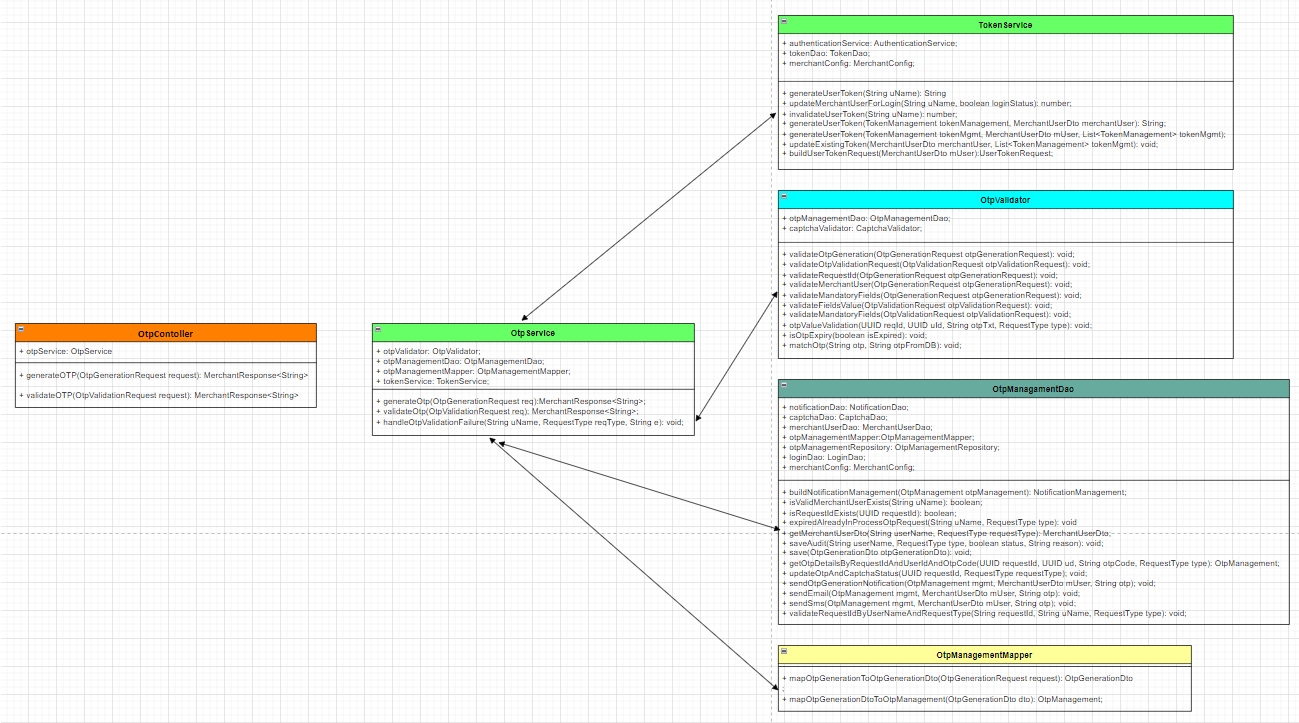
Log all incoming requests, including headers and body parameters (excluding sensitive information) to the database.

* Log Response

Log all responses, including generated token details and any error messages except the token.

### OTP Regeneration

#### Class Diagram



#### API Definition / Implementation

* **Endpoint**: /merchant/v1/otp/regeneration
* **Method**: POST
* **Description:** This API regenerates an OTP for the client to display during login or other actions requiring validation. The OTP ensures an additional layer of security by preventing automated login attempts. Upon successful validation of the mandatory details including user name, requestId, and requestType, an OTP is generated, otherwise an error message is displayed.
* **Request Details**
  + **Content-Type**: application/json
  + **Request Object**

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Description** | **Mandatory** | **Data Type** |
| userName | Merchant user id. | Yes | string |
| requestType | The context for login generation. | Yes | string |
| requestId | Unique request id | Yes | string |

* + **Example Request**

{

"userName": " merchantUser01",

“requestId”: “UUID”,  
 "requestType": "login"  
}

* **Response**
  + **Content-Type**: application/json
  + **Response Object**

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Description** | **Mandatory** | **Data Type** |
| otpId | Unique identifier for the OTP generated. | Yes | string |
| otp | Hashed value for generated OTP. | Yes | string |
| requestId | Echo of the requestId from the client request. | Yes | string |

* + **Example Response – Success 200 Ok**

{

"status": 1,

"data": [ "Otp Generated successsfully"]

}

* + Example request – error 200 Ok

{

"status":0,

"error": [

{

"errorCode": String,

"errorMessage": String

}

]

}

|  |  |
| --- | --- |
| Error Code | Error Description |
| 7001 | User name / Request Type / RequestId is Mandatory. |

#### Validation

* **UserName:** The field value must be a valid merchant user id or email or mobile number.
* **RequestType:** The field must contain a valid type, such as login, reset\_ password.
* **RequestId:** The field must contain a unique request id.

#### Business Logic

* **Step 1: OTP controller**

Request reaches OTP controller and calls the OTP service.

* **Step 2: OTP service**

Service validates OTP, and then calls DAO to save OTP.

* **Step 3: OTP validator**

Validates OTPrequest for mandatory fields, uniqueness of OTP and uniqueness of request id.

* **Step 4: OTP service**

On successful validation generate OTP.

* **Step 5: OTP DAO**

Calls GemFire repository to save OTP code, expiry and id.

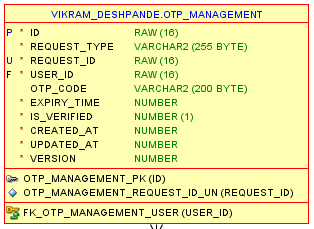
* **Step 6: OTP Repository**

Saves OTP information.

* **Step 7: OTP service**

Saves Notification and send success response.

#### GemFire Cache Region – OTP\_MANAGEMENT



#### Information

* If valid, extract user details and then set them in the security context.
* If invalid, reject the request with an unauthorized response.
* Audit all login success & failed attempt, along with error messages.
* Log Request

Log all incoming requests, including headers and body parameters (excluding sensitive information) to the database.

* Log Response

Log all responses, including generated token details and any error messages except the token.

### OTP Validation

#### API Definition / Implementation

* **Endpoint:** /merchant/v1/otp/validation
* **Method:** POST
* **Description:** This API provides capability to OTP validation for user login or other actions requiring validation. The API validates the mandatory details including user name, requestId, requestType, and OTP hash. The requestId must be unique, otherwise an error message is displayed and the validation is failed.
* **Request details:**
* **Content-Type:** application/json
* **Response object:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Description** | **Mandatory** | **Data Type** |
| otp | Hash value of Entered OTP | Yes | string |
| requestType | The context for login generation. | Yes | string |
| requestId | Request ID which shared on OTP generation | Yes | String |
| userName | The field value will be merchant user idor email or mobile number. | Yes | String |

* + **Example Request**

"userName": "SBIEPAY",

"requestType":"LOGIN",

"requestId": "21529ddf-ce3f-4547-a4c4-d741c893b81e",

"captchaText":"E3Wib1qoQe2p6Pl+y06dh10/cyqTVimJTOvUqoRny43jeSDELT0sWDWEQHSUvzdRdXQw7Hr9gf/r3a7zPpQG0Q==",

"otp":"zshBt9u4cT9it0RuV+FRBqyH2EKJn56RCR/WTqqzYMvh42Ip8VbquQoaMli0g8Ibacdbog8baP5U93Kbr4pDkQ=="

}

* **Example** **response –** Success 200 ok

{

"status": 1,

"data": [

"eyJhbGciOiJIUzUxMiJ9.eyJyb2xlIjpbIkFETUlOIl0sInRva2VuVHlwZSI6IlVTRVIiLCJ1c2VybmFtZSI6IlNCSUVQQVkiLCJzdWIiOiJTQklFUEFZIiwiaXNzIjoiaHR0cHM6Ly9zYmkuZXBheS5jb20iLCJpYXQiOjE3MzY5MjQ0NjUsImV4cCI6MTczNjk0MjQwNX0.PEFyJ5RWpv31vm5\_edMrwnTkyCL333uLjyGKyf5IqVP5T8rdG\_dXFas8TRHQOwZO-TnrB924rMkJH0D-FHR6mA"

]

}

* **Example Response –** Error 200 ok

{

"errors": [

{

"errorCode":"ERROR\_CODE ",

"errorMessage":"ERROR\_MSG"

},

],

"status":0

}

|  |  |
| --- | --- |
| Error Code | Error Description |
| 7003 | Merchant user not found. |
| Respective generic error code and description with field Name | |

#### Validation

##### Request Data Validation

* **userName:** Validate merchant userName or email or mobile number.
* **requestType:** The field must contain a valid type, such as login, reset\_ password.
* **requestId:** Validate merchant unique request id.
* **Otp:** The field must contain OTP hash.

##### Business Validation

* The OTP solution must remain valid only for a predefined period, for example 5 minutes. After this time, the OTP will be deleted from the GemFire region, and the user has to request for a new one.
* Duplicate requestId values must not be allowed. If a request with the same requestId is received, it will be rejected.

#### Business Logic

* **Step 1: OTP controller**

Request reaches OTP controller and calls the OTP service.

* **Step 2: OTP service**

Service validates request for mandatory fields, and then calls the DAO to check user.

* **Step 3: OTP validator**

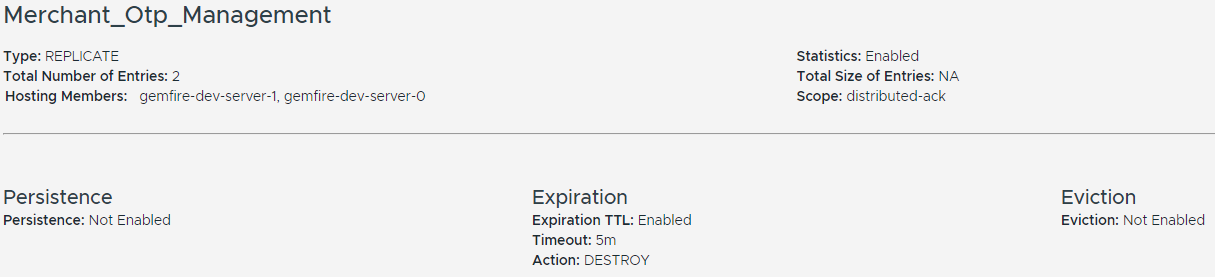
Validates OTP.

* **Step 4: OTP service**

For existing valid user, send success response with JWT token otherwise send custom exception.

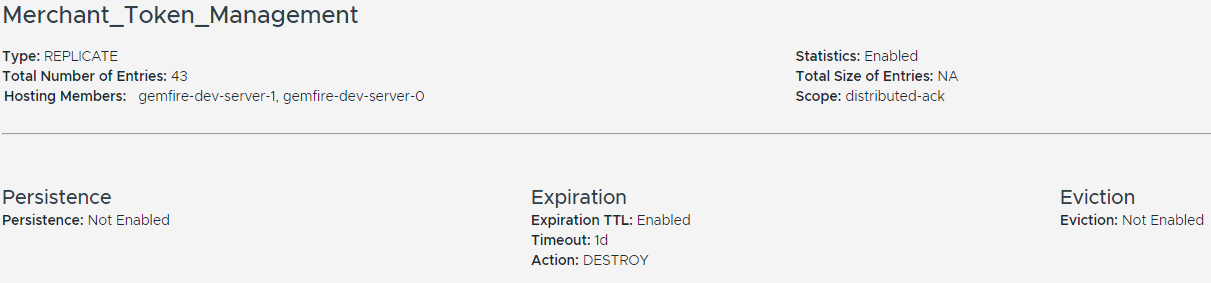
##### GEMFIRE Region – Merchant\_OTP\_MANAGEMENT

GEMFIRE Region – Merchant\_OTP\_MANAGEMENT region should be created:



##### GEMFIRE Region – Merchant\_Token\_MANAGEMENT

GEMFIRE Region – Merchant\_Token\_MANAGEMENT region should be present to store the token and delete after its predefined expiry time.



#### Information

* If valid, extract user details and then set them in the security context.
* If invalid, reject the request with an unauthorized response.
* Log Request

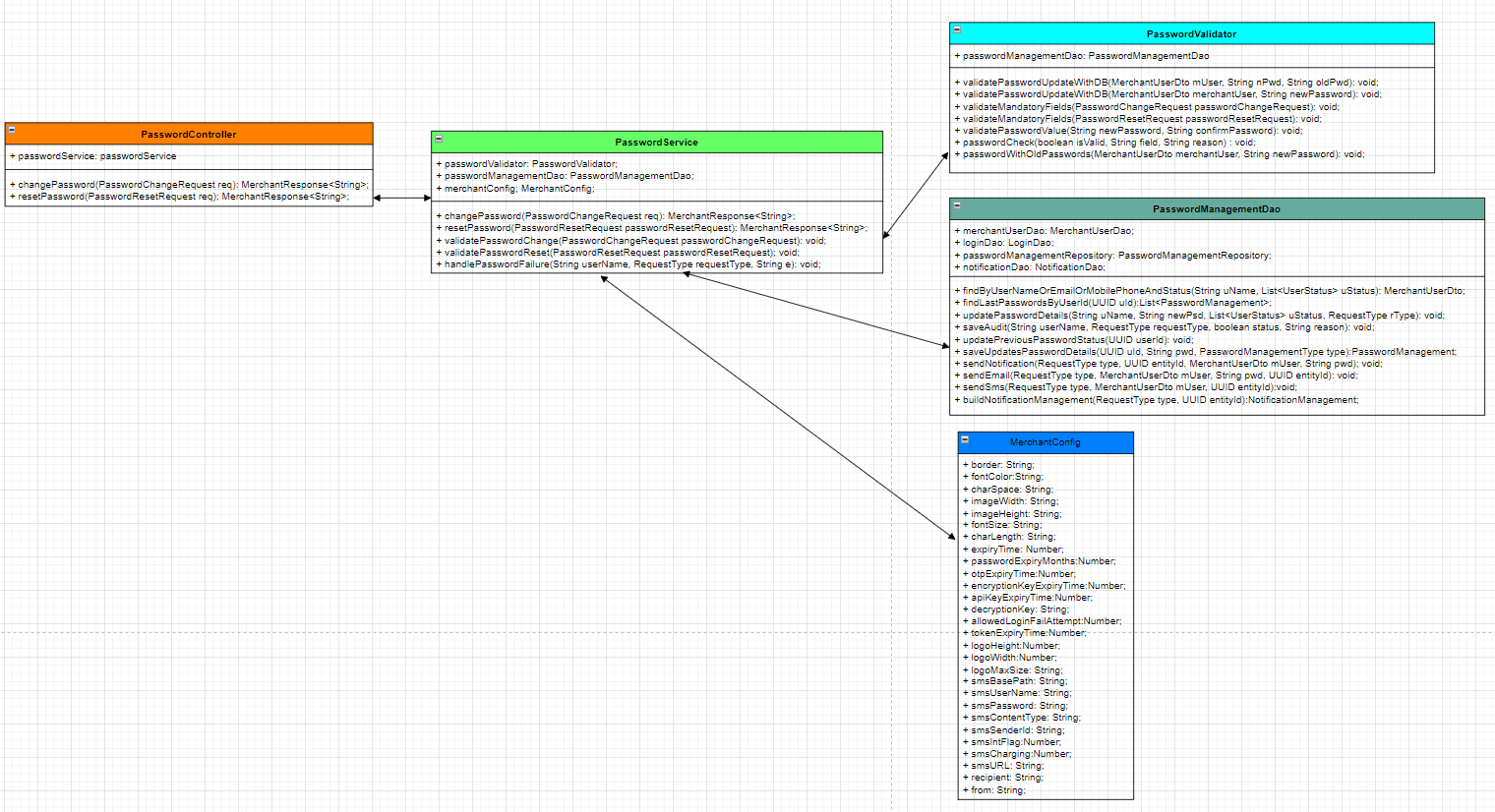
Log all incoming requests, including headers and body parameters (excluding sensitive information) to the database.

* Log Response

Log all responses, including generated token details and any error messages except the token.

### Change Password

#### Class Diagram



#### API Definition / Implementation

* **Endpoint:** /merchant/v1/password/change
* **Method:** POST
* **Description:** This API provides capability to change password.
* **Request details:**
* **Content-Type:** application/json
* **Request object:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Description** | **Mandatory** | **Data Type** |
| userName | Merchant userName / emailId / mobile | Yes | string |
| oldPasswor | Hashed old password | Yes | string |
| newPassword | Hashed new password | Yes | String |
| confirmPassword | Hashed confirm new password | Yes | String |

* + **Example Request**

{

"userName": "test",

"oldPassword": "hiLSRAdY/NmaMscVsw9gOrVBC1dSfz0VK2/ts0DGNXmKp8xXzR59Yt5BcC9w6nDu1qY17cKB110P0IK2B8cilg==",

"newPassword": "hppSRAdY/NmaMscVsw9gOrVBC1dSfz0VK2/ts0DGNXmKp8xXzR59Yt5BcC9w6nDu1qY17cKB110P0IK2B8cilg==",

"confirmPassword": "hppSRAdY/NmaMscVsw9gOrVBC1dSfz0VK2/ts0DGNXmKp8xXzR59Yt5BcC9w6nDu1qY17cKB110P0IK2B8cilg=="

}

* **Example** **response –** Successs 200 ok

{

"data": [

{

"token": "Password Changed Successsfully"

}

],

"status":1,

}

* **Example Response –** Error 200 ok

{

"errors": [

{

"errorCode":"ERROR\_CODE ",

"errorMessage":"ERROR\_MSG"

},

],

"status":0

}

|  |  |
| --- | --- |
| Error Code | Error Description |
| 7001 | User Name/Old password/New password/ Confirm Password is mandatory. |
| 7002 | New password is invalid. Reason: New password does not meet the acceptance criteria |
| Password is invalid. Reason: The new password and confirmation password do not match |
| 7003 | Merchant user not found. |
| Respective generic error code and description with field Name | |

#### Validation

* **userName:**  - Validate merchant userName or email Id or mobile number.
* **oldPassword -** This field must contain a valid old password.
* **newPassword:**  - This field must contain a valid pattern for the new password.
* **confirmPassword -** This field must contain the same value as newPassword.
* New password and confirm password should be matched.
* The new password should be as per the password policy.
* The user must have an old password and expired status.

#### Business Logic

* **Password controller**

Request reaches password controller and calls the password service.

* **Password service**

Service validates request for mandatory fields, calls the DAO to check user.

* **Password validator**

Validates old password, new password and confirm password provided by user.

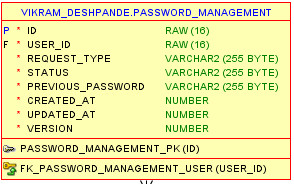
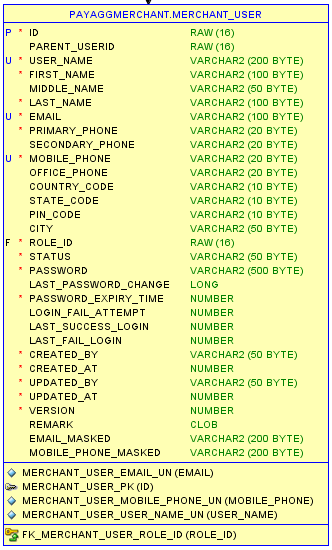
* **Password management DAO**

On successful validation, password will be updated.

* **Password service**

On success send success response, otherwise send custom exception.

#### Database Table – Update MERCHANT\_USER and PASSWORD\_MANAGEMENT



#### Information

* + Log Request

Log all incoming requests, including headers and body parameters (excluding sensitive information) to the database.

* + Log Response

Log all responses,including generated token details and any error messages except the token.

### Reset Password

#### API Definition / Implementation

* **Endpoint:** /merchant/v1/password/reset
* **Method:** POST
* **Description:** This API provides capability to reset password.
* **Request details:**
* **Content-Type:** application/json
* **Response object:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Description** | **Mandatory** | **Data Type** |
| userName | Merchant userName / emailId / mobile | Yes | string |
| newPassword | Hashed new password | Yes | String |
| confirmPassword | Hashed confirm new password | Yes | String |

* + **Example Request**

{

{

"userName": "test",

"newPassword": "hiLSRAdY/NmaMscVsw9gOrVBC1dSfz0VK2/ts0DGNXmKp8xXzR59Yt5BcC9w6nDu1qY17cKB110P0IK2B8cilg== ",

"confirmPassword": "hiLSRAdY/NmaMscVsw9gOrVBC1dSfz0VK2/ts0DGNXmKp8xXzR59Yt5BcC9w6nDu1qY17cKB110P0IK2B8cilg== "

}}

|  |
| --- |
| 7002 |

|  |  |
| --- | --- |
| **Error Code** | **Error Description** |
| 7001 | User Name/New password/ Confirm Password is mandatory. |
| 7002 | New password is invalid. Reason : New password does not meet the acceptance criteria |
| Password is invalid. Reason : The new password and confirmation password do not match |
| 7003 | Merchant user not found. |
| Respective generic error code and description with field Name | |

**Example** **Response –** Success 200 ok

{

"data": [

{

"token": “Password Reset Successful."

}

],

"status":1,

}

**Example Response –** Error 200 ok

{

"errors": [

{

"errorCode":"ERROR\_CODE ",

"errorMessage":"ERROR\_MSG"

},

],

"status":0

}

#### Validation

##### Request Data Validation

* **userName:**  This field validates merchant userName or email id or mobile number.
* **newPassword:** This field must contain a valid pattern for the new password.
* **confirmPassword:** This field must contain the same value as newPassword.
* New password and confirm password should be matched.
* The new password should be as per the password policy.
* The user must have an old password and active status.

#### Business Logic

* **Step 1: Password controller**

Request reaches Password controller and calls password service.

* **Step 2: Password service**

Service validates request for mandatory fields, calls DAO to check user.

* **Step 3: Password validator**

Validates new password and confirm password provided by user.

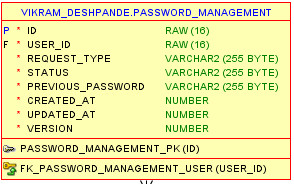
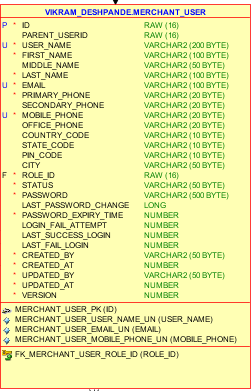
* **Step 4: Password management DAO**

On successful validation password will be updated.

* **Step 5: Password service**

On success, send success response, otherwise send custom exception.

#### Database table – Update MERCHANT\_USER and PASSWORD\_MANAGEMENT



#### Information

* + Log Request

Log all incoming requests, including headers and body parameters (excluding sensitive information) in the database.

* + Log Response

Log all responses, including generated token details and any error messages except the token.

### Token Validation

#### API Definition / Implementation

* **Endpoint:** /merchant/v1/validation/ token
* **Method:** GET
* **Description:** This API provides capability to OTP validation for user login or other actions requiring validation. A valid token is required to perform certain actions including payments and transactions, viewing reports, managing users, raising invoice, etc.
* **Request details:**
* **Content-Type:** application/json
* **Request object:** None
* **Example** **response –** Success 200 ok

{

{

"status": 1

}

}

* **Example Response –** Error 200 ok

{

"errors": [

{

"errorCode":"ERROR\_CODE ",

"errorMessage":"ERROR\_MSG"

},

],

"status":0

}

#### Validation

* For existing valid user, send Success response with JWT token otherwise send custom exception.

#### Business Logic - NA

#### Information - NA

### Merchant User Captcha Validation

#### API Definition / Implementation

* **Endpoint:** /merchant/v1/validation/user/captcha
* **Method:** GET
* **Description:** This API validates the merchant user and captcha during password reset. Merchant user’s details including username, email Id, and phone number are validated It also checks if the captcha is active or inactive.
* **Request details:**
* **Content-Type:** application/json
* **Request body:**

{

"userName": "Neeraj1",

"requestId": "7cf8a379-7b81-41ed-822e-7033988c89a4",

"requestType": "RESET\_PASSWORD",

"captchaText": "iUw/qN5/jG9U1hv+GP2llmVZwjJChsREsBPLqwQV+MboiOjw8EaAZ75oDwZqwuANbfRx5SvxT0PvoPW148GsKw=="

}

* **Example** **response –** Success 200 ok

{

"status": 1,  
 "data": [  
 "User validated Successfully"  
 ]

}

* **Example Response –** Error 200 ok

{

"errors": [

{

"errorCode":"ERROR\_CODE ",

"errorMessage":"ERROR\_MSG"

},

],

"status":0

}

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Description** | **Mandatory** | **Data Type** |
| userName | Merchant userName / emailId / mobile | Yes | string |
| requestId | Request Id | Yes | UUID |
| requestType | LOGIN("LOGIN"), RESET\_PASSWORD("RESET\_PASSWORD"), CHANGE\_PASSWORD("CHANGE\_PASSWORD"), PROFILE\_EDIT("PROFILE\_EDIT"); | Yes | String |
| captchaText | Captcha request | Yes | String |

#### Validation

* For existing valid user check access token is not expired otherwise send custom exception.
* Validate user for valid name or phone number or email.
* Validate request type is for reset password.
* Validate captcha is still active and
* Check for username and status should be active.

#### Business Logic

* Validation controller

Request reaches controller and service is called for further processing.

* Validation service

Validate captcha request

* Validate request type.
* Validate merchant user validation request for captcha.

#### Information

* Log Request

Log all incoming requests, including headers and body parameters (excluding sensitive information) to the database.

* Log Response

Log all responses, including generated token details and any error messages except the token.

#### Validate logged in user has access to active mId

#### API Definition / Implementation

* **Endpoint:** /merchant/v1/validation/access/active/{userName}/{mId}
* **Method:** GET
* **Description:** This API ensures that the logged-in user has access to the mId and that the mId is in the active state. A list of merchant info is checked to verify that the user name is valid and the mId is active. It also verifies that the token used is a valid one.
* **Request details:**
* **Content-Type:** application/json
* **Request parameters**

userName – Username

mId – Merchant identifier

* **Request body:** None

**Sample Request URL:**

<https://dev.epay.sbi/merchant/v1/validation/access/active/Neeraj1/1000003>

* **Example** **response –** Success 200 ok

{

"status": 1,  
 "data": [  
 "MId Validated Successfully"  
 ]

}

* **Example Response –** Error 200 ok

{

"errors": [

{

"errorCode":"ERROR\_CODE ",

"errorMessage":"ERROR\_MSG"

},

],

"status":0

}

#### Validation

* For existing valid user check access token is not expired otherwise send custom exception.
* Check access to merchant identifier for a merchant user.

#### Business Logic

* Validation controller - Request reaches controller and service is called for further processing.
* Validation service
* Use to fetch list of merchant info for user name.
* Process merchant info list to check user has access to mId.
* Access to mID exists, send success message otherwise raise custom exception.

#### Information

* Log Request

Log all incoming requests, including headers and body parameters (excluding sensitive information) to the database.

* Log Response

Log all responses, including generated token details and any error messages except the token.

### Validate logged in user has access to particular mId

#### API Definition / Implementation

* **Endpoint:** /merchant/v1/validation/access/{userName}/{mId}
* **Method:** GET
* **Description:** This API validates that a merchant user has a valid user name and captcha.
* **Request details:**
* **Content-Type:** application/json
* **Request parameters**

userName – User name

mId – Merchant identifier

* **Request body:** None

**Request URL:** <https://dev.epay.sbi/merchant/v1/validation/access/Neeraj1/1000003>

* **Example** **response –** Success 200 ok

{

"status": 1,  
 "data": [  
 "MId Validated Successfully"  
 ]

}

* **Example Response –** Error 200 ok

{

"errors": [

{

"errorCode":"ERROR\_CODE ",

"errorMessage":"ERROR\_MSG"

},

],

"status":0

}

#### Validation

* For an existing valid user, check that the access token is not expired otherwise send custom exception.
* Process merchant info list to check user has access to MID and MID should be active.
* Check that the user has access to merchant details.

#### Business Logic

* Validation controller

Request reaches controller and service is called for further processing.

* Validation service
* Use to fetch list of merchant info for user name.
* Process merchant info list to check user has access to MID and MID should be active.

#### Information

* Log Request

Log all incoming requests, including headers and body parameters (excluding sensitive information) to the database.

* Log Response

Log all responses, including generated token details and any error messages except the token.

# SD\_MS - Merchant Portal – Merchant Panel

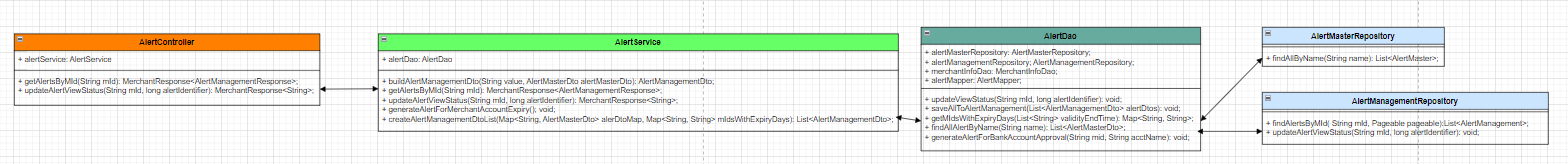


## Menu Bar

### Theme button – UI implementation. Users can select dark, light themes.

### Merchant Portal – User Alerts.

#### Class Diagram



#### Merchant Portal - Update User Alerts (TBD)

##### API Definition and Implementation

* **Endpoint**: /merchant/v1/alert/{mId}
* **Method:** GET
* **Description:** On Successful login, this API sends alerts related to transaction limit, password validity, bank account etc.
* **Request details:**
* **Authorization:** Bearer <Merchant Access Token (Admin Portal Authorization)
* **Path Parameter:** The Merchant ID, used to identify the merchant account.
* **Response:**
* **Example response** – Success 200

{

"status": 1,

"data": [

{

"alertDescription": string,

"alertIdentifier": number

}

]

}

**Sample response**

|  |
| --- |
| {  "status": 1,  "data": [],  "count": 0 } |

* **Example response** – Error 200 ok

{

"errors": [

{

"errorCode":"ERROR\_CODE ",

"errorMessage":"ERROR\_MSG"

},

],

"status":0

}

##### Validation

* Header should have valid JWT token.
* The API can only be accessed by an an authenticated users via the merchant access token.
* User must have active status.

##### Business Logic

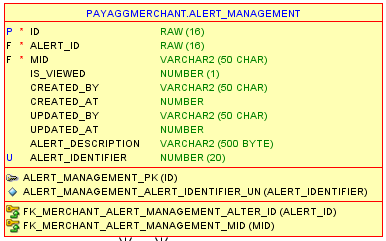
* **Step 1: User alert controller**

Request reaches user alert controller and calls the merchant user service.

* **Step 2: User alert service**
* Fetch alert entries from DB for a merchant identifier.
* Filter alerts if user have role as “USER” else send response with all

data retrieved from database.

##### Database table – ALERT\_MANAGEMENT



##### Information

* + Log Request

Log all incoming requests, including headers and body parameters (excluding sensitive information) to the database.

* + Log Response

Log all responses, including generated token details and any error messages except the token.

##### Merchant Portal – Update User Alerts

##### API Definition and implementation

* **Endpoint**: /merchant/v1/alert/{mId}/{alertIdentifier}
* **Method:** PUT
* **Description:** On Successful login, the user will receive an alert related to Transaction limit, password validity, bank account etc. API can update read status of alert message.
* **Request details**
* **Authorization:** Bearer <Merchant Access Token (Admin Portal Authorization)
* **Path Parameter:** The Merchant ID, used to identify the merchant account.
* **Sample request URL**: https://dev.epay.sbi/merchant/v1/alert/1000003/1
* **Response**

**Example response** – Success 201 ok

|  |
| --- |
| "status": 1,  "data": [  "View updated Successfully"  ],  "count": 1,  "total": 1 } |

* **Example response** – Error 200 ok

{

"errors": [

{

"errorCode":"ERROR\_CODE ",

"errorMessage":"ERROR\_MSG"

},

],

"status":0

}

##### Validation

* Header should have a valid JWT token.
* JWT token should have the valid Merchant user role.
* The API can only be accessed by an authenticated users via the merchant access token.
* User must be in the active status.

##### Business Logic

* **Step 1: User alert controller**

Request reach user alert controller and call merchant user service.

* **Step 2: User alert service**

Calluser a DAO to update alert status.

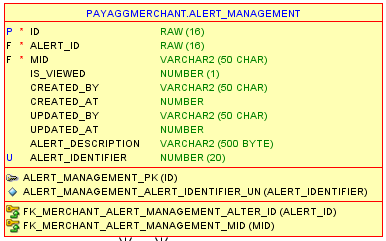
* **Step 3: User alert DAO**

Update alert entry in DB.

* **Step 4: User alert service**

Send success response otherwise send custom exception.

##### Database Table – ALERT\_MANAGEMENT



##### Information

* If invalid, reject the request with an unauthorized response.
* Log Request:

Log all incoming requests, including headers and body parameters (excluding sensitive information) to the database.

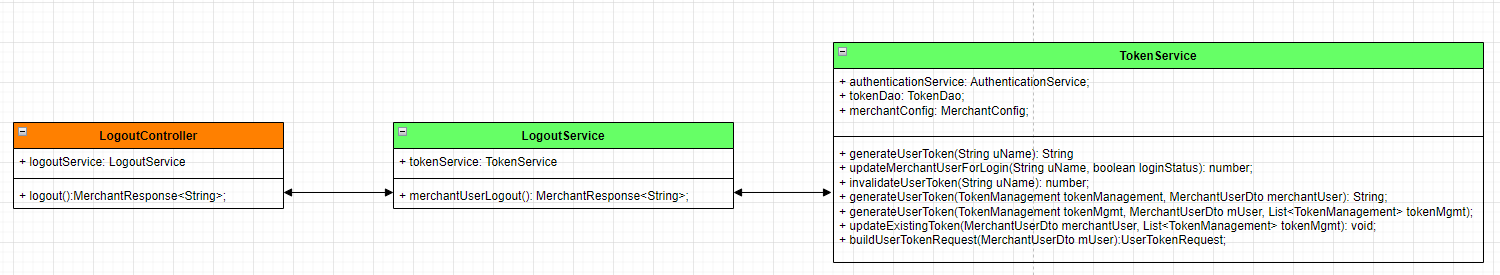
* Log Response:

Log all responses, including generated token details and any error messages except the token.

#### User Profile – Login User Profile, API defined in section 2.6.1.1

#### Merchant Portal - User Logout

##### Class diagram



##### API Definition and Implementation

* **Endpoint**: /merchant/v1/user/logout
* **Method:** POST
* **Description:** This API allows the merchant user to log out of the system Successsfully. It sets the merchant access token status as expired..
* **Request details:**
* **Authorization:** Bearer <Merchant Access Token (Admin Portal Authorization)
* Sample request URL : https://dev.epay.sbi/merchant/v1/user/logout
* **Response:**

**Example response** – Success 201 ok

|  |
| --- |
| {  "status": 1,  "data": [  "Log0ut Successfully"  ] } |

* **Example response** – Error 200 ok

{

"errors": [

{

"errorCode":"ERROR\_CODE ",

"errorMessage":"ERROR\_MSG"

},

],

"status":0

}

##### Validation

* The API can be accessed only by the authenticated users via the merchant access token.
* Check for the merchant access token in the database, if not found display a custom exception.
* Check that the useris in theactive status.

##### Business logic

* **Step 1: Merchant user controller**

Request reaches merchant user controller and calls the merchant user service.

* **Step 2: Merchant user service**

Service validates request for username active status, and then calls the DAO to check user.

* **Step 3: Merchant user validator**

Validate token if exists and token status.

* **Step 4: Merchant user service**

Calluser DAO to update status.

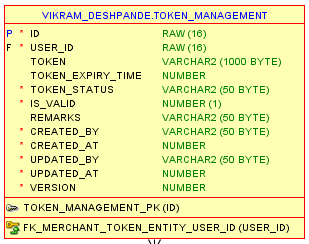
* **Step 5: Merchant user DAO**

Update status of token status, set to Expired.

* **Step 6: User notification service**

Send success response otherwise send custom exception.

##### Database table – TOKEN\_MANAGEMENT

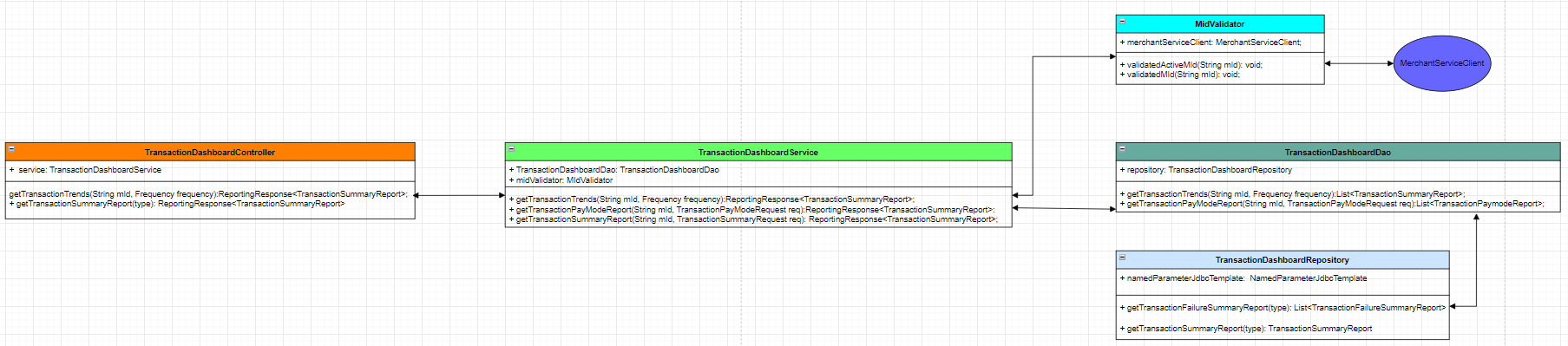


##### Information

* If invalid, reject the request with an unauthorized response.
* Log Request
  + Log all incoming requests, including headers and body parameters (excluding sensitive information) to the database.
* Log Response
  + Log all responses, including generated token details and any error messages except the token.

## Dashboard

### Class Diagram



### Transaction Trends

#### API Definition and Implementation

* **Endpoint**: /reporting/v1/transaction/trends/{mId}/{frequency}
* **Method**: GET
* **Description:** This API retrieves transaction trend data for a given merchant ID (MID) based on the specified frequency (DAILY/MONTHLY/YEARLY). It also checks if the mId is available and valid.
* **Request details:** 
  + Headers: Requires valid login token.

**Authorization:** Bearer <login token>

* Path params:

|  |  |
| --- | --- |
| Parameter | comment |
| mId | Merchant identifier |
| frequency | <DAILY or MONTHLY or YEARLY > |

* Request body: Empty
* **Response details:**
* **Response:** Success: 200 – Ok

{  
 "status": 1,  
 "data": [  
 {  
 "totalSuccessAmount": number,  
 "totalSuccessCount": number,  
 "totalFailureCount": number,  
 "transactionDate": number,  
 "totalFailureAmount": number,  
 "SuccessPercentage": number,  
 "failPercentage": number,  
 "transactionDailyFailure": [  
 {  
 "failureReason": "string"  
 "failureCount": number,  
 "failurePercentage": number  
 }  
 ]  
 }

],  
 "count": 1,  
 "total": 1,  
}

* **Sample Success Response**

|  |
| --- |
| {  "data": [  {  "totalSuccessAmount": 7,  "totalSuccessCount": 5,  "transactionDate": "2025-03-20 00:00:00"  },  {  "totalSuccessAmount": 7,  "totalSuccessCount": 3,  "transactionDate": "2025-03-19 00:00:00"  },  {  "totalSuccessAmount": 51,  "totalSuccessCount": 16,  "transactionDate": "2025-03-18 00:00:00"  },  {  "totalSuccessAmount": 19,  "totalSuccessCount": 15,  "transactionDate": "2025-03-17 00:00:00"  },  {  "totalSuccessAmount": 22,  "totalSuccessCount": 19,  "transactionDate": "2025-03-15 00:00:00"  },  {  "totalSuccessAmount": 10,  "totalSuccessCount": 7,  "transactionDate": "2025-03-13 00:00:00"  },  {  "totalSuccessAmount": 16,  "totalSuccessCount": 8,  "transactionDate": "2025-03-12 00:00:00"  },  {  "totalSuccessAmount": 0,  "totalSuccessCount": 0,  "transactionDate": "2025-03-11 00:00:00"  }  ]  } |

* **Response**: Error 200 – Ok

{

"status":0,

"error": [

{

"errorCode": String,

"errorMessage": String

}

]

}

##### Validation

* Check if an mId is available and valid. If not, then return an error message.

##### Business Logic

* **Step 1: Transaction dashboard controller**

Request reaches controller and calls transaction dashboard service.

* **Step 2: Transaction dashboard service**

Call MID validator service to validate merchant identifier, on Successful validation call transaction dashboard DAO.

* **Step 3: Transaction dashboard DAO**

Call transaction dashboard repository.

* **Step 4: Transaction dashboard repository**

Repository will fetch records for daily trends.

* **Step 5: Transaction dashboard service**

Process data and generate transaction trends response.

* **Step 6: Merchant User service**

Send error or custom exception if any exception.

##### Information

* If valid, extract user details and set them in the security context.
* If invalid, reject the request with an unauthorized response.
* Log Request

Log all incoming requests, including headers and body parameters (excluding sensitive information) to the database.

* Log Response

Log all responses, including generated token details and any error messages except the token.

##### Database View

View is derived from the transaction tables.

|  |
| --- |
| **Table name** |
| TRANSACTION\_TXN |
| AGG\_MERCHANT\_ORDER\_HYBRID\_FEE\_DTLS\_TXN |

|  |  |  |
| --- | --- | --- |
| VIEW\_TRANSACTION\_DAILY\_REPORT | | |
| **Column** | **Data type** | **Nullable** |
| MID | VARCHAR2(100) | Yes |
| TOTAL\_DEBIT\_AMOUNT | NUMBER | Yes |
| TOTAL\_SUCCESS\_AMOUNT | NUMBER | Yes |
| TOTAL\_FAILED\_AMOUNT | NUMBER | Yes |
| SUCCESS\_COUNT | NUMBER | Yes |
| FAIL\_COUNT | NUMBER | Yes |
| TRANSACTION\_DATE | DATE | Yes |

### Transaction Summary

#### API Definition and Implementation

* **Endpoint**: /reporting/v1/transaction/summary/{mId}
* **Method**: POST
* **Description:**Success. This API creates a summary report for all the Successs and failure transactions for a given merchant ID (mId) and for a specified date range.
* **Path params**

|  |  |
| --- | --- |
| Parameter | comment |
| mId | Merchant identifier |

* **Request details:** 
  + Headers: Requires a valid login token.

**Authorization:** Bearer <login token>

* **Request body:**

|  |  |  |  |
| --- | --- | --- | --- |
| Property | Description | Mandatory | Data Type |
| fromDate | From Date | Yes | string |
| toDate | To Date | Yes | String |

* Response details:
* **Response:** Success: 200 – Ok

{  
 "status": 1,  
 "data": [  
 {  
 "totalSuccessAmount": number,  
 "totalSuccessCount": number,  
 "totalFailureCount": number,  
 "transactionDate": number,  
 "totalFailureAmount": number,  
 "SuccessPercentage": number,  
 "failPercentage": number,  
 "transactionDailyFailure": [  
 {  
 "failureReason": "string"  
 "failureCount": number,  
 "failurePercentage": number  
 }  
 ]  
 }

],  
 "count": 1,  
 "total": 1,  
}

* **Response**: Error 200 – Ok

{

"status":0,

"error": [

{

"errorCode": String,

"errorMessage": String

}

]

}

##### Validation

* . Check if a particular mId is available and valid. If not, then return an error message.

##### Business Logic

* **Step 1: Transaction dashboard controller**

Request reaches controller to get transaction summary and then call’s transaction dashboard service.

* **Step 2: Transaction dashboard service**

The transaction dashboard service calls MID validator service to validate merchant identifier., On successful validation, it calls the transaction dashboard DAO.

* **Step 3: Transaction dashboard DAO**

The transaction dashboard DAO calls transaction dashboard repository to display the summary data.

* **Step 4: Transaction dashboard repository**

The transaction dashboard repository displays the records for transaction summary of a merchant.

* **Step 5: Transaction dashboard service**

The transaction dashboard service processes data and sends a summary response.

##### Information

* Log Request

Log all incoming requests, including headers and body parameters (excluding sensitive information).

* Log Response

Log all responses, including generated token details and any error messages except the token.

##### Database Table

View is derived from the transaction tables.

|  |
| --- |
| **Table name** |
| TRANSACTION\_TXN |
| AGG\_MERCHANT\_ORDER\_HYBRID\_FEE\_DTLS\_TXN |

|  |  |  |
| --- | --- | --- |
| VIEW\_TRANSACTION\_PAYMODE\_DAILY\_REPORT | | |
| **Column** | **Data type** | **Nullable** |
| MID | VARCHAR2(100) | Yes |
| TOTAL\_DEBIT\_AMOUNT | NUMBER | Yes |
| PAYMODE\_CODE | VARCHAR2(50) | Yes |
| TOTAL\_SUCCESS\_AMOUNT | NUMBER | Yes |
| TOTAL\_FAILED\_AMOUNT | NUMBER | Yes |
| SUCCESS\_COUNT | NUMBER | Yes |
| FAIL\_COUNT | NUMBER | Yes |
| TRANSACTION\_DATE | DATE | Yes |

### Transaction – Refund Details (TBD)

### Transaction – Details (TBD)

### Transaction – Chargeback Details (TBD)

### Transaction – Daily Settlement Summary

#### API Definition and Implementation

* **Endpoint**: /reporting/v1/daily/summary/settlement/{mId}
* **Method**: GET
* **Description:** This API provides daily settlement data for a given merchant ID
* **Request body:** Empty
* **Path parameter:**
* **mId –** Merchant identifier
* **Response body:**
* Response - Success 200 Ok

{

“status”: 1,

“data”: [

{

"todaySettlement": number

"pastSettlement": number

}

],  
 "count": 1,  
 "total": 1,

}

* Response – Error 200 Ok

{

"status":0,

"error": [

{

"errorCode": String,

"errorMessage": String

}

]

}

#### Validation

* . Check if an mId is available and valid. If not, then return an error message.

##### Business logic

* **Step-1: TransactionDailySummary controller**

Request reaches controller and TransactionDailySummary service for further processing.

* **Step-2: TransactionDailySummary service**
  + Validate merchant, call external service to validate if merchant has access. Raise exception if no access found.
  + DAO, use TransactionSummary Repository to fetch settlement data for today and prior day.
  + Send response with current day settlement and past day settlement amount. For error exception send appropriate custom message.

##### Information

* Log Request

Log all incoming requests, including headers and body parameters (excluding sensitive information).

* Log Response

Log all responses, including generated token details and any error messages except the token.

##### Materialized View

**VIEW\_TRANSACTION\_DASHBOARD\_REPORT**

### Transaction – Daily Refund Summary

##### API Definition and implementation

* **Endpoint**: /reporting/v1/daily/summary/refund/{mId}
* **Method**: GET
* **Description:** This API provides daily refund summary data for a given merchant ID
* **Request body:** Empty
* **Path parameter:**
* **mId –** Merchant identifier
* **Response body:**
* Response - Success 200 Ok

{

“status”: 1,

“data”: [

{

"refundCount": number,

"totalAvailableAmt": number,

"totalRefundAmt": number,

"createdDate": <date string>

}

],  
 "count": 1,  
 "total": 1,

}

* Sample Success Response

|  |
| --- |
| "data": [  {  "refundCount": 8,  "totalAvailableAmt": 12.2,  "totalRefundAmt": 0.8,  "createdDate": "24 MAR 2025"  }  ]  } |

* **Response** – Error 200 Ok

{

"status":0,

"error": [

{

"errorCode": String,

"errorMessage": String

}

]

}

##### Validation

* + . Check if an mId is available and valid. If not, then return an error message.

##### Business logic

* **Step-1: TransactionDailySummary controller**

Request reaches the controller and calls the TransactionDailySummary service for further processing.

* **Step-2: TransactionDailySummary service**
* Validates merchant, calls external service to validate if merchant has access to the Merchant Portal. Displays an error message if no access found.
* Use refund summary DAO to fetch refund data for today and prior day.
* Sends response with current day refund and previous day refund amount. Displays an error message, if any error occurs.

##### Information

* Log Request

Log all incoming requests, including headers and body parameters (excluding sensitive information) .

* Log Response

Log all responses, including generated token details and any error messages except the token.

##### Materialized View

VIEW\_REFUND\_DASHBOARD\_SUMMARY

### Transaction – Daily Transaction Summary

##### API Definition and implementation

* **Endpoint**: /reporting/v1/daily/summary/transaction/{mId}
* **Method**: GET
* **Description:** This API provides a summary of daily transaction data for a given merchant ID
* **Request body:** Empty
* **Path parameter:**
* **mId –** Merchant identifier
* **Response body:**
* Response - Success 200 Ok

{

“status”: 1,

“data”: [

{

"totalTransactionCount": number,

"totalAmount": number,

"transactionDate": string

}

],  
 "count": 1,  
 "total": 1,

}

* **Response** – Error 200 Ok

{

"status":0,

"error": [

{

"errorCode": String,

"errorMessage": String

}

]

}

##### Validation

##### Check if a particular mId is available and valid, otherwise return an error message.

##### Business logic

* **Step-1: TransactionDailySummary controller**

Request reaches controller and TransactionDailySummary service for further processing.

* **Step-2: TransactionDailySummary service**
* Validate merchant, call external service to validate if merchant has access. Raise exception if no access found.
* DAO, use TransactionSummary Repository to fetch transaction data for today and prior day.
* Send response with current day settlement and past day settlement amount. For error exception send appropriate custom message.

##### Information

* Log Request

Log all incoming requests, including headers and body parameters (excluding sensitive information).

* Log Response
* Log all responses, including generated token details and any error messages except the token.

##### Materialized View

**VIEW\_TRANSACTION\_DASHBOARD\_REPORT**

### Transaction - Payment Mode Health

#### Downtime API - Listing

##### API Definition and implementation

* **Endpoint**: /merchant/v1/downtime/api/list
* **Method**: GET
* **Description:** This API enables the merchant service to call the admin service to get a list of valid payment modes.
* **Request body:** Empty
* **Response:**
* **Sample response -** Success 200 OK

{

“status”:1,

“data”: [

{

"payGtwId":"string",

"payGtwName":"string",

"payModeCode":"string",

"downTimeStartDatetime":"string",

"downTimeEndDatetime":"string"

}

]

}

Sample Success response

|  |
| --- |
| {  "status": 1,  "data": [  {  "payGtwId": "140",  "payModeCode": "WALLET",  "downTimeStartDatetime": "2024-11-08 15:15:18",  "downTimeEndDatetime": "2024-11-08 15:20:18"  },  {  "payGtwId": "76",  "payModeCode": "NB",  "downTimeStartDatetime": "2021-04-15 11:12:19",  "downTimeEndDatetime": "2021-04-15 14:17:19"  },  {  "payGtwId": "51",  "payModeCode": "NB",  "downTimeStartDatetime": "2021-04-17 20:01:45",  "downTimeEndDatetime": "2021-04-17 20:06:45"  },  {  "payGtwId": "51",  "payModeCode": "NB",  "downTimeStartDatetime": "2021-04-17 20:40:07",  "downTimeEndDatetime": "2021-04-17 20:45:07"  },  {  "payGtwId": "15",  "payModeCode": "NB",  "downTimeStartDatetime": "2024-07-26 11:17:25",  "downTimeEndDatetime": "2024-07-26 12:00:09"  },  {  "payGtwId": "98",  "payModeCode": "NB",  "downTimeStartDatetime": "2021-04-16 10:26:18",  "downTimeEndDatetime": "2021-04-16 10:59:18"  },  {  "payGtwId": "20",  "payModeCode": "NB",  "downTimeStartDatetime": "2021-04-16 14:54:53",  "downTimeEndDatetime": "2021-04-16 15:59:53"  },  {  "payGtwId": "38",  "payModeCode": "NB",  "downTimeStartDatetime": "2021-04-17 22:46:05",  "downTimeEndDatetime": "2021-04-17 23:51:05"  },  {  "payGtwId": "142",  "payModeCode": "NB",  "downTimeStartDatetime": "2021-04-16 12:35:24",  "downTimeEndDatetime": "2021-04-16 12:40:24"  },  {  "payGtwId": "140",  "payModeCode": "WALLET",  "downTimeStartDatetime": "2024-11-08 15:15:18",  "downTimeEndDatetime": "2024-11-08 15:20:18"  },  {  "payGtwId": "76",  "payModeCode": "NB",  "downTimeStartDatetime": "2021-04-16 10:15:19",  "downTimeEndDatetime": "2021-04-16 11:20:19"  },  {  "payGtwId": "26",  "payModeCode": "NB",  "downTimeStartDatetime": "2021-11-22 12:24:08",  "downTimeEndDatetime": "2021-11-22 12:29:08"  },  {  "payGtwId": "27",  "payModeCode": "NB",  "downTimeStartDatetime": "2021-04-16 15:12:58",  "downTimeEndDatetime": "2021-04-16 15:17:58"  },  {  "payGtwId": "27",  "payModeCode": "NB",  "downTimeStartDatetime": "2021-04-16 10:27:08",  "downTimeEndDatetime": "2021-04-16 10:41:08"  },  {  "payGtwId": "52",  "payModeCode": "NB",  "downTimeStartDatetime": "2021-04-17 20:58:41",  "downTimeEndDatetime": "2021-04-17 22:03:41"  },  {  "payGtwId": "42",  "payModeCode": "NB",  "downTimeStartDatetime": "2021-04-17 21:43:09",  "downTimeEndDatetime": "2021-04-17 21:48:09"  }  ],  "count": 16  } |

* Sample response: Error 200 Ok

{

"errors": [

{

"errorCode":"ERROR\_CODE ",

"errorMessage":"ERROR\_MSG"

},

],

"status":0

}

##### Validation - None

##### Business Logic

* **Step 1:Downtime API controller**

Request reach user downtime API controller and call downtime API service.

* **Step 2: Downtime API service**

Calluser a Downtime API DAO.

* **Step 3: Downtime API DAO**

Downtime API DAO calls Admin Client to fetch list of paymodes.

* **Step 4: Downtime API service**

Send success response otherwise send custom exception.

#### Downtime API - Listing

##### API Definition and Implementation

* **Endpoint**: /merchant/v1/downtime/api/details/{mId]
* **Method**: GET
* **Description:** This API enables the merchant service to call the admin service to get a list of valid payment modes for a valid merchant Id. **Path variable:** Merchant id
* **Request body:** Empty
* **Response body:**
* **Sample response body-** Success 200 OK

{

"staus": 1,

“data”: [

{

"payMode"; "string",

"subPayMode": "string"

"downTimeDetails": [

"code": "string",

"name": "string",

"paymentGateWayId": number,

"isPayModeDown": boolean

]

"downtimeCount": "string"

}

]

}

* Sample Success response

|  |
| --- |
| {  "status": 1,  "data": [  {  "payMode": "cards",  "subPayMode": "cc",  "downTimeDetails": [  {  "code": "RUPAY",  "name": "State Bank of India New PG",  "paymentGateWayId": "171",  "payModeDown": **false**  },  {  "code": "VISA",  "name": "State Bank of India New PG",  "paymentGateWayId": "139",  "payModeDown": **false**  },  {  "code": "MASTER",  "name": "State Bank of India New PG",  "paymentGateWayId": "138",  "payModeDown": **false**  }  ],  "downtimeCount": 0  },  {  "payMode": "cards",  "subPayMode": "pc",  "downTimeDetails": [  {  "code": "MASTER",  "name": "State Bank of India New PG",  "paymentGateWayId": "163",  "payModeDown": **false**  },  {  "code": "RUPAY",  "name": "State Bank of India New PG",  "paymentGateWayId": "172",  "payModeDown": **false**  },  {  "code": "VISA",  "name": "State Bank of India New PG",  "paymentGateWayId": "164",  "payModeDown": **false**  }  ],  "downtimeCount": 0  },  {  "payMode": "INB",  "downTimeDetails": [  {  "name": "STATE BANK OF INDIA",  "payModeDown": **false**  },  {  "code": "119",  "name": "Mehsana Urban Co. Op. Bank Ltd - Corporate",  "paymentGateWayId": "115",  "payModeDown": **false**  },  {  "code": "41",  "name": "Karnataka Bank",  "paymentGateWayId": "66",  "payModeDown": **false**  },  {  "code": "4553",  "name": "IndusInd Bank- Corporate",  "paymentGateWayId": "224",  "payModeDown": **false**  },  {  "code": "133",  "name": "Bank Of Baroda Corp",  "paymentGateWayId": "149",  "payModeDown": **false**  },  {  "code": "4557",  "name": "Bandhan Bank - Retail",  "paymentGateWayId": "230",  "payModeDown": **false**  },  {  "code": "122",  "name": "Indian OverSeas Bank Retail",  "paymentGateWayId": "120",  "payModeDown": **false**  },  {  "code": "4544",  "name": "Yes Bank - Corporate",  "paymentGateWayId": "190",  "payModeDown": **false**  },  {  "code": "116",  "name": "HDFC-DP Bank",  "paymentGateWayId": "108",  "payModeDown": **false**  },  {  "code": "18",  "name": "Jammu and Kashmir",  "paymentGateWayId": "38",  "payModeDown": **true**  },  {  "code": "127",  "name": "punjab and maharashtra co-operative bank ltd - Corporate",  "paymentGateWayId": "131",  "payModeDown": **false**  },  {  "code": "135",  "name": "ICICI Corporate Bank",  "paymentGateWayId": "156",  "payModeDown": **false**  },  {  "code": "27",  "name": "DCB BANK Personal",  "paymentGateWayId": "42",  "payModeDown": **true**  },  {  "code": "32",  "name": "IndusInd Bank",  "paymentGateWayId": "51",  "payModeDown": **true**  },  {  "code": "98",  "name": "Janata Sahakari Bank Ltd. Pune",  "paymentGateWayId": "98",  "payModeDown": **true**  },  {  "code": "4538",  "name": "Ratnakar Bank Limited - Retail",  "paymentGateWayId": "186",  "payModeDown": **false**  },  {  "code": "4550",  "name": "HSBC Retail - Billdesk",  "paymentGateWayId": "196",  "payModeDown": **false**  },  {  "code": "44",  "name": "Ratnakar-DP Bank",  "paymentGateWayId": "67",  "payModeDown": **false**  },  {  "code": "38",  "name": "Citi Bank",  "paymentGateWayId": "58",  "payModeDown": **false**  },  {  "code": "129",  "name": "PNB Corporate Bank",  "paymentGateWayId": "132",  "payModeDown": **false**  },  {  "code": "46",  "name": "Central Bank of India",  "paymentGateWayId": "64",  "payModeDown": **false**  },  {  "code": "76",  "name": "IDBI Bank-Corporate",  "paymentGateWayId": "87",  "payModeDown": **false**  },  {  "code": "126",  "name": "Union Bank of India-Corporate",  "paymentGateWayId": "128",  "payModeDown": **false**  },  {  "code": "142",  "name": "COSMOS BANK - Retail",  "paymentGateWayId": "180",  "payModeDown": **false**  },  {  "code": "100",  "name": "Allahabad Bank - Retail",  "paymentGateWayId": "100",  "payModeDown": **false**  },  {  "code": "7",  "name": "United Bank of India",  "paymentGateWayId": "13",  "payModeDown": **false**  },  {  "code": "111",  "name": "punjab and maharashtra co-operative bank ltd",  "paymentGateWayId": "102",  "payModeDown": **false**  },  {  "code": "33",  "name": "Syndicate Bank",  "paymentGateWayId": "52",  "payModeDown": **true**  },  {  "code": "39",  "name": "Deutsche Bank",  "paymentGateWayId": "61",  "payModeDown": **false**  },  {  "code": "12",  "name": "Federal Bank",  "paymentGateWayId": "31",  "payModeDown": **false**  },  {  "code": "134",  "name": "Bharat Bank",  "paymentGateWayId": "155",  "payModeDown": **false**  },  {  "code": "4556",  "name": "Karnataka Bank-Corporate",  "paymentGateWayId": "229",  "payModeDown": **false**  },  {  "code": "48",  "name": "Indian Overseas Bank",  "paymentGateWayId": "65",  "payModeDown": **false**  },  {  "code": "54",  "name": "Kotak Bank",  "paymentGateWayId": "73",  "payModeDown": **false**  },  {  "code": "42",  "name": "Andhra Bank",  "paymentGateWayId": "63",  "payModeDown": **false**  },  {  "code": "13",  "name": "Catholic Syrian Bank",  "paymentGateWayId": "33",  "payModeDown": **false**  },  {  "code": "112",  "name": "SVC - CORPORATE",  "paymentGateWayId": "103",  "payModeDown": **false**  },  {  "code": "4540",  "name": "AU Small Finance Bank",  "paymentGateWayId": "187",  "payModeDown": **false**  },  {  "code": "4562",  "name": "HDFC Retail Bank-NBBL",  "paymentGateWayId": "236",  "payModeDown": **false**  },  {  "code": "47",  "name": "PNB Bank",  "paymentGateWayId": "59",  "payModeDown": **false**  },  {  "code": "4545",  "name": "IDFC Bank",  "paymentGateWayId": "191",  "payModeDown": **false**  },  {  "code": "80",  "name": "Saraswat Bank",  "paymentGateWayId": "90",  "payModeDown": **false**  },  {  "code": "16",  "name": "City Union Bank",  "paymentGateWayId": "36",  "payModeDown": **false**  },  {  "code": "4554",  "name": "Thane Janata Sahakari Bank",  "paymentGateWayId": "225",  "payModeDown": **false**  },  {  "code": "124",  "name": "SBIePay Test Bank - Corporate",  "paymentGateWayId": "126",  "payModeDown": **false**  },  {  "code": "53",  "name": "Lakshmi Vilas Bank",  "paymentGateWayId": "72",  "payModeDown": **false**  },  {  "code": "115",  "name": "Canara Bank - Retail",  "paymentGateWayId": "106",  "payModeDown": **false**  },  {  "code": "10",  "name": "Vijaya Bank",  "paymentGateWayId": "29",  "payModeDown": **false**  },  {  "code": "4539",  "name": "Ratnakar Bank Limited - Corporate",  "paymentGateWayId": "185",  "payModeDown": **false**  },  {  "code": "78",  "name": "HDFC Corporate Bank",  "paymentGateWayId": "89",  "payModeDown": **false**  },  {  "code": "1",  "name": "State Bank of India",  "paymentGateWayId": "86",  "payModeDown": **false**  },  {  "code": "31",  "name": "Corporation Bank",  "paymentGateWayId": "50",  "payModeDown": **false**  },  {  "code": "19",  "name": "Dena Bank",  "paymentGateWayId": "39",  "payModeDown": **false**  },  {  "code": "132",  "name": "Bank of Baroda",  "paymentGateWayId": "148",  "payModeDown": **false**  },  {  "code": "26",  "name": "Punjab and Sind Bank",  "paymentGateWayId": "41",  "payModeDown": **false**  },  {  "code": "4543",  "name": "AU Small Finance Bank - Corporate",  "paymentGateWayId": "189",  "payModeDown": **false**  },  {  "code": "30",  "name": "Tamilnad Mercantile Bank",  "paymentGateWayId": "45",  "payModeDown": **false**  },  {  "code": "99",  "name": "Andhra Bank - Corporate",  "paymentGateWayId": "99",  "payModeDown": **false**  },  {  "code": "4551",  "name": "Capital Small Finance Bank",  "paymentGateWayId": "200",  "payModeDown": **false**  },  {  "code": "143",  "name": "Axis Bank-Retail",  "paymentGateWayId": "181",  "payModeDown": **false**  },  {  "code": "114",  "name": "Allahabad Bank - Corporate",  "paymentGateWayId": "105",  "payModeDown": **false**  },  {  "code": "95",  "name": "Kotak Mahindra Bank",  "paymentGateWayId": "95",  "payModeDown": **false**  },  {  "code": "36",  "name": "ICICI DP Bank - Retail",  "paymentGateWayId": "56",  "payModeDown": **false**  },  {  "code": "14",  "name": "IDBI Bank-Retail",  "paymentGateWayId": "34",  "payModeDown": **false**  },  {  "code": "113",  "name": "SVC Co-operative Bank Ltd - Retail",  "paymentGateWayId": "104",  "payModeDown": **false**  },  {  "code": "4541",  "name": "Janata Sahakari Bank Ltd. Pune - Corporate",  "paymentGateWayId": "188",  "payModeDown": **false**  },  {  "code": "45",  "name": "Direct Pay",  "paymentGateWayId": "62",  "payModeDown": **false**  },  {  "code": "52",  "name": "Bank of India",  "paymentGateWayId": "71",  "payModeDown": **false**  },  {  "code": "15",  "name": "Indian Bank",  "paymentGateWayId": "35",  "payModeDown": **false**  },  {  "code": "9",  "name": "ING Vysya Bank - now Kotak",  "paymentGateWayId": "28",  "payModeDown": **false**  },  {  "code": "131",  "name": "ICICI Bank",  "paymentGateWayId": "147",  "payModeDown": **false**  },  {  "code": "40",  "name": "Bank of Bahrain and Kuwait",  "paymentGateWayId": "68",  "payModeDown": **false**  },  {  "code": "51",  "name": "YES Bank",  "paymentGateWayId": "69",  "payModeDown": **false**  },  {  "code": "4560",  "name": "JSB - Corporate Bank",  "paymentGateWayId": "232",  "payModeDown": **false**  },  {  "code": "4558",  "name": "RRB Bank",  "paymentGateWayId": "231",  "payModeDown": **false**  },  {  "code": "4547",  "name": "Kotak Mahindra Bank- Corporate",  "paymentGateWayId": "193",  "payModeDown": **false**  },  {  "code": "37",  "name": "HDFC Retail Bank",  "paymentGateWayId": "57",  "payModeDown": **false**  },  {  "code": "138",  "name": "Corporation-Corporate",  "paymentGateWayId": "174",  "payModeDown": **false**  },  {  "code": "139",  "name": "Axis Bank - Corporate",  "paymentGateWayId": "175",  "payModeDown": **false**  },  {  "code": "118",  "name": "Mehsana Urban Co. Op. Bank Ltd - Retail",  "paymentGateWayId": "114",  "payModeDown": **false**  },  {  "code": "11",  "name": "Test",  "paymentGateWayId": "30",  "payModeDown": **false**  },  {  "code": "137",  "name": "Central Bank of India-Corporate",  "paymentGateWayId": "165",  "payModeDown": **false**  },  {  "code": "97",  "name": "Axis Bank",  "paymentGateWayId": "97",  "payModeDown": **false**  },  {  "code": "150",  "name": "ESAF-Retail-Bank",  "paymentGateWayId": "184",  "payModeDown": **false**  },  {  "code": "43",  "name": "Canara Bank",  "paymentGateWayId": "60",  "payModeDown": **false**  },  {  "code": "35",  "name": "Dhanlaxmi Bank-Retail",  "paymentGateWayId": "53",  "payModeDown": **false**  },  {  "code": "20",  "name": "Bank of Maharashtra",  "paymentGateWayId": "40",  "payModeDown": **false**  },  {  "code": "75",  "name": "Dhanlaxmi Bank-Corporate",  "paymentGateWayId": "77",  "payModeDown": **false**  },  {  "code": "61",  "name": "UCO Bank",  "paymentGateWayId": "74",  "payModeDown": **false**  },  {  "code": "8",  "name": "Karur Vysya Bank",  "paymentGateWayId": "27",  "payModeDown": **true**  },  {  "code": "125",  "name": "Union Bank of India-Retail",  "paymentGateWayId": "127",  "payModeDown": **false**  },  {  "code": "28",  "name": "South Indian Bank",  "paymentGateWayId": "43",  "payModeDown": **false**  },  {  "code": "77",  "name": "ICICI DP Bank - Corporate",  "paymentGateWayId": "88",  "payModeDown": **false**  },  {  "code": "17",  "name": "Oriental Bank of Commerce",  "paymentGateWayId": "37",  "payModeDown": **false**  }  ],  "downtimeCount": 6  },  {  "payMode": "cards",  "subPayMode": "dc",  "downTimeDetails": [  {  "code": "MASTER",  "name": "State Bank of India WIBMO PG",  "paymentGateWayId": "226",  "payModeDown": **false**  },  {  "code": "VISA",  "name": "State Bank of India WIBMO PG",  "paymentGateWayId": "228",  "payModeDown": **false**  },  {  "code": "RUPAY",  "name": "State Bank of India WIBMO PG",  "paymentGateWayId": "227",  "payModeDown": **false**  }  ],  "downtimeCount": 0  },  {  "payMode": "UPI",  "downTimeDetails": [  {  "code": "UPI",  "name": "SBIePay Test Bank",  "paymentGateWayId": "133",  "payModeDown": **false**  },  {  "code": "QR",  "name": "SBIePay Test Bank",  "paymentGateWayId": "182",  "payModeDown": **false**  }  ],  "downtimeCount": 0  }  ],  "count": 5  } |

* Sample response: Error 200 Ok

{

"errors": [

{

"errorCode":"ERROR\_CODE ",

"errorMessage":"ERROR\_MSG"

},

],

"status":0

}

##### Business logic

* **Step 1: Downtime API controller**

Request reaches user downtime API controller and calls the downtime API service.

* **Step 2: Downtime API service**

Calluser a Downtime API DAO.

* **Step 3: Downtime API DAO**

Downtime API DAO calls Admin Client to fetch list of paymodes associated to merchant id.

* **Step 4: Downtime API service**

Send Success response otherwise send custom exception.

### Transaction - Recent Transaction

#### API Implementation

* **Endpoint**: /reporting/v1/ transaction/recent/{mId}
* **Method**: POST
* **Description:** This API allows you to retrieve the recent transaction data for a given merchant ID (mId) based on the specified frequency (DAILY/MONTHLY) and time. .
* **Request body:**

|  |  |  |  |
| --- | --- | --- | --- |
| Property | Description | Mandatory | Data Type |
| fromDate | From Date | Yes | string |
| toDate | To Date | Yes | String |
| frequency | frequency | No | String |

* **Path parameter:**
* **mId –** Merchant identifier
* Sample request body

|  |
| --- |
| {  "fromDate": "17-Feb-2025",  "toDate": "17-Mar-2025",  "frequency": "MONTHLY"  } |

* Sample response:

|  |
| --- |
| {  "status": 1,  "data": [  {  "totalTransactionCount": 34,  "totalOrderAmount": 55.08,  "totalRefundAmount": 2.7,  "totalTaxAmount": 39.14,  "totalNetSettlementAmount": 87.14,  "totalPendingSettlementAmount": 87.14,  "transactionDate": "Mar-2025"  }  ],  "count": 1,  "total": 1  } |

* **Response body:**
* Response - Success 200 Ok

{

“status”: 1,

“data”: [

{

"totalTransactionCount": number,

"totalAmount": number,

"transactionDate": string

}

],  
 "count": 1,  
 "total": 1,

}

* Response – Error 200 Ok

{

"status":0,

"error": [

{

"errorCode": String,

"errorMessage": String

}

]

}

##### Validation

* . Check if an mId is available and valid. If not, then return an error message.

##### Business Logic

* **Step-1: TransactionDashboard controller**

Request reaches controller and TransactionDailySummary service for further processing.

* **Step-2: Transaction Dashboard service:**
* Validate merchant, call external service to validate if merchant has access. Raise exception if no access found.
* Use DAO to fetch recent transaction data for today and prior day.
* Send response with current day settlement and past day settlement amount. For error exception send appropriate custom message.

##### Information

* Log Request

Log all incoming requests, including headers and body parameters (excluding sensitive information).

* Log Response

Log all responses, including generated token details and any error messages except the token.

##### Materialized View

**VIEW\_TRANSACTION\_DASHBOARD\_REPORT**

### Transaction – Download file for recent transaction

#### API Implementation

* **Endpoint**: /reporting/v1/transaction/download /{mId}
* **Method**: POST
* **Description:** This API allows you to download the recent transaction data for a given merchant ID (mId) based on the specified frequency (DAILY/MONTHLY) and time.
* **Request body:**

|  |  |  |  |
| --- | --- | --- | --- |
| Property | Description | Mandatory | Data Type |
| fromDate | From Date | Yes | string |
| toDate | To Date | Yes | String |

* **Path parameter:**
* mId –Merchant identifier
* Sample Request Body

|  |
| --- |
| {  "fromDate": "17-Feb-2025",  "toDate": "17-Mar-2025",  "frequency": "MONTHLY"  } |

* Sample response

|  |
| --- |
| Total Transaction Count,Total Order Amount,Total Refund Amount,Total Tax Amount,Total Net Settlement Amount,Total Settled Amount,Total Pending Settlement Amount,Transaction Date  34,  55.08,  2.7,  39.14,  87.14,  0.0,  87.14,Mar-2025 |

* **Response body:**
* Response - Success 200 Ok

{

“status”: 1,

“data”: [

{

"totalTransactionCount": number,

"totalAmount": number,

"transactionDate": string

}

],  
 "count": 1,  
 "total": 1,

}

* Response – Error 200 Ok

{

"status":0,

"error": [

{

"errorCode": String,

"errorMessage": String

}

]

}

##### Validation

* . Check if an mId is available and valid. If not, then return an error message.

##### Business Logic

* **Step-1: TransactionDashboard controller**

Request reaches controller and TransactionDailySummary service for further processing.

* **Step-2: Transaction Dashboard service**
* Validate merchant, call external service to validate if merchant has access. Raise exception if no access found.
* Use DAO to fetch recent transaction data for today and prior day.
* Build CSV file and send file as response with current day settlement and past day settlement amount. For error exception send appropriate custom message.

##### Information

* Log Request

Log all incoming requests, including headers and body parameters (excluding sensitive information).

* Log Response

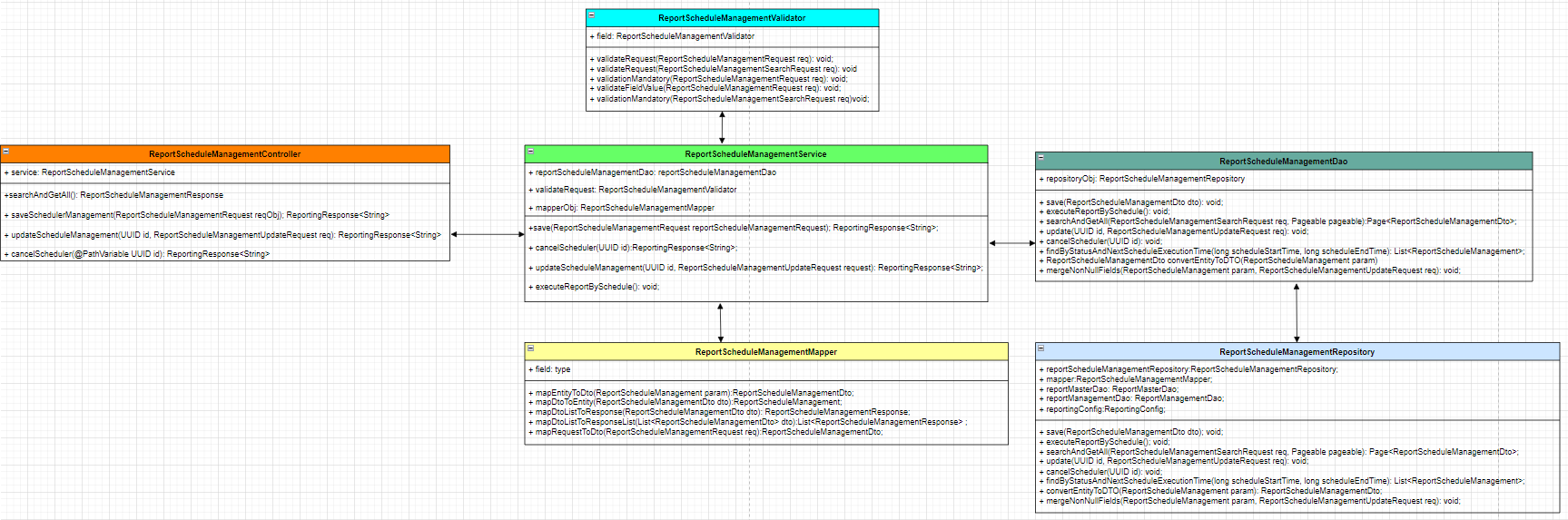
Log all responses, including generated token details and any error messages except the token.

##### Materialized View

VIEW\_TRANSACTION\_DASHBOARD\_REPORT

## Reports

### Class Diagram



#### Reports - Get All Reports

##### API Definition / Implementation

* **Endpoint**: /reporting/v1/report
* **Method**: GET
* **Description:** This API allows to display a list of all types of reports including order, settlement, transaction, refund, and changeback.
* **Request details:**
* **Authorization:** Bearer <Merchant Access Token>
* **Request body**: Empty
* **Response details:**
* **Response sample -** 200 Ok

{

“status”: 1,

“data”: [

{

“name”: string,

“description”: string,

“sequence”: number

}

],

“count”: 1

}

* Sample Response

|  |
| --- |
| {  "status": 1,  "data": [  {  "name": "ORDER",  "description": "All the Order Related Data Reports",  "sequence": 1  },  {  "name": "SETTLEMENTS",  "description": "All the Settlements Related Data Reports",  "sequence": 3  },  {  "name": "TRANSACTION",  "description": "All the Transaction Related Data Reports",  "sequence": 2  },  {  "name": "REFUNDS",  "description": "All the Refunds Related Data Reports",  "sequence": 4  },  {  "name": "CHARGEBACK",  "description": "All the Chargeback Related Data Reports",  "sequence": 5  }  ],  "count": 5  } |

* **Response sample –** Error 200 Ok

{

"errors": [

{

"errorCode":"ERROR\_CODE ",

"errorMessage":"ERROR\_MSG"

},

],

"status":0

}

|  |  |
| --- | --- |
| Error code | Error description |
| 7003 | Report is not found. |

##### Validation - NA

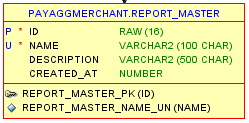
##### Business Logic

* **Step1: Report controller**

Request will reach report controller and call report service.

* **Step 2: Report service**
* Service calls report repository to fetch data.
* Create response using result coming from repository. Success or custom exception will be sent.

##### Database table - REPORT\_MASTER



##### Information

* Log Request

Log all incoming requests, including headers and body parameters (excluding sensitive information).

* Log Response

Log all responses, including generated token details and any error messages except the token.

### Reports – Create a Report

#### API Definition / Implementation

* **Endpoint**: /reporting/v1/report/management
* **Method**: POST
* **Description: :** This API provides capability to create a report for a particular mId with details such as type of reports (‘order’, ‘settlements’, ‘transaction’, ‘refunds’, ‘chargeback’ or ‘merchant fee’), duration, and file format..
* **Request details:**
* **Authorization:** Bearer <Merchant Access Token>
* **Request body**:

{

"report": "string",

"mId: "List Of MID"

"durationFromDate":"String",

"durationToDate":"String",

“reportFormat”: <csv or pdf or xls>

}

* **Sample request**

|  |
| --- |
| {  "durationFromDate": 1742910284376,  "durationToDate": 1742910284376,  "format": "CSV",  "report": "ORDER",  "mid": "1000003"  } |

* **Sample response**

|  |
| --- |
| {  "status": 1,  "data": [  "ReportManagementRequest received Successfully"  ]  } |

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Description** | **Mandatory** | **Data Type** |
| report | report name | Yes | string |
| mId | merchant ID. | Yes | string |
| durationFromDate | date range | Yes | Long |
| durationToDate | date range | Yes | Long |
| format | report format | Yes | string |

* **Response details:**
* **Response sample -** 200 Ok

{

"status": 1,

"data": [

"ReportManagementRequest received Successfully"

]

}

* Response sample – Error 200 Ok

{

"errors": [

{

"errorCode":"ERROR\_CODE ",

"errorMessage":"ERROR\_MSG"

},

],

"status":0

}

|  |  |
| --- | --- |
| Error code | Error description |
| 7001 | MID / report/ Duration fromDate/Duration toDate / Format is mandatory |
| 7002 | Invalid format. Valid values are CSV, XLS or PDF. |

#### Validation

* Check mandatory fields MID, Report, duration and format.
* Check for report type rquested, raise customer exception

#### Business Logic

* **Step1: Spring Security Filter**

It will filter the request and check if the token present passes the request to the token controller otherwise 401 error will throw. Captcha, Login, and OTP are on the whitelist.

* **Step 2: Report management controller**

Request will reach report controller and call report service.

* **Step 3: Report management service**

Service calls report validator to validate request body.

* **Step 4:** **Report management validator**

Validate report name, date range and mids.

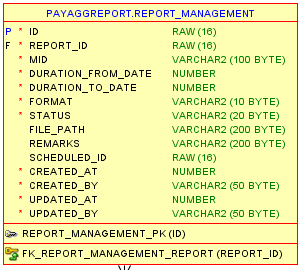
* **Step 5: Report management repository**

Repository will fetch data as per user provided inputs like mids, date range etc.

* **Step 6: Report management service**

Service willcreate response using result coming from repository. Success or custom exception will be sent.

#### Database Table - REPORT\_MANAGEMENT



#### Information

* Log Request

Log all incoming requests, including headers and body parameters (excluding sensitive information) to the database.

* Log Response

Log all responses, including generated token details and any error messages except the token.

### Reports - Get All Reports

#### API Definition / Implementation

* **Endpoint**: /reporting/v1/report/management/search
* **Method**: POST
* **Description:** This API provides capability to fetch list of all reports.
* **Request details:**
* **Authorization:** Bearer <Merchant Access Token>
* **Request body**:

{

"report": "string",

"mId: "string"

"durationFromDate":"String",

"durationToDate":"String",

“format”: <csv or pdf or xls>

}

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Description** | **Mandatory** | **Data Type** |
| report | report name | Yes | string |
| mId | merchant ID. | Yes | string |
| durationFromDate | date range | Yes | Long |
| durationToDate | date range | Yes | Long |
| format | report format | Yes | string |

* **Response details:**
* **Response sample -** 200 Ok

{

List Of Report Generation Data

}

* **Response sample –** Error 200 Ok

{

"errors": [

{

"errorCode":"ERROR\_CODE ",

"errorMessage":"ERROR\_MSG"

},

],

"status":0

}

Sample request

|  |
| --- |
| {  "durationFromDate": 1742910284376,  "durationToDate": 1742910284376,  "format": "CSV",  "report": "ORDER",  "mid": "1000003"  } |

Sample response

|  |
| --- |
| {  "status": 1,  "data": [  {  "report": "ORDER",  "durationFromDate": 1742910284376,  "durationToDate": 1742910284376,  "format": "CSV",  "status": "TO\_BE\_GENERATE",  "requestStateTime": 1742825556305,  "executionTime": 1742825556305,  "mid": "1000003"  }  ],  "count": 1,  "total": 1  } |

|  |  |
| --- | --- |
| Error code | Error description |
| 7003 | Report is not found. |

#### Validation

* Merchant identifier should be valid and active.

#### Business Logic

* **Step1: Report controller**

Request will reach report controller and call report service.

* **Step 2: Report service**

Service calls report repository to fetch data as per filter data.

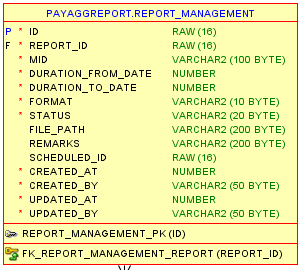
* **Step 3: Report repository**

Repository will fetch data as per user provided inputs like mids, date range etc.

* **Step 4: Report service**

Service will create response using result coming from repository. Success or custom exception will be sent.

#### Database Table - REPORT\_MANAGEMENT



#### Information

* Log Request

Log all incoming requests, including headers and body parameters (excluding sensitive information).

* Log Response

Log all responses, including generated token details and any error messages except the token.

### Reports - Schedule a report

#### API Definition / Implementation

* **Endpoint**: /reporting/v1/report/schedule/management
* **Method**: POST
* **Description:** Service saves a new report scheduler management request for generating or creating a report scheduler.
* **Request details:**
* **Authorization:** Bearer <Access Token>
* **Path Parameter:** The report name.
* **Request body**:

{

“report”: “string”,

"mId: "string"

"frequency":"<daily or monthly or yearly>",

"format": "string"

" scheduleExecutionTime ":"String",

}

* **Response details:**
* **Response sample -** 200 Ok

{

“status”: 1,

“data”: [

“ScheduleManagement Request received Successfully”

],

“count”: 1,

“total”: 1

}

* Response sample – Error 200 Ok

{

"errors": [

{

"errorCode":"ERROR\_CODE ",

"errorMessage":"ERROR\_MSG"

},

],

"status":0

}

Sample request

|  |
| --- |
| {  "format": "CSV",  "frequency": "DAILY",  "report": "ORDER",  "scheduleExecutionTime": "12:30 PM",  "mid": "1000003"  } |

Sample response

|  |
| --- |
| {  "status": 1,  "data": [  "ScheduleManagement Request received Successfully"  ]  } |

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Description** | **Mandatory** | **Data Type** |
| report | report name | Yes | string |
| mId | merchant ID. | Yes | string |
| frequency | frequency | Yes | String |
| scheduleExecutionTime | time for execution | Yes | String |
| format | report format | Yes | string |

#### Validation

* Check mandatory fields mId, report type, Schedule time, format type and valid frequency option.

#### Business Logic

* **Step1: Report controller**

Request will reach report controller and call report service.

* **Step 2: Report service**

Service calls report validator to validate incoming request.to fetch scheduled reports data as per filter data.

* **Step 3: Report validator**

Validate date range, MIDs and report name, if data invalid raise exception.

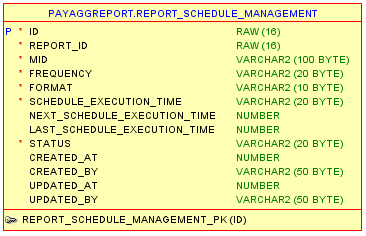
* **Step 4: Report repository**

Save report schedule with info: mids, generation time, frequency, format.

* **Step 5: Report service**

Service willcreate response using result coming from repository. Success or custom exception will be sent.

#### Database Table – REPORT\_SCHEDULER\_MANAGEMENT



#### Information

* Log Request

Log all incoming requests, including headers and body parameters (excluding sensitive information).

* Log Response

Log all responses, including generated token details and any error messages except the token.

### Reports – Update Scheduled Report

#### API Definition / Implementation

* **Endpoint**: /reporting/v1/report/schedule/management/{id}
* **Method**: PUT
* **Description:** This API provides capability to update a report based on frequency, format, and execution time. (more details required).
* **Request details:**
* **Authorization:** Bearer <Access Token>
* **Path Parameter:**

id - Scheduled report ID

* **Request body**:

{

"frequency":"String",

"format": "string",

"scheduleExecutionTime":"String"

}

* **Response details:**
* **Response sample -** 200 Ok

{

“status”: 1,

“data”: [

“updated Successfully”

]

“count”: 1,

“total”:1

}

* Response sample – Error 200 Ok

{

"errors": [

{

"errorCode":"ERROR\_CODE ",

"errorMessage":"ERROR\_MSG"

},

],

"status":0

}

Sample request: <https://dev.epay.sbi/reporting/v1>report/schedule/management/BE30FE7F-620C-4261-A763-C0794EC5D5D2

|  |
| --- |
| {  "frequency": "MONTHLY",  "format": "PDF",  "scheduleExecutionTime": "12:30 PM"  } |

Sample response

|  |
| --- |
| {  "data": [  "updated Successfully"  ],  "count": 1,  "total": 1  } |

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Description** | **Mandatory** | **Data Type** |
| frequency | frequency | No | Frequency  DAILY, MONTHLY, YEARLY |
| scheduleExecutionTime | next scheduled execution time. | No | String |
| format | report format | No | ReportFormat  CSV, XLS, PDF; |

#### Validation

* .Check report id exists, raise custom exception if report not found.

#### Business Logic

* **Step 1: Report controller**

Request will reach report controller and call report service.

* **Step 2: Report service**

Service calls report validator to validate incoming request.

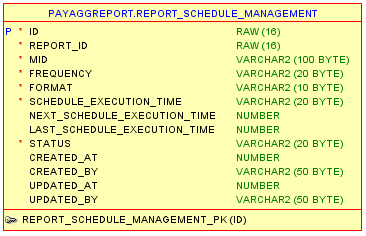
* **Step 4: Report repository**

Update report schedule with info like, frequency, format, scheduled execution time.

* **Step 5: Report service**

Service willcreate response using result coming from repository. Success or custom exception will be sent.

#### Database table – REPORT\_SCHEDULER\_MANAGEMENT



#### Information

* Log Request

Log all incoming requests, including headers and body parameters (excluding sensitive information).

* Log Response

Log all responses, including generated token details and any error messages except the token.

### Reports - Get All Schedule Report Request

#### API Definition / Implementation

* **Endpoint**: /reporting/v1/report/schedule/management/search
* **Method**: POST
* **Description:** This API provides capability to search for a list of reports based on search criteria including mId, frequency, report type, format, status, etc.
* **Request details:**
* **Authorization:** Bearer <Merchant Access Token>
* **Request body**:

{

"report": “string",

"mId: "string",

“frequency”: “string”,

“format”: "string”,

" scheduleStartExecutionTime ":"String",

" scheduleEndExecutionTime ":"String",

}

* **Response details:**
* **Response sample -** 200 Ok

{

“status”: 1,

“Data”: [

{

"scheduleRequestId": "string",

"report": <ORDER or SETTLEMENTS or TRANSACTION or REFUNDS or CHARGEBACK or FEES\_INVOICE or GST\_INVOICE>,

"mId": "string",

"format": "CSV, XLS, PDF"

"frequency: "DAILY, MONTHLY, YEARLY",

"scheduleExecutionTime": "",

"nextScheduleExecutionTime": long,

"lastScheduleExecutionTime": long,

"status": <TO\_BE\_START or IN\_QUEUE or CANCELLED>,

"requestStateTime": long,

"executionTime": long

}

]

}

* Response sample – Error 200 Ok

{

"errors": [

{

"errorCode":"ERROR\_CODE ",

"errorMessage":"ERROR\_MSG"

},

],

"status":0

}

Sample request example

|  |
| --- |
| {  "report": "ORDER",  "frequency": "MONTHLY",  "format": "CSV",  "scheduleStartExecutionTime": 1739255174000,  "scheduleEndExecutionTime": 1742880374000,  "mid": "1000003"  } |

Sample successs response example

|  |
| --- |
| {  "status": 1,  "data": [  {  "scheduleRequestId": "450a11b3-442d-490e-8e99-ead001a77950",  "report": "ORDER",  "format": "CSV",  "frequency": "DAILY",  "scheduleExecutionTime": "12:30 PM",  "nextScheduleExecutionTime": 1742905800000,  "status": "TO\_BE\_START",  "requestStateTime": 1742826569713,  "executionTime": 1742826569713,  "mid": "1000003"  },  {  "scheduleRequestId": "3902a131-7030-4986-8e5a-44e5aeb19e4f",  "report": "ORDER",  "format": "CSV",  "frequency": "DAILY",  "scheduleExecutionTime": "12:00 PM",  "nextScheduleExecutionTime": 1742904000000,  "lastScheduleExecutionTime": 1742815800008,  "status": "TO\_BE\_START",  "requestStateTime": 1742722299286,  "executionTime": 1742815800030,  "mid": "1000003"  },  {  "scheduleRequestId": "08d65a3a-1bd2-47c2-bed6-d2b94fbab863",  "report": "ORDER",  "format": "PDF",  "frequency": "YEARLY",  "scheduleExecutionTime": "12:30 PM",  "nextScheduleExecutionTime": 1774355400000,  "lastScheduleExecutionTime": 1742815800009,  "status": "TO\_BE\_START",  "requestStateTime": 1742722174204,  "executionTime": 1742815800032,  "mid": "1000003"  },  {  "scheduleRequestId": "ab0d70f6-1fef-4fcc-aa09-c8d28dc2e6e7",  "report": "ORDER",  "format": "XLS",  "frequency": "MONTHLY",  "scheduleExecutionTime": "1:30 PM",  "status": "CANCELLED",  "requestStateTime": 1742291421533,  "executionTime": 1742722240142,  "mid": "1000003"  },  {  "scheduleRequestId": "cda8fbec-86e1-4c85-a157-940ce899e670",  "report": "ORDER",  "format": "CSV",  "frequency": "DAILY",  "scheduleExecutionTime": "1:30 PM",  "nextScheduleExecutionTime": 1742909400000,  "lastScheduleExecutionTime": 1742821200012,  "status": "TO\_BE\_START",  "requestStateTime": 1738583509336,  "executionTime": 1742821200016,  "mid": "1000003"  }  ],  "count": 5,  "total": 5  } |

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Description** | **Mandatory** | **Data Type** |
| report | report name | No | string |
| mId | merchant ID. | Yes | string |
| frequency | Frequency | No | String |
| scheduleStartExecutionTime | date range | No | Long |
| scheduleEndExecutionTime | date range | No | Long |
| format | report format | No | string |

#### Validation - None

#### Business Logic

* **Step1: Spring Security Filter**

It will filter the request and check if the token present passes the request to the token controller otherwise 401 error will throw. Captcha, Login, and OTP are on the whitelist.

* **Step 2: Report Schedule controller**

Request will reach report controller and call report service.

* **Step 3: Report Schedule service**

Service calls report repository to fetch scheduled reports data as per filter data.

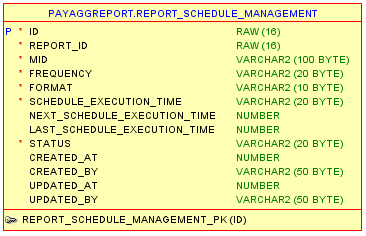
* **Step 4: Report schedule repository**

Repository will fetch data as per user provided inputs like mids, date range etc.

* **Step 5: Report schedule service**

Service will create response using result coming from repository. Success or custom exception will be sent.

#### Database table



#### Information

* Log Request

Log all incoming requests, including headers and body parameters (excluding sensitive information).

* Log Response

Log all responses, including generated token details and any error messages except the token.

### Reports – Download generated report for Merchant

#### API Definition / Implementation

* **Endpoint**: /reporting/v1/report/management/download/{mId}
* **Method**: POST
* **Description:** This APIallows to download the generated report for the given merchant id (mId) and save it at the specified location.
* **Request details:**
* **Authorization:** Bearer <Access Token>
* **Path Parameter:**

mId - merchant identifier

* **Request body**:

{

“filePath”: <AWS S3 bucket string>

}

* **Response –** Report file, format is one opted by user.

**Sample request**

|  |
| --- |
| {  "filePath": "C:\\Users\\V1014352\\Downloads\\1000003\_GST\_REPORT\_1742824740614.csv"  } |

**Sample response**

|  |
| --- |
| {  "status": 0,  "errors": [  {  "errorCode": "8003",  "errorMessage": "Report is not available to download."  }  ]  } |

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Description** | **Mandatory** | **Data Type** |
| filePath | file location | Yes | String |

#### Validation

* Check mandatory fields mId should be valid.
* Check file path exists.

#### Business Logic

* **Report management controller –** Request reaches controller, service is called to further process report download request.
* **Report management service –**
* Validate report download request.
* On successful validation send file as response.

#### Business logic - NA

#### Information

* Log Request

Log all incoming requests, including headers and body parameters (excluding sensitive information).

* Log Response

Log all responses, including generated token details and any error messages except the token.

### Reports – Cancel Report Scheduler

#### API Definition / Implementation

* **Endpoint**: /reporting/v1/report/schedule/management/cancel/{id}
* **Method**: PUT
* **Description:** This API provides capability to cancel a scheduled report? Need more details.
* **Request details:**
* **Authorization:** Bearer <Access Token>
* **Path Parameter:**

id - Scheduled report ID

* **Request body**: Empty
* **Response details:**
* **Response sample -** 200 Ok

{

“status”: 1,

“data”: [

“Schedule cancelled”

]

“count”: 1,

“total”:1

}

* Response sample – Error 200 Ok

{

"errors": [

{

"errorCode":"ERROR\_CODE ",

"errorMessage":"ERROR\_MSG"

},

],

"status":0

}

Sample Request

|  |
| --- |
| <https://dev.epay.sbi/reporting/v1>/report/schedule/management/cancel/:id? |
| Path variable: BE30FE7F-620C-4261-A763-C0794EC5D5D2 |

Sample Response

|  |
| --- |
| {  "data": [  "scheduler cancelled"  ]  } |

#### Validation

* Check report id exists, raise custom exception if report not found.

#### Business logic

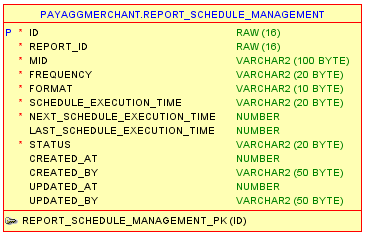
* **Step 1: Report controller**

Request will reach report controller and call report service.

* **Step 2: Report service**

Set scheduler status to cancelled and execution date to null.

#### Database table – REPORT\_SCHEDULER\_MANAGEMENT



#### Information

* Log Request

Log all incoming requests, including headers and body parameters (excluding sensitive information).

* Log Response

Log all responses, including generated token details and any error messages except the token.

### Reports – Scheduler

#### Recent Transaction

* **Endpoint**: NA
* **Method**: NA
* **Description:** This API helps user to create a scheduler for the desired reports of different types such as Order, Settlements, Transaction, Refunds, and Chargeback. This scheduler service runs every 30 minutes to find a report that needs to be scheduled.

#### Business logic

* **Step 1:** Report event scheduler

Runs every 30 minutes on scheduler management table. Checks for any report if any report generation needs to be generated.

* **Step 2:** Once entry found to report generation, report event will be created in report management table. Also, the event will be published to Kafka Queue with report event id.
* **Step 3:** Kafka consumer will read event from Queue and details like id, report type from DB will be used to call report generation logic specific to report type.
* **Step 4:** Report generation logicwill get data from database view for date range provided in scheduled report table. Once the file is created, the same will be pushed to cloud and path will be stored in DB and status will be marked Success.

#### Database tables:

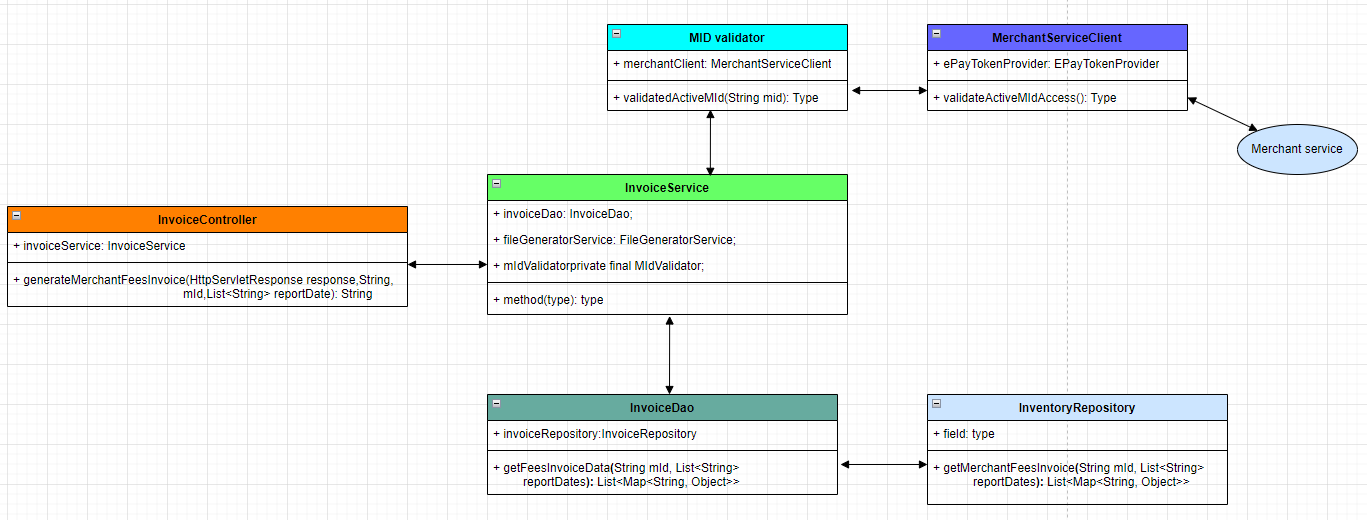
A. REPORT\_SCHEDULER\_MANAGMENT

B. REPORT\_MANAGMENT

## Invoice

### Report Download - Merchant Fees

#### Class Diagram



#### API Definition / Implementation

* **Endpoint**: /reporting/v1/invoice/fees/{mId}
* **Method**: POST
* **Description:** This API allows the Admin user to create the transaction fee invoice report and download the generated report in the PDF format based on the selected date and month.
* **Request details:**
* **Authorization:** Bearer <Merchant Access Token (Admin Portal Authorization)
* **Path Parameter:** The Merchant ID and report date.
* **Request body**:

|  |
| --- |
| {  “reportMonths”: [  “Dec-2024”, “Nov-2024”, “OCT-2024”  ]  } |

* **Response details:**
* Sample response: Success response - 200 Ok, One PDF file or Zip of PDF files for Merchant Fees **1000003\_Fees\_Report.zip**
* Sample response: Error response – 200 Ok. E.g.: Invalid Date Pattern or No report present for given date(s).

{

"errors": [

{

"errorCode":"ERROR\_CODE ",

"errorMessage":"ERROR\_MSG"

},

],

"status":0

}

|  |  |
| --- | --- |
| Error code | Error description |
| 8041 | PDF file generation failed. |

#### Validation

* Validate request for date list for format validity
* Merchant should be valid and active.

#### Business logic

* **Step1: Spring Security Filter**

It will filter the request and check if the token present passes the request to the token controller otherwise 401 error will throw. Captcha, Login, and OTP are on the whitelist.

* **Step 2: Invoice controller**

Request will reach Invoice controller and call Invoice service.

* **Step 3: Invoice service**

Service calls merchant Fees dao.

* **Step 4: Merchant fee DAO & Repository**

Report data will be fetched for a mid and month.

* **Step 5: Invoice service**

Set one PDF content or zip of multiples PDF in response to download.

**Note:** Generated PDF will be saved on cloud and file details will be available in database.

#### Database View

|  |  |  |
| --- | --- | --- |
| **VIEW\_TRANSACTION\_FEES\_INVOICE** | | |
| **Column** | **Data type** | **Nullable** |
| MID | VARCHAR2(100) | Yes |
| MERCHANT\_NAME | VARCHAR2(100) | No |
| ATRN\_NUM | VARCHAR2(255) | Yes |
| SBI\_ORDER\_REF\_NUMBER | VARCHAR2(255) | Yes |
| ORDER\_AMOUNT | NUMBER(15,2) | Yes |
| PAYMODE\_CODE | VARCHAR2(50) | Yes |
| PAY\_PROC\_ID | VARCHAR2(20) | Yes |
| PAYMENT\_GATEWAY | VARCHAR2(100) | Yes |
| MERCHANT\_FEE\_AMOUNT | NUMBER(16,2) | No |
| SERVICE\_TAX\_AMOUNT | NUMBER(16,2) | No |
| TRANSACTION\_DATE | DATE | Yes |

View derived from tables listed below.

|  |
| --- |
| **Table name** |
| TRANSACTION\_TXN |
| MERCHANT\_INFO\_MERCH |
| AGG\_MERCHANT\_ORDER\_HYBRID\_FEE\_DTLS\_TXN |

#### Information

* If valid, extract user details and set them in the security context.
* If invalid, reject the request with an unauthorized response.
* Log Request

Log all incoming requests, including headers and body parameters (excluding sensitive information) to the database.

* Log Response

Log all responses, including generated token details and any error messages except the token.

### Report Download - GST Invoice

#### API Definition / Implementation

* **Endpoint**: /reporting/v1/invoice/merchant/gst/{mId}
* **Method**: POST
* **Description:** This API allows the Admin user to create the GST invoice report and download the generated report in the PDF format based on the selected date and month..
* **Request details:**
* **Authorization:** Bearer <Merchant Access Token (Admin Portal Authorization)
* **Path Parameter:** The Merchant ID and report date.
* **Request body**:

{

“reportMonths”: [

“Dec-2024”, “Nov-2024”, “OCT-2024”

]

}

* **Response details:**
* Sample response: Success response - 200 Ok, One PDF file or Zip of PDF files for GST amount **1000003\_GST\_REPORT.zip**
* Sample response: Error response – 200 Ok. E.g.: Invalid Date Pattern or No report present for given date(s).

{

"errors": [

{

"errorCode":"ERROR\_CODE ",

"errorMessage":"ERROR\_MSG"

},

],

"status":0

}

#### Validation

* Validate request for date list for format validity
* Merchant should be valid and active.

#### Business Logic

* **Step1: Spring Security Filter**

It will filter the request and check if the token present passes the request to the token controller otherwise 401 error will throw. Captcha, Login, and OTP are on the whitelist.

* **Step 2: Invoice controller**

Request will reach Invoice controller and call Invoice service.

* **Step 3: Invoice service**

Service calls merchant GST dao.

* **Step 4: Merchant fee DAO & Repository**

Report data will be fetched from view for a mid and month.

* **Step 5: Invoice service**

Set one PDF content or zip of multiple PDF in response to download.

**Note:** Generated PDF will be saved on cloud and file details will be available in DB.

#### Database View - VIEW\_TRANSACTION\_GST\_INVOICE

#### Information

* If valid, extract user details and set them in the security context.
* If invalid, reject the request with an unauthorized response.
* Log Request

Log all incoming requests, including headers and body parameters (excluding sensitive information) to the database.

* Log Response

Log all responses, including generated token details and any error messages except the token.

## User Management - User Management

### User Management – Merchant User Profile

#### View User Profile

#### API Definition and Implementation

* **Endpoint:** /merchant/v1/user/profile
* **Method:** GET
* **Description:** This API allows an active merchant user to view the profile details including user name, first name, middle name, last name, mobile number, email Id, role, etc.
* **Request details:**
* Request header: Token contains valid username.

{  
 "Authorization": Bearer <Token with username and role>  
}

* **Response:**
* Example response – Success 200 Ok

{

"data": [

{

"userName": String,

"firstName": String,

"middleName": String,

"lastName": String,

"email": String,

"emailMasked": String,

"mobilePhone": String,

"mobilePhoneMasked": String,

"roleName": String

"createdAt": DATE,

" status" : String

}

],

"status":1,

"count": 1,

"total": 1

}

* Example response – 200 Ok with error

{

"errors": [

{

"errorCode":"ERROR\_CODE ",

"errorMessage":"ERROR\_MSG"

},

],

"status":0

}

Sample Success response

|  |
| --- |
| {  "status": 1,  "data": [  {  "userName": "Neeraj",  "firstName": "Neeraj",  "email": "[neeraj123@gmail.com](mailto:neeraj123@gmail.com)",  "emailMasked": "n\*\*\*\*\*\*\*23@\*\*\*\*\*\*\*om",  "mobilePhone": "7021408295",  "mobilePhoneMasked": "\*\*\*\*\*\*\*654",  "roleName": "ADMIN",  "createdAt": 1739428970245,  "status": "ACTIVE"  }  ],  "total": 1  } |

#### Validation

* Check Active status of merchant, otherwise raise custom exception.

#### Business logic

* **Step 1: Merchant User Service controller**

Request reach merchant user controller and call merchant user service.

* **Step 2: Merchant user DAO**

DAO will call merchant user repository.

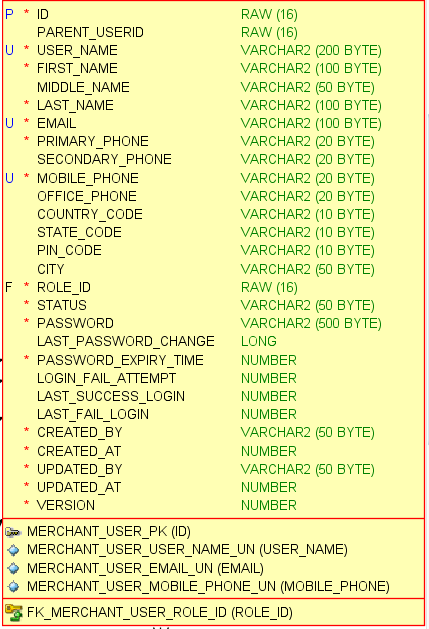
* **Step 3: Merchant user repository**

User profile will be fetched for user with active status.

* **Step 4: Merchant user service**

Send Success response with user profle otherwise send custom exception.

#### Database table – MERCHANT\_USER



#### Information

* + If valid, extract user details and set them in the security context.
  + If invalid, reject the request with an unauthorized response.
  + Log Request

Log all incoming requests, including headers and body parameters (excluding sensitive information) to the database.

* + Log Response

Log all responses, including generated token details and any error messages except the token.

### Update User profile

#### API Definition and implementation

* **Endpoint:** /merchant/v1/user/profile
* **Method:** PUT
* **Description:** This API allows an active merchant user to update or modify the profile details including user name, first name, middle name, last name, mobile number, email Id, role, etc.
* **Request details:**
* Request header: Token contains valid username.

{  
 "Authorization": Bearer <Token with username and role>  
}

* **Request body**:

{

"requestId": string

"userName": string

"firstName": string

"email": string

"mobilePhone": string

"role": UUID string

}

* **Response:**
* Example response – Success 200 Ok

{

“status”: 1,

“data”: [

{

"userName": string,

"firstName": string,

"middleName": string,

"lastName": string,

"email": string,

"emailMasked": string,

"primaryPhone": string,

"mobilePhone": string,

"mobilePhoneMasked": string,

"countryCode": string,

"stateCode": string,

"pinCode": string,

"city": string,

"roleName": string,

"status": string,

"remark": string,

"isFirstLogin": boolean,

"createdAt": long,

"updatedAt": long

}

]

}

Sample request

|  |
| --- |
| {  "email": "[neeraj123@gmail.com](mailto:neeraj123@gmail.com)",  "firstName": "Neeraj",  "mobilePhone": "7021408295",  "userName": "Neeraj",  "requestId": "d26889ac-4acc-4f23-b2f5-318d065889d8",  "role": "2bd0c275-b4df-f43f-e063-7c86b10a71ac"  } |

Sample response

|  |
| --- |
| {  "status": 0,  "errors": [  {  "errorCode": "1002",  "errorMessage": "OTP Verification is invalid. Reason: OTP Verification Fail"  }  ]  } |

* Example response – 200 Ok with error

{

"errors": [

{

"errorCode":"ERROR\_CODE ",

"errorMessage":"ERROR\_MSG"

},

],

"status":0

}

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Description** | **Mandatory** | **Data Type** |
| requestId | UUID request ID | No | string |
| userName | username | Yes | string |
| firstName | First name | Yes | string |
| email | email address | Yes | string |
| mobilePhone | Mobile number | Yes | string |
| role | role | No | UUID |

#### Validation

* Validate mandatory fields username, mobile phone, first name email.
* Also, regex validation in place for fields values.
* Check for uniqueness for login fields like username, mobile number and email.
* Check for request id validity.

#### Business Logic

* **Step 1:Merchant User Service controller**

Request reach merchant user controller and call merchant user service.

* **Step 2: Merchant user service**

Service call validator.

* **Step 3:** **Merchant user validator**

Service validates request for mandatory fields values, username is checked for if any existing username.

* **Step 4: Merchant user service**

Service call merchant user DAO.

* **Step 5: Merchant user DAO**

DAO will call merchant user repository.

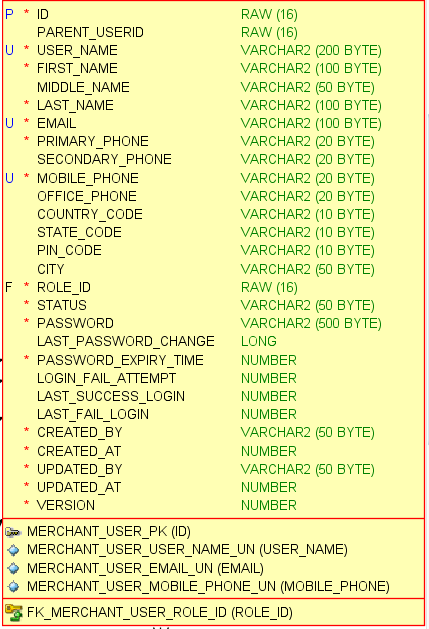
* **Step 6: Merchant user repository**

User profile will be saved.

* **Step 7: Merchant user service**

Send Success response otherwise send custom exception.

#### Database Table – MERCHANT\_USER



#### Information

* + If valid, extract user details and set them in the security context.
  + If invalid, reject the request with an unauthorized response.
  + Log Request

Log all incoming requests, including headers and body parameters (excluding sensitive information) to the database.

* + Log Response

Log all responses, including generated token details and any error messages except the token.

### Merchant Portal – Merchant User Access and Menu

#### Get User Access and Menu

#### API Definition and Implementation

* **Endpoint**: /merchant/v1/user/access/{userName}
* **Method**: GET
* **Description:** This API helps to view the merchant’s access permission’s details such as role, accessibility, etc. and the menu items including dashboard, report, user management, invoice, profile, etc.
* **Request Details:**

**Content-Type**: application/json

**Authorization (String):** Bearer <JWT Token>

* **Path Parameter:** userName or email or mobile number
* **Response:** Success: 200 – Ok

{

"status": 1,

"data": [

{

"assignedMenus": [

{

"id": "cb0e0651-f695-4863-afd7-cecfa5f12d9f",

"permissionId": "29efb3d8-8025-7f77-e063-7c86b10a914d",

"permissionCode": "3",

"userId": "9b026a9a-6c7d-4d53-b21c-a8dd5f94b2ec",

"menuId": "29efb3d8-8026-7f77-e063-7c86b10a914d",

"menuCode": "DASHBOARD",

"menuName": "Dashboard"

},

{

"id": "2304dc9f-29a6-4b84-9a3b-a4ee87b25d51",

"permissionId": "29efb3d8-8025-7f77-e063-7c86b10a914d",

"permissionCode": "3",

"userId": "9b026a9a-6c7d-4d53-b21c-a8dd5f94b2ec",

"menuId": "29efb3d8-8027-7f77-e063-7c86b10a914d",

"menuCode": "REPORT",

"menuName": "Report"

}

],

"unAssignedMenus": [

{

"id": "cb0e0651-f695-4863-afd7-cecfa5f12d9f",

"permissionId": "29efb3d8-8025-7f77-e063-7c86b10a914d",

"permissionCode": "3",

"userId": "9b026a9a-6c7d-4d53-b21c-a8dd5f94b2ec",

"menuId": "29efb3d8-8026-7f77-e063-7c86b10a914d",

"menuCode": "DASHBOARD",

"menuName": "Dashboard"

},

{

"id": "2304dc9f-29a6-4b84-9a3b-a4ee87b25d51",

"permissionId": "29efb3d8-8025-7f77-e063-7c86b10a914d",

"permissionCode": "3",

"userId": "9b026a9a-6c7d-4d53-b21c-a8dd5f94b2ec",

"menuId": "29efb3d8-8027-7f77-e063-7c86b10a914d",

"menuCode": "REPORT",

"menuName": "Report"

}

],

}

]

}

* Sample Success response

|  |
| --- |
| {  "data": [  {  "assignedMenus": [  {  "menuCode": "DASHBOARD",  "menuName": "Dashboard",  "permissionCode": "3"  },  {  "menuCode": "REPORT",  "menuName": "Report",  "permissionCode": "3"  },  {  "menuCode": "TRX\_REFUND",  "menuName": "Transaction & Refund",  "permissionCode": "3"  },  {  "menuCode": "INVOICE",  "menuName": "Invoice",  "permissionCode": "3"  },  {  "menuCode": "USER\_MGMT",  "menuName": "User Management",  "permissionCode": "3"  },  {  "menuCode": "ACC\_SETTING",  "menuName": "Account & Settings",  "permissionCode": "3"  },  {  "menuCode": "HELP\_SUPPORT",  "menuName": "Help & Support",  "permissionCode": "3"  },  {  "menuCode": "PROFILE",  "menuName": "Profile",  "permissionCode": "3"  },  {  "menuCode": "MANAGE\_USERS",  "menuName": "Manage Users",  "permissionCode": "3"  },  {  "menuCode": "ACTIVE\_USERS",  "menuName": "Active Users",  "permissionCode": "3"  },  {  "menuCode": "MID\_MAPPING",  "menuName": "MID Mapping",  "permissionCode": "3"  },  {  "menuCode": "TOOL\_MAPPING",  "menuName": "Tool Mapping",  "permissionCode": "3"  }  ],  "unAssignedMenus": []  }  ]  } |

* **Response:** Error: 200 – Ok

{

"errors": [

{

"errorCode":"ERROR\_CODE ",

"errorMessage":"ERROR\_MSG"

},

],

"status":0

}

#### Validation

* Given user should be present and status should be active.
* Logged user should be parent of merchant user.

#### Business Logic

* **Step 1**: **Merchant user controller**

Request reach merchant user controller and call Merchant user service.

* **Step 2**: **Merchant user service**

Service validates request for mandatory fields, call DAO to fetch user. If no user found send Error response.

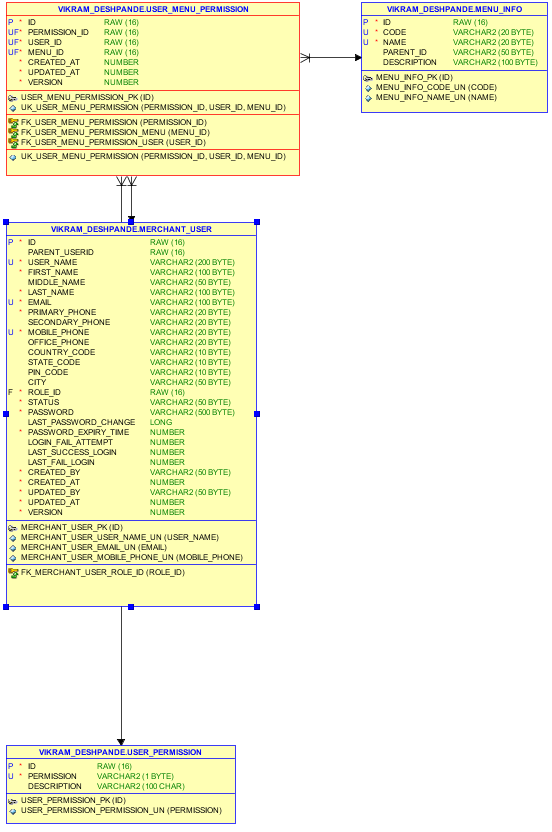
* **Step 3**: **Merchant user DAO**

Pull merchant user details and menu / permissions.

* **Step 4**: **Merchant user service**

Send Successful response with user details and menu / permissions otherwise send custom exception.

#### Database Table – USER\_MENU\_PERMISSION



#### Information

* + Log Request

Log all incoming requests, including headers and body parameters (excluding sensitive information) to the database.

* + Log Response

Log all responses, including generated token details and any error messages except the token.

### Assign User Access and Menu

#### API Definition and Implementation

* **Endpoint**: /merchant/v1/user/access/{userName}
* **Method**: POST
* **Description:** This API assigns menu items and access rights to the merchant users.
* **Request Details:**

**Content-Type**: application/json

**Authorization (String):** Bearer <JWT Token>

* **Path Parameter:** userName or email or mobile number
* **Request body:**

{

“assignedMenus”: [ “DASHBOARD”, “REPORT“]

}

* **Response:** Success: 200 – Ok

{

"status": 1,

"data": [

{

"userAccess": [

{

"permissionCode": "3",

"menuCode": "DASHBOARD",

"menuName": "Dashboard"

},

{

"permissionCode": "3",

"menuCode": "REPORT",

"menuName": "Report"

}

]

}

]

}

* Sample request

|  |
| --- |
| [  "DASHBOARD",  "REPORT"  ] |

* Sample response

|  |
| --- |
| {  "data": [  "Menus is assigned Successfully"  ]  } |

* **Response:** Error: 200 – Ok

{

"errors": [

{

"errorCode":"ERROR\_CODE ",

"errorMessage":"ERROR\_MSG"

},

],

"status":0

}

#### Validation

* Given user should be present with active status.
* Check the list of menus, raise custom exception if empty.
* Logged in user should be parent of user for whom request been processed.
* Menu list should match with menu access provided to parent user.

#### Business Logic

* **Step 1: Merchant user controller**

Request reach merchant user controller and call Merchant user service.

* **Step 2: Merchant user service**

Service calls validator for request content.

* **Step 3:** **Merchant user validator**

Validates request for mandatory fields, If no user found send Error response.

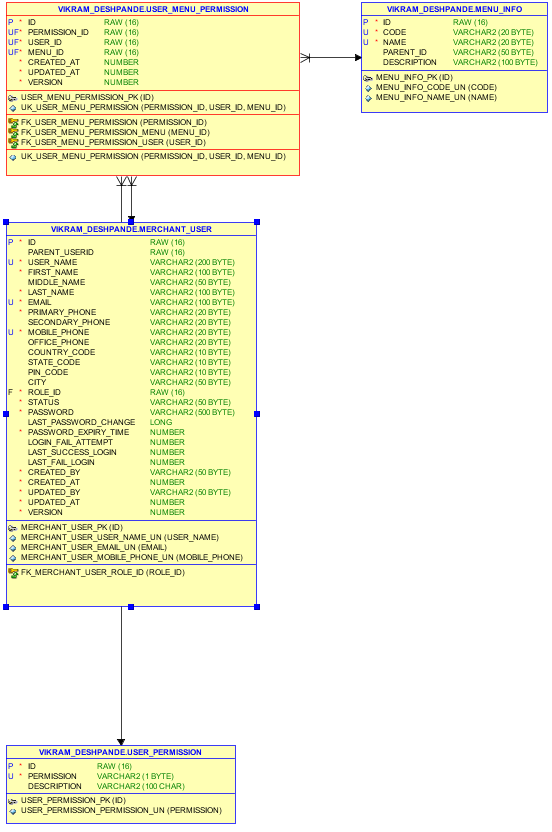
* **Step 4: Merchant user DAO**

Pull merchant user details and menu / permissions. Compare existing assigned with incoming menu request to add update user menus.

* **Step 5: Merchant user service**

Send successful response with user details and menu / permissions otherwise send custom exception.

#### Database Table – USER\_MENU\_PERMISSION



#### Information

* + If valid, extract user details and set them in the security context.
  + If invalid, reject the request with an unauthorized response.
  + Log Request

Log all incoming requests, including headers and body parameters (excluding sensitive information) to the database.

* + Log Response

Log all responses, including generated token details and any error messages except the token.

### Get User Access and Menu

#### API Definition and Implementation

* **Endpoint**: /merchant/v1/user/access
* **Method**: GET
* **Description:** This API helps to view the profile details of the logged-in merchant users and their access permissions.                                                permission.
* **Request Details:**
  + **Content-Type**: application/json
  + **Authorization (String):** Bearer <JWT Token>
* **Path Parameter:**No parameter
* **Response:** Success: 200 – Ok

{

  "status": 1,

  "data": [

    {

      "userAccess": [

        {

          "id": "cb0e0651-f695-4863-afd7-cecfa5f12d9f",

          "permissionId": "29efb3d8-8025-7f77-e063-7c86b10a914d",

          "permissionCode": "3",

          "userId": "9b026a9a-6c7d-4d53-b21c-a8dd5f94b2ec",

          "menuId": "29efb3d8-8026-7f77-e063-7c86b10a914d",

          "menuCode": "DASHBOARD",

          "menuName": "Dashboard"

        },

        {

          "id": "2304dc9f-29a6-4b84-9a3b-a4ee87b25d51",

          "permissionId": "29efb3d8-8025-7f77-e063-7c86b10a914d",

          "permissionCode": "3",

          "userId": "9b026a9a-6c7d-4d53-b21c-a8dd5f94b2ec",

          "menuId": "29efb3d8-8027-7f77-e063-7c86b10a914d",

          "menuCode": "REPORT",

          "menuName": "Report"

        }

          ],

      "userInfo": {

        "userName": "String",

        "firstName": "String",

        "email": "abchj@xyx.com",

        "emailMasked": "abcXXX@xyx.com",

        "mobilePhone": "1234567890",

“mobilePhoneMasked”: “12345xxxxx”,

“remark”: “Test remark”,

        "roleName": "User",

        "status": "ACTIVE",

      “loginFailAttempt” : number,

        "passwordExpiryTime": 1742731665682,

        "createdBy": "SBI EPay",

        "createdAt": 1734955665724,

        "updatedBy": "SBI EPay",

        "updatedAt": 1734955665724,

        "isFirstLogin": "true",

        "lastSuccessLogin": 1734955665724

      }

    }

  ]

}

* **Response:** Error: 200 – Ok

{

  "errors": [

    {

      "errorCode":"ERROR\_CODE ",

      "errorMessage":"ERROR\_MSG"

    },

  ],

"status":0

}

Sample Success response

|  |
| --- |
| {  "status": 1,  "data": [  {  "userAccess": [  {  "menuCode": "DASHBOARD",  "menuName": "Dashboard",  "permissionCode": "3"  },  {  "menuCode": "REPORT",  "menuName": "Report",  "permissionCode": "3"  },  {  "menuCode": "TRX\_REFUND",  "menuName": "Transaction & Refund",  "permissionCode": "3"  },  {  "menuCode": "INVOICE",  "menuName": "Invoice",  "permissionCode": "3"  },  {  "menuCode": "USER\_MGMT",  "menuName": "User Management",  "permissionCode": "3"  },  {  "menuCode": "ACC\_SETTING",  "menuName": "Account & Settings",  "permissionCode": "3"  },  {  "menuCode": "HELP\_SUPPORT",  "menuName": "Help & Support",  "permissionCode": "3"  },  {  "menuCode": "PROFILE",  "menuName": "Profile",  "permissionCode": "3"  },  {  "menuCode": "MANAGE\_USERS",  "menuName": "Manage Users",  "permissionCode": "3"  },  {  "menuCode": "ACTIVE\_USERS",  "menuName": "Active Users",  "permissionCode": "3"  },  {  "menuCode": "MID\_MAPPING",  "menuName": "MID Mapping",  "permissionCode": "3"  },  {  "menuCode": "TOOL\_MAPPING",  "menuName": "Tool Mapping",  "permissionCode": "3"  }  ],  "userInfo": {  "userName": "Neeraj",  "firstName": "Neeraj",  "email": "[neeraj123@gmail.com](mailto:neeraj123@gmail.com)",  "emailMasked": "n\*\*\*\*\*\*\*23@\*\*\*\*\*\*\*om",  "mobilePhone": "7021408295",  "mobilePhoneMasked": "\*\*\*\*\*\*\*654",  "roleName": "ADMIN",  "status": "ACTIVE",  "passwordExpiryTime": 1750142347526,  "createdBy": "Neeraj1",  "createdAt": 1739428970245,  "updatedBy": "anonymousUser",  "updatedAt": 1742889881199,  "lastSuccessLogin": 1742889881198,  "firstLogin": **false**  }  }  ]  } |

#### Validation

* Header should have the valid JWT token.
* JWT token should have the login user details.
* Given user should be valid and active.

#### Business Logic

* **Step 1: Merchant user controller**

Request reach merchant user controller and call Merchant user service.

* **Step 2: Merchant user service**

Service validates status of logged in user.

* **Step 3: Merchant user DAO**

Pull merchant user details and menu / permissions.

* **Step 4: Merchant user service**

Send successful response with user details and menu / permissions otherwise send custom exception.

#### Database Table – USER\_MENU\_PERMISSION

#### Information

* Log Request

Log all incoming requests, including headers and body parameters (excluding sensitive information) to the database.

* Log Response

Log all responses, including generated token details and any error messages except the token.

### Merchant Portal: User Management View

#### Profile Tab – Create Users

##### API definition and implementations

* **Endpoint**: /merchant/v1/user
* **Method:** POST
* **Description:** This API helps Admin or Super Admin to create merchant users.
* **Request details:** 
  + Path Parameter: none
  + Headers: Requires valid Merchant User Login Token.

**Authorization:** Bearer <Merchant login token>

* + Request object:

{

"userName": string,

"firstName": string,

"middleName": string,

"lastName": string,

"email": string,

"primaryPhone": number,

"secondaryPhone": number,

"mobilePhone": number,

"officePhone": number,

"countryCode": string,

"stateCode": string,

"pinCode": number,

"city": string,

"role": string,

"assignedMId": List<String>

}

* **Response**: Success: 200 – Ok

{

data: [

{

"userName": string,

"firstName": string,

"middleName": string,

"lastName": string,

"email": string,

"primaryPhone": number,

"secondaryPhone": number,

"mobilePhone": number,

"officePhone": number,

"countryCode": string,

"stateCode": string,

"pinCode": number,

"city": string,

"role": string,

"assignedMIds": List<String>

}

],

"status”: 1,

"count": 1,

"total": 1

}

* **Response**: Error 200 – Ok

{

"status":0,

"error": [

{

"errorCode": String,

"errorMessage": String

}

]

Sample request

|  |
| --- |
| {  "email": "[shikha123@gmail.com](mailto:shikha123@gmail.com)",  "firstName": "Shikha",  "mobilePhone": "78965412304",  "userName": "Shikhas",  "requestId": "d26889ac-4acc-4f23-b2f5-318d065889d8",  "role": "2E0075E9-A83D-9DB7-E063-7C86B10ADCA3"  } |

Sample Success response

|  |
| --- |
| {  "status": 1,  "data": [  {  "userName": "Shikhas",  "firstName": "Shikha",  "email": "[shikha123@gmail.com](mailto:shikha123@gmail.com)",  "emailMasked": "s\*\*\*\*\*\*\*23@\*\*\*\*\*\*\*om",  "mobilePhone": "78965412304",  "mobilePhoneMasked": "\*\*\*\*\*\*\*304",  "roleName": "ADMIN",  "status": "ACTIVE",  "createdAt": 1742894189194,  "updatedAt": 1742894189194,  "firstLogin": **true**  }  ],  "total": 1  } |

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Description** | **Mandatory** | **Data Type** |
| requestId | UUID request ID | No | string |
| userName | username | Yes | string |
| firstName | First name | Yes | string |
| email | email address | Yes | string |
| mobilePhone | Mobile number | Yes | string |
| role | role | No | UUID |

##### Validation

* Given user should be present.
* Check for mandatory fields user name, email and phone number.
* Request should have valid request id.

##### Business Logic

* **Step 1: Merchant user controller**

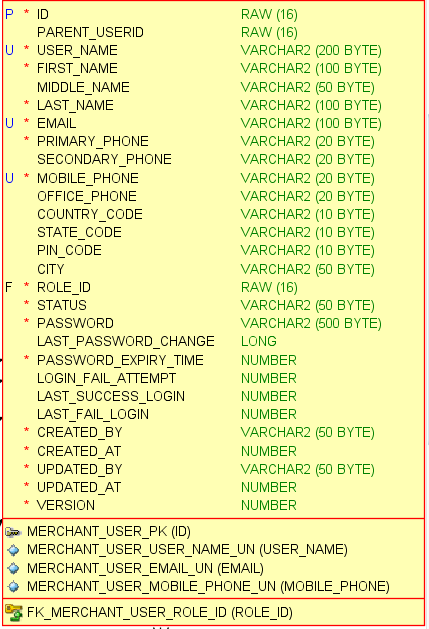
Request reach merchant user controller.

* **Step 2: Merchant user service –** Service validates request for mandatory fields user name, email phone number.

**Step 3: Merchant user DAO –** On successful validation save User.

**Step 4: Merchant user service –** Send successful response with user having MIDs and otherwise send custom exception.

##### Database table – MERCHANT\_USER



##### Information

* + If valid, extract user details and set them in the security context.
  + If invalid, reject the request with an unauthorized response.
  + Log Request

Log all incoming requests, including headers and body parameters (excluding sensitive information) to the database.

* + Log Response

Log all responses, including generated token details and any error messages except the token.

#### Profile Tab Update Users

##### API definition and Implementations

* **Endpoint**: /merchant/v1/user
* **Method:** PUT
* **Description:** This API allows Admin or Super Admin to create merchant users.
* **Request details:** 
  + Path Parameter: type (values: API or Encryption)
  + Headers: Requires valid Merchant User Login Token.

**Authorization:** Bearer <Merchant login token>

* + Request object:

{

"userName": string,

"firstName": string,

"middleName": string,

"lastName": string,

"email": string,

"primaryPhone": number,

"secondaryPhone": number,

"mobilePhone": number,

"officePhone": number,

"countryCode": string,

"stateCode": string,

"pinCode": number,

"city": string,

"role": string,

"assignedMId": List<String>

}

* **Response**: Succes: 200 – Ok

{

data: [

{

"userName": string,

"firstName": string,

"middleName": string,

"lastName": string,

"email": string,

"primaryPhone": number,

"secondaryPhone": number,

"mobilePhone": number,

"officePhone": number,

"countryCode": string,

"stateCode": string,

"pinCode": number,

"city": string,

"role": string,

"assignedMIds": List<String>

}

],

"status”: 1,

"count": 1,

"total": 1

}

* **Response**: Error 200 – Ok

{

"status":0,

"error": [

{

"errorCode": String,

"errorMessage": String

}

]

}

Sample request

|  |
| --- |
| {  "email": "[shikha1234@gmail.com](mailto:shikha1234@gmail.com)",  "firstName": "Shikha",  "mobilePhone": "9876541304",  "userName": "Shikhas",  "requestId": "d26889ac-4acc-4f23-b2f5-318d065889d8",  "role": "2E0075E9-A83D-9DB7-E063-7C86B10ADCA3"  } |

Sample response

|  |
| --- |
| {  "status": 1,  "data": [  {  "userName": "Shikhas",  "firstName": "Shikha",  "email": "[shikha1234@gmail.com](mailto:shikha1234@gmail.com)",  "emailMasked": "s\*\*\*\*\*\*\*\*34@\*\*\*\*\*\*\*\*om",  "mobilePhone": "9876541304",  "mobilePhoneMasked": "\*\*\*\*\*\*\*304",  "roleName": "ADMIN",  "status": "ACTIVE",  "createdAt": 1742894189194,  "updatedAt": 1742894487538,  "firstLogin": **true**  }  ],  "total": 1  } |

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Description** | **Mandatory** | **Data Type** |
| requestId | UUID request ID | No | string |
| userName | username | Yes | string |
| firstName | First name | Yes | string |
| email | Email Address | Yes | string |
| mobilePhone | Mobile number | Yes | string |
| role | role | No | UUID |

##### Validation

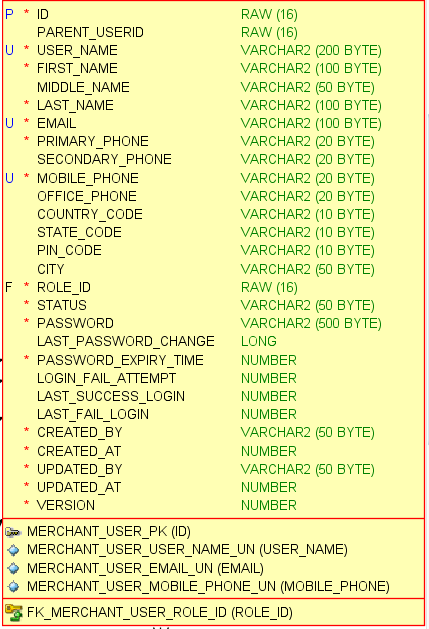
* Given user should be present.
* Check for mandatory fields user name, email and phone number.
* Request should have valid request id.

##### Business Logic

* **Step 1: Merchant user controller**

Request reach merchant user controller. **Step 2: Merchant user service**Service validates request for mandatory fields user name, email phone number.**Step 3: Merchant user DAO** On successful validation update user.**Step 4: Merchant user service**Send successful response with user having MIDs and otherwise send custom exception.

##### Database Table – MERCHANT\_USER



##### Information

* + If valid, extract user details and set them in the security context.
  + If invalid, reject the request with an unauthorized response.
  + Log Request

Log all incoming requests, including headers and body parameters (excluding sensitive information) to the database.

* + Log Response

Log all responses, including generated token details and any error messages except the token.

### Manage Users

#### API definition and Implementations

* **Endpoint**: /merchant/v1/user/search
* **Method:** POST
* **Description:**
* This API allows you to filter or search the merchant users based on their user name, mobile number, creation date, and status..
* **Request details:** 
  + Path Parameter: type (values: API or Encryption)
  + Headers: Requires valid Merchant User Login Token.

**Authorization:** Bearer <Merchant login token>

* + **Request object**:

{

"userName": string

"mobileNo": number

"dateRange:{

"fromDate": <date>

"toDate": <date>

},

"status": string

}

* **Response:** Success 200 – Ok

{

data: [

{

"userName": string,

"firstName": string,

"middleName": string,

"lastName": string,

"email": string,

"primaryPhone": number,

"secondaryPhone": number,

"mobilePhone": number,

"officePhone": number,

"countryCode": string,

"stateCode": string,

"pinCode": number,

"city": string,

"role": string,

"assignedMIds": List<String>

},

{

"userName": string,

"firstName": string,

"middleName": string,

"lastName": string,

"email": string,

"primaryPhone": number,

"secondaryPhone": number,

"mobilePhone": number,

"officePhone": number,

"countryCode": string,

"stateCode": string,

"pinCode": number,

"city": string,

"role": string,

"assignedMIds": List<String>

}

],

"status: 1,

"count": 2,

"total": 2

}

* **Response:** Error 200 – Ok

{

"status":0,

"error": [

{

"errorCode": String,

"errorMessage": String

}

]

}

Sample request

|  |
| --- |
| {  "userName": "ravi",  "name": "Ravi",  "mobilePhone": "8791602346",  "fromDate": "1740120741608",  "toDate": "1740120741608",  "status": "ACTIVE"  } |

Sample response

|  |
| --- |
| {  "status": 1,  "data": [  {  "userName": "ravi",  "firstName": "Ravi",  "email": "[ravi.rathore@cedge.in](mailto:ravi.rathore@cedge.in)",  "emailMasked": "r\*\*\*\*\*\*\*\*\*\*re@\*\*\*\*\*\*\*\*\*\*in",  "mobilePhone": "8791602346",  "mobilePhoneMasked": "\*\*\*\*\*\*\*346",  "roleName": "ADMIN",  "status": "INACTIVE",  "remark": "[STATUS UPDATE from ACTIVE to INACTIVE ]Remark:ok D",  "createdAt": 1740043870512,  "updatedAt": 1742633194657,  "firstLogin": **false**  }  ],  "count": 1,  "total": 1  } |

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Description** | **Mandatory** | **Data Type** |
| name | UUID request ID | No | string |
| userName | username | No | string |
| fromDate | First name | No | Long |
| toDate | email address | No | Long |
| mobilePhone | Mobile number | No | string |
| status | status | No | string |

#### Validation

##### Request Data Validation

* Header should have the valid JWT token.
* JWT token should have the Merchant as a role.

##### Business data validation

* Given user should be present.
* Status value should be correct.

##### Business Logic

* + **Step 1: Merchant user controller**

Request reach merchant user controller and call merchant user service.

* + **Step 2: Merchant user service**

Service validates request for mandatory fields, call DAO to check login user if active.

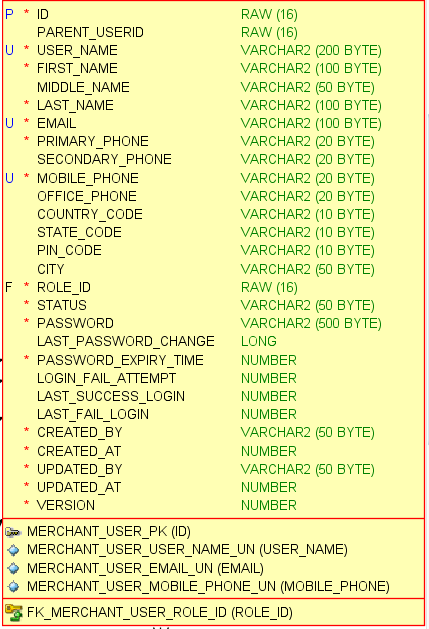
* + **Step 3: Merchant user DAO**

Pull data with filter values.

* + **Step 4: Merchant user service**

Send Successful response with user details and menu / permissions otherwise send custom exception.

##### Database Table – MERCHANT\_USER



##### Information

* + If valid, extract user details and set them in the security context.
  + If invalid, reject the request with an unauthorized response.
  + Log Request: Log all incoming requests, including headers and body parameters (excluding sensitive information) to the database.
  + Log Response: Log all responses, including generated token details and any error messages except the token.

### User Status Update

#### API Definition and Implementations

* **Endpoint**: /merchant/v1/user/status
* **Method:** PUT
* **Description:** This API helps to update the status of merchant users such as active, inactive, blocked, expired..
* **Request details:** 
  + Path Parameter: None
  + Headers: Requires valid Merchant User Login Token.

**Authorization:** Bearer <Merchant login token>

* Request body:

{

"userName": "string",

"userStatus": <status\_string>,

"remark"; "string"

}

* **Response details**
* Sample response – Success 200 Ok

{

"status":1 ,

"data": [

{

"userName":"string"

"firstName":"string"

"middleName":"string"

"lastName":"string"

"email":"string"

"primaryPhone":"string"

"mobilePhone":"string"

"countryCode":"string"

"stateCode":"string"

"pinCode":"string"

"city":"string"

"role":"string"

"status":"string"

"remark":"string"

"createdAt":"string"

"updatedAt":"string"

}

]

}

* Sample response – Error 200 Ok

{

  "errors": [

    {

      "errorCode":"ERROR\_CODE ",

      "errorMessage":"ERROR\_MSG"

    },

  ],

"status":0

}

Sample request

|  |
| --- |
| {  "remark": "User status updated Successfully",  "userName": "ravi",  "userStatus": "ACTIVE"  } |

Sample response

|  |
| --- |
| {  "status": 1,  "data": [  {  "userName": "ravi",  "firstName": "Ravi",  "email": "[ravi.rathore@cedge.in](mailto:ravi.rathore@cedge.in)",  "emailMasked": "r\*\*\*\*\*\*\*\*\*\*re@\*\*\*\*\*\*\*\*\*\*in",  "mobilePhone": "8791602346",  "mobilePhoneMasked": "\*\*\*\*\*\*\*346",  "status": "ACTIVE",  "remark": "[STATUS UPDATE from INACTIVE to ACTIVE ]Remark:User status updated Successsfully",  "createdAt": 1740043870512,  "updatedAt": 1742895908913,  "firstLogin": **false**  }  ],  "total": 1  } |

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Description** | **Mandatory** | **Data Type** |
| userName | username | Yes | string |
| remark | remark | Yes | string |
| userStatus | status | No | UserStatus  *ACTIVE*, *INACTIVE*, *BLOCKED*, *EXPIRED*; |

#### Validation

* Token should have the Admin or Super admin as a role.
* Given username should be present.
* Remark length should not exceed 200 characters.
* Status value should be valid listed one (ACTIVE, INACTIVE, BLOCKED, EXPIRED).

#### Business Logic

* **Step 1: Merchant user controller**

Request reach merchant user controller and call merchant user service.

* **Step 2: Merchant user service**

Service calls validator service.

* **Step 3: Merchant user Validator**

Validator checks username, mandatory fields and remark word length.

* **Step 4: Merchant user service**

On successful validation, call merchant user DAO.

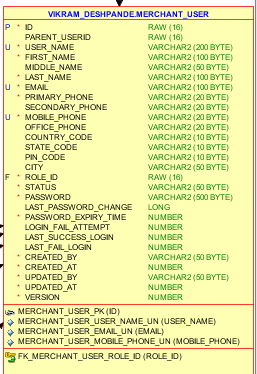
* **Step 5: Merchant user DAO**

DAO update willmerchant user with incoming status.

* **Step 6: Merchant user service**

Send successful response otherwise send custom exception.

#### Database table – MERCHANT\_USER



#### Information

* + Log Request

Log all incoming requests, including headers and body parameters (excluding sensitive information) to the database.

* + Log Response

Log all responses, including generated token details and any error messages except the token.

### Assign - Merchant User and MIDs mapping

#### API definition and Implementations

* **Endpoint**: /merchant/v1/user/mid/mapping/{userName}
* **Method:** POST
* **Description:** This API allows Admin or Super Admin to assign an mId to an active merchant user.
* **Request details:** 
  + **Path Parameter:**

userName – Logged in user name.

* + **Headers**:

**Authorization:** Bearer <Merchant login token>

* **Request body**:

{

“assignedMIds”: [ List of MIDs ]

}

* **Response details**
* **Sample response –** Success 200 Ok

{

“status” 1,

“data”: [

{

“unAssignedMIds”: [List of MIDs],

“assignedMIDs”: [List of saved MIds]

}

]

}

* **Sample response**: Error 200 Ok

{

  "errors": [

    {

      "errorCode":"ERROR\_CODE ",

      "errorMessage":"ERROR\_MSG"

      },

  ],

"status":0

}

Sample request

|  |
| --- |
| [    "1000003"  ] |

Sample Success reponse

|  |
| --- |
| {  "data": [  "MIds is assigned Successfully"  ]  } |

#### Validation

##### Request Data Validation

* Header should have the valid JWT token.
* JWT token should have the Merchant as a role.

##### Business data validation

* Given user should be present.
* Status value should be correct.

##### Business Logic

* **Step 1: Merchant user controller**

Request reach merchant user controller and call merchant user service.

* **Step 2: Merchant user service**

Service fetches User object and validator is called.

* **Step 3: Merchant user Validator**

Validate if logged in user is parent of user for whom MIDs mapping is initiated.

* **Step 4: Merchant user service**

On successful validation, evaluate MIDs to be unassigned and new MIDs to be assigned. Call merchant user DAO.

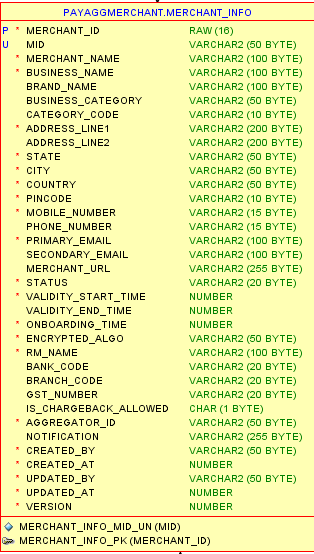
* **Step 5: Merchant user DAO**

Delete / add MIDs with merchant user.

* **Step 6: Merchant user service**

Send successful response otherwise send custom exception.

##### Database table – MERCHANT\_INFO



##### Information

* + Log Request

Log all incoming requests, including headers and body parameters (excluding sensitive information) to the database.

* + Log Response

Log all responses, including generated token details and any error messages except the token.

### Get - Merchant User and MIDs Mapping

#### API Definition and Implementations

* **Endpoint**: /merchant/v1/user/mid/mapping/{userName}
* **Method:** GET
* **Description:** This API helps to view a list of all merchant users associated with the active mIds.
* **Request details:** 
  + **Path Parameter:**

userName – Merchant user name.

* + **Headers**:

**Authorization:** Bearer <Merchant login token>

* Request body:

{

“assignedMIds”: [ List of MIDs ]

}

* **Response details**
* **Sample response –** Success 200 Ok

{

“status” 1,

“data”: [

{

“unAssignedMIds”: [List of MIDs],

“assignedMIDs”: [List of saved MIds]

}

]

}

Sample Success response example

|  |
| --- |
| {  "data": [  {  "assignedMIds": [  {  "merchantName": "GRIPS",  "status": "ACTIVE",  "mid": "1000003"  }  ],  "unAssignedMIds": []  }  ]  } |

* Sample response: Error 200 Ok

{

  "errors": [

    {

      "errorCode":"ERROR\_CODE ",

      "errorMessage":"ERROR\_MSG"

      },

  ],

"status":0

}

#### Validation

##### Request Data Validation

* Header should have the valid JWT token.
* Validates if the logged-in user is the parent of the requested merchant user.
* Given user should be present and status should be active.

##### Business logic

* **Step 1: Merchant user controller**

Request reach merchant user controller and call merchant user service.

* **Step 2: Merchant user service**

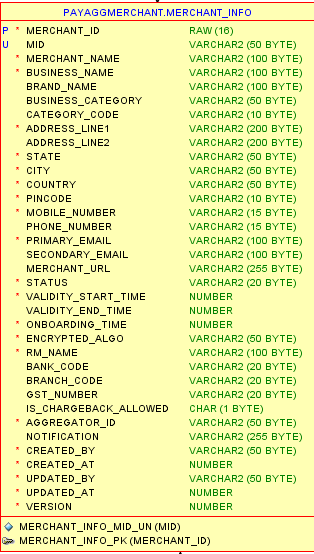
Service fetches User object and validator is called.

* **Step 3: Merchant user Validator**

Validate if logged in user is parent of user for whom MIDs mapping is initiated.

* **Step 4: Merchant user service**
* On Successful validation, fetch MID listing for user.
* Send Successful response with MIDs list.

##### Database table – MERCHANT\_INFO



##### Information

* + Log Request

Log all incoming requests, including headers and body parameters (excluding sensitive information) to the database.

* + Log Response

Log all responses, including generated token details and any error messages except the token.

### Merchant Portal – Get Merchant User Roles

#### API definition and implementation

* **Endpoint:** /merchant/v1/user/roles
* **Method:** GET
* **Description:** This API helps to view a list of all the user roles available for the active merchants.
* **Request details:** 
  + Headers: Requires valid Merchant User Login Token.

**Authorization:** Bearer <Merchant login token>

* Request body: Empty
* **Response details:**
* **Response:** Success: 200 – Ok

{

[

{

“id”: UUID,

“role”: string,

“description”: string

},

],

“Status"; 1

}

* **Response**: Error 200 – Ok

{

"status":0,

"error": [

{

"errorCode": String,

"errorMessage": String

}

]

}

Sample Success response

|  |
| --- |
| {  "status": 1,  "data": [  {  "id": "2e0075e9-a83c-9db7-e063-7c86b10adca3",  "role": "SUPER\_ADMIN",  "description": "Super Admin will be created by SBIePay"  },  {  "id": "2e0075e9-a83d-9db7-e063-7c86b10adca3",  "role": "ADMIN",  "description": "Admins will be created either during onboarding of MID, or later by superadmins."  },  {  "id": "2e0075e9-a83e-9db7-e063-7c86b10adca3",  "role": "USER",  "description": "Users will be created by Admins/Superadmins."  }  ],  "total": 3  } |

##### Validation

* Check MID is present and valid, if not return error message.
* The API can only be accessed by users authenticated via the Merchant Access Token.

##### Business logic

* + **Step 1: MerchantUser Controller**

Request reach controller and call merchant user service.

* + **Step 2: MerchantUser service**

Service call merchant user DAO.

* + **Step 3: MerchantUser DAO**

Merchant use DAO will Merchant user repository.

* + **Step 4: MerchantUser repository**

Repository will fetch roles list.

* + **Step 5: MerchantUser**

Send Successful response of roles list.

##### Information

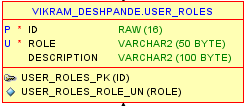
* Log Request

Log all incoming requests, including headers and body parameters (excluding sensitive information) to the database.

* Log Response

Log all responses, including generated token details and any error messages except the token.

##### Database table – USER\_ROLES



### Merchant Portal – Get Users List for Merchant Identifier.

#### API definition and implementation

* **Endpoint:** /merchant/v1/user/download
* **Method:** POST
* **Description:** This API helps to view a list of users associated with an active merchant in the CSV file format.
* **Request details:** 
  + Headers: Requires valid Merchant User Login Token.

**Authorization:** Bearer <Merchant login token>

* Path params:
* Request body:

{

    "userName": String,

    "name" String,

    "mobilePhone": String,

    "fromDate": long,

    "toDate": long,

    "status": String

}

* **Response details:**
* **Response :** CSV file
* **Response**: Error 200 – Ok

{

"status":0,

"error": [

{

"errorCode": String,

"errorMessage": String

}

]

}

Sample request

|  |
| --- |
| {  "userName": "ravi",  "name": "Ravi",  "mobilePhone": "8791602346",  "fromDate": 1737366148271,  "toDate": 1747366148271,  "status": "ACTIVE"  } |

Sample response

|  |
| --- |
| User Id,Name,Email,Mobile No.,User Type,Status,Creation Date ravi,Ravi,r\*\*\*\*\*\*\*\*\*\*re@\*\*\*\*\*\*\*\*\*\*in,\*\*\*\*\*\*\*346,ADMIN,ACTIVE,20 Feb 2025 09:31:10 AM |

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Description** | **Mandatory** | **Data Type** |
| name | UUID request ID | No | string |
| userName | username | No | string |
| fromDate | First name | No | Long |
| toDate | email address | No | Long |
| mobilePhone | Mobile number | No | string |
| status | status | No | string |

#### Validation

##### Request Data Validation

* Header should have the valid JWT token.
* JWT token should have the Merchant as a role.
* Check MID is present and valid, if not return error message.

##### Business data validation

* The API can only be accessed by users authenticated via the Merchant Access Token.

##### Business logic

* + **Step 1: Merchant User Controller**

Request reach controller and call merchant user service.

* + **Step 2: Merchant User service**

Call validator service to validate merchant identifier, on Successful validation c call merchant user DAO.

* + **Step 3: Merchant User DAO**

DAO will call merchant user repository.

* + **Step 4: Merchant User repository**

Repository will fetch list of users for merchant identifier.

* + **Step 5: File generator service**

Process data to generate file generation. Send file stream on Success.

* + **Step 6: Merchant User service**

Send error or custom exception if any.

##### Information

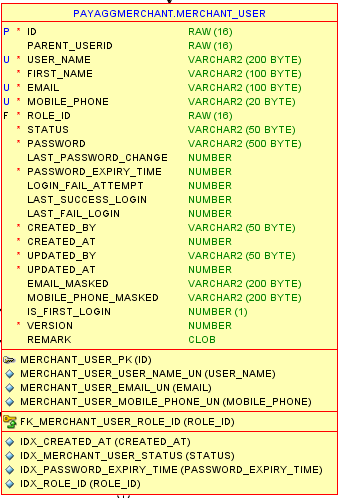
* Log Request

Log all incoming requests, including headers and body parameters (excluding sensitive information) securely to the database.

* Log Response

Log all responses, including generated token details and any error messages except the token.

##### Database table – MERCHANT\_USER



### Merchant Portal – Merchant Access Info

#### API implementation

* **Endpoint:** /merchant/v1/user/access/mids/info
* **Method:** GET
* **Description:** This API displays the completeaccess details of the logged-in merchant user..
* **Request details:** 
  + Headers: Requires valid Merchant User Login Token.

**Authorization:** Bearer <Merchant login token>

* Path params: Pagination details JSON string

e.g "{"page": 0,"size": 1,"sort": "mId,asc"}"

* Request body: None
* **Response:**
* Success response

{

"status": 1,

"data": [

“mId”: string,

“merchantName”: string,

“status”: string

],

"count": 1,

"total": 1,

}

* Sample Success response

|  |
| --- |
| {  "status": 1,  "data": [  {  "merchantName": "GRIPS",  "status": "ACTIVE",  "mid": "1000003"  }  ],  "count": 1,  "total": 1  } |

* Error response

{

"errors": [

{

"errorCode":"ERROR\_CODE ",

"errorMessage":"ERROR\_MSG"

},

],

"status":0

#### Validation

* None

##### Business logic

* **Merchant user controller**

Request reaches controller, request delegated to service for further processing.

* **Merchant user service**
* Retrieve user name from token.
* Fetch merchants for logged in user.
* **Merchant user controller**

Send Success response with list of merchants.

##### Information

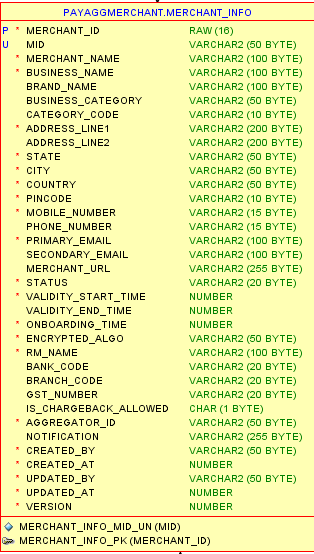
* + - Log Request

Log all incoming requests, including headers and body parameters (excluding sensitive information) securely to the database.

* + - Log Response

Log all responses, including generated token details and any error messages except the token.

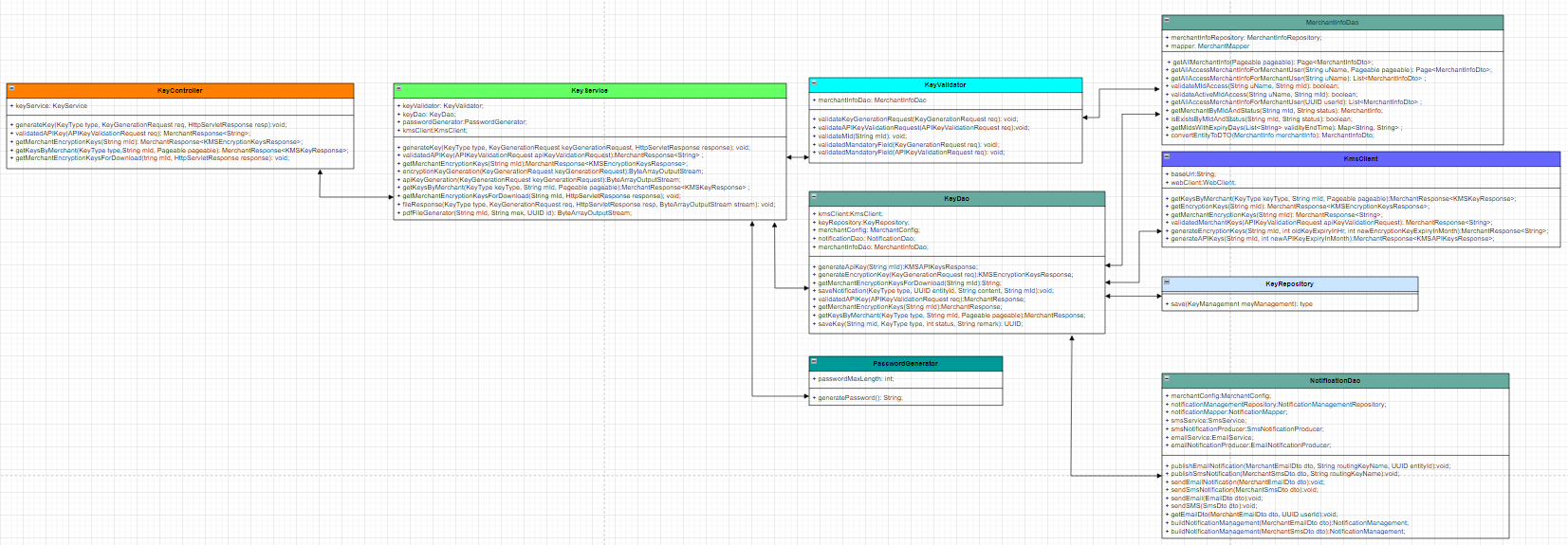
##### Database table MERCHANT\_INFO



## Account and Settings

### API Key and Encryption Key Management

#### Class diagram



#### Generate API and encryption key for Merchant

##### API definition and implementations

* **Endpoint**: /merchant/v1/key/generation/{type}
* **Method:** POST
* **Description:**
* This API helps in the generation, management, and secure distribution of the API and encryption keys. It alsointegrates the API and encryption keys with KMS (Key Management Service) and provides the functionality for generating passwordprotected pdfs to securely share the keys with the merchants via Notification Service with Kafka.
* **Request details:** 
  + - * Path Parameter: type (values: API or Encryption)
      * Headers: Requires valid Merchant User Login Token.

**Authorization:** Bearer <Merchant login token>

* **Request object:**

{   
 “mId”: 1000056,

" oldKeyExpiryInHr": 24 // number of hours for expiry   
}

* **Response: Success: 200 – Ok**

{

"data": [

{

"keyId": "12345-67890-ABCDE",

"createdDate": 1672444800000,

"status": "GENERATED",

}

],

"status":1,

"count": 1,

"total": 1

}

Sample request

|  |
| --- |
| {  "mId": "1000003",  "oldKeyExpiryInHr": 12  } |

Sample response

|  |
| --- |
| {  "status": 0,  "errors": [  {  "errorCode": "1006",  "errorMessage": "API-Key generation failed."  }  ]  } |

* **Response**: Error 200 – Ok

{

"status":0,

"error": [

{

"errorCode": String,

"errorMessage": String

}

]

}

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Description** | **Mandatory** | **Data Type** |
| mId | Merchant ID | Yes | string |
| oldKeyExpiryInHr | oldKeyExpiryInHr | No | int |

##### Validation

* Key type should be one of { API or ENCRYPTION }
* Merchant identifier should be valid with active status.

##### Business Logic

* + **Step 1: Key controller**

Request reach key controller and call Key service.

* + **Step 2: Key service**

Service calls key validator on successful validation key generation will be called.

* + **Step 3: Key validator**

Validates key generation request for mandatory fields and DAO will be called for merchant MID.

* + **Step 4: Key DAO**

KMS service called to generate key.

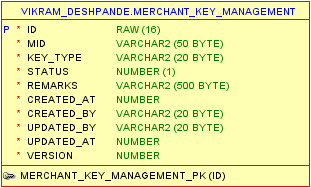
* + **Step 5: Notification service**

Email sent to User.

* + **Step 6: Key service**

Send Successful response in PDF format, file will have API key and encryption key.

##### Database table - MERCHANT\_KEY\_MANAGEMENT



##### Information

* + Log Request

Log all incoming requests, including headers and body parameters (excluding sensitive information).

* + Log Response

Log all responses, including generated token details and any error messages except the token.

#### Validate API key for Merchant

##### API definition and implementations

* **Endpoint**: /merchant/v1/key/validated/api
* **Method:** POST
* **Description:** This API helps in validating the API and encryption keys. This service connects with a third-party KMS (Key Management Service) and functionality for validating merchant keys.
* **Request details:** 
  + - * Headers: Requires valid Merchant User Login Token.

**Authorization:** Bearer <Merchant login token>

* + - **Request object:**

{   
 “apiKey”: “string”,

" apiKeySecret": "string”   
}

* + - **Response:** Success: 200 – Ok

{

"data": [ ],

"status":1,

"count": 1,

"total": 1

}

* + - **Response**: Error 200 – Ok

{

"status":0,

"error": [

{

"errorCode": String,

"errorMessage": String

}

]

}

Sample request

|  |
| --- |
| {  "apiKey": "/ZMrnWEDVNX4FTRsSmd2ng==",  "apiKeySecret": "0ot5ovBkyNw2rnyeADtjUNrCfY6HqEdCWhxa0Nonr6k="  } |

Sample response

|  |
| --- |
| {  "status": 1,  "data": [  "1000003"  ]  } |

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Description** | **Mandatory** | **Data Type** |
| apiKey | API key | Yes | string |
| apiKeySecret | api secret Key | Yes | string |

##### Validation

* Check request for mandatory fileds api key and api key secret.

##### Business Logic

* + **Step 1: Key controller**

Request reach key controller and call Key service.

* + **Step 2: Key service**

Service calls key validator.

* + **Step 3: Key validator**

Validator checks for mandatory fields, on validation Key DAO will be called.

* + **Step 4: Key DAO**

DAO callsKMS service to validate merchant keys.

* + **Step 5: Key service**

Send Successful response.

##### Database table - NA

##### Information

* + Log Request: Log all incoming requests, including headers and body parameters (excluding sensitive information).
  + Log Response: Log all responses, including generated token details and any error messages except the token.

#### Get Merchant Encryption Keys

##### API definition and implementations

* **Endpoint**: /merchant/v1/key/encryption/{mId}
* **Method:** GET
* **Description:** Service for API encryption keys for the given merchant ID.
* **Request details:**
* Path parameters

mId – Merchant identifier

* + - * Headers: Requires valid Merchant User Login Token.

**Authorization:** Bearer <Merchant login token>

* + - **Request body:** Empty
    - **Response:** Success: 200 – Ok

{

"data": [

{

“kek”: string,

“mek”: string

}

],

"status":1,

"count": 1,

"total": 1

}

* + - **Response**: Error 200 – Ok

{

"status":0,

"error": [

{

"errorCode": String,

"errorMessage": String

}

]

}

Sample response

|  |
| --- |
| {  "status": 1,  "data": [  {  "mek": "AAAAAAAAAAAAAAAAAAAAAIh86i5YCKcYJrQdtuYSR3eJiuMr3USZsD5gFAm/YXL1k8l0vXtpFuBVI80CDLCefooGdxD/u3kbVqj7aA==",  "kek": "AAAAAAAAAAAAAAAAAAAAAHAUW3KooF0/rDJWakfnq8WdGqX98V44IIeZRRBMscasV/KeIHvShCUIaaw3e5IUcOqFIx1kxyeuYocYyg=="  }  ]  } |

##### Validation

* + Check request for mandatory field merchant identifier with active status.

##### Business Logic

* + **Step 1: Key controller**

Request reach key controller and call Key service.

* + **Step 2: Key service**

Service calls key validator.

* + **Step 3: Key validator**

Validator checks for mandatory fields, on validation Key DAO will be called.

* + **Step 4: Key DAO**

DAO callsKMS client service to fetch merchant keys.

* + **Step 5: Key service**

Send Successful response, will have API key and encryption key.

##### Database table - NA

##### Information

* + Log Request

Log all incoming requests, including headers and body parameters (excluding sensitive information).

* + Log Response

Log all responses, including generated token details and any error messages except the token.

#### List of keys for a Merchant

##### API definition and implementations

* **Endpoint**: /merchant/v1/key/{mId}/{keyType}
* **Method:** GET
* **Description:** This API retrieves a list of keys for the given merchant ID and key type.
* **Request details:**
* Path parameters

mId – Merchant identifier

keyType – API or ENCRYPTION

* + - * Headers: Requires valid Merchant User Login Token.

**Authorization:** Bearer <Merchant login token>

* **Request body:** Empty
* **Response:** Success: 200 – Ok

{

"data": [

{

"merchantId": String,

"expiryTime": long,

"status": String,

"createdAt": long

}

],

"status":1,

"count": 1,

"total": 1

}

* **Response**: Error 200 – Ok

{

"status":0,

"error": [

{

"errorCode": String,

"errorMessage": String

}

]

}

Sample response

|  |
| --- |
| {  "status": 1,  "data": [  {  "merchantId": "1000003",  "expiryTime": 1791039547228,  "status": "ACTIVE",  "createdAt": 1738594747231  },  {  "merchantId": "1000003",  "expiryTime": 1770732050835,  "status": "INACTIVE",  "createdAt": 1739196050838  },  {  "merchantId": "1000003",  "expiryTime": 1770731894647,  "status": "INACTIVE",  "createdAt": 1739195894651  },  {  "merchantId": "1000003",  "expiryTime": 1770731893898,  "status": "INACTIVE",  "createdAt": 1739195893919  },  {  "merchantId": "1000003",  "expiryTime": 1791033993779,  "status": "INACTIVE",  "createdAt": 1738589193781  },  {  "merchantId": "1000003",  "expiryTime": 1791033470484,  "status": "INACTIVE",  "createdAt": 1738588670487  },  {  "merchantId": "1000003",  "expiryTime": 1770124262382,  "status": "INACTIVE",  "createdAt": 1738588262384  },  {  "merchantId": "1000003",  "expiryTime": 1770123553749,  "status": "INACTIVE",  "createdAt": 1738587553753  },  {  "merchantId": "1000003",  "expiryTime": 1791031974121,  "status": "INACTIVE",  "createdAt": 1738587174124  },  {  "merchantId": "1000003",  "expiryTime": 1770122147572,  "status": "INACTIVE",  "createdAt": 1738586147575  },  {  "merchantId": "1000003",  "expiryTime": 1791030856075,  "status": "INACTIVE",  "createdAt": 1738586056078  },  {  "merchantId": "1000003",  "expiryTime": 1791030854303,  "status": "INACTIVE",  "createdAt": 1738586054306  },  {  "merchantId": "1000003",  "expiryTime": 1791030772868,  "status": "INACTIVE",  "createdAt": 1738585972871  },  {  "merchantId": "1000003",  "expiryTime": 1791030772145,  "status": "INACTIVE",  "createdAt": 1738585972148  },  {  "merchantId": "1000003",  "expiryTime": 1791030771466,  "status": "INACTIVE",  "createdAt": 1738585971474  },  {  "merchantId": "1000003",  "expiryTime": 1791030770735,  "status": "INACTIVE",  "createdAt": 1738585970738  },  {  "merchantId": "1000003",  "expiryTime": 1791030770003,  "status": "INACTIVE",  "createdAt": 1738585970006  },  {  "merchantId": "1000003",  "expiryTime": 1791030769324,  "status": "INACTIVE",  "createdAt": 1738585969327  },  {  "merchantId": "1000003",  "expiryTime": 1791030768537,  "status": "INACTIVE",  "createdAt": 1738585968540  },  {  "merchantId": "1000003",  "expiryTime": 1791030767618,  "status": "INACTIVE",  "createdAt": 1738585967620  },  {  "merchantId": "1000003",  "expiryTime": 1791030766051,  "status": "INACTIVE",  "createdAt": 1738585966053  },  {  "merchantId": "1000003",  "expiryTime": 1791030560209,  "status": "INACTIVE",  "createdAt": 1738585760211  },  {  "merchantId": "1000003",  "expiryTime": 1791030559225,  "status": "INACTIVE",  "createdAt": 1738585759228  },  {  "merchantId": "1000003",  "expiryTime": 1791030558093,  "status": "INACTIVE",  "createdAt": 1738585758096  },  {  "merchantId": "1000003",  "expiryTime": 1791030556194,  "status": "INACTIVE",  "createdAt": 1738585756197  },  {  "merchantId": "1000003",  "expiryTime": 1791030427762,  "status": "INACTIVE",  "createdAt": 1738585627765  },  {  "merchantId": "1000003",  "expiryTime": 1791030176385,  "status": "INACTIVE",  "createdAt": 1738585376388  },  {  "merchantId": "1000003",  "expiryTime": 1770118236794,  "status": "INACTIVE",  "createdAt": 1738582236798  },  {  "merchantId": "1000003",  "expiryTime": 1770117231376,  "status": "INACTIVE",  "createdAt": 1738581231391  },  {  "merchantId": "1000003",  "expiryTime": 1790857397775,  "status": "INACTIVE",  "createdAt": 1738412597816  },  {  "merchantId": "1000003",  "expiryTime": 1769947012849,  "status": "INACTIVE",  "createdAt": 1738411012887  },  {  "merchantId": "1000003",  "expiryTime": 1769946283435,  "status": "INACTIVE",  "createdAt": 1738410283438  },  {  "merchantId": "1000003",  "expiryTime": 1769945582149,  "status": "INACTIVE",  "createdAt": 1738409582165  },  {  "merchantId": "1000003",  "expiryTime": 1769859987036,  "status": "INACTIVE",  "createdAt": 1738323987039  },  {  "merchantId": "1000003",  "expiryTime": 1769859986494,  "status": "INACTIVE",  "createdAt": 1738323986498  },  {  "merchantId": "1000003",  "expiryTime": 1769858976188,  "status": "INACTIVE",  "createdAt": 1738322976192  },  {  "merchantId": "1000003",  "expiryTime": 1769858411582,  "status": "INACTIVE",  "createdAt": 1738322411586  },  {  "merchantId": "1000003",  "expiryTime": 1790766848056,  "status": "INACTIVE",  "createdAt": 1738322048060  },  {  "merchantId": "1000003",  "expiryTime": 1769857526887,  "status": "INACTIVE",  "createdAt": 1738321526925  },  {  "merchantId": "1000003",  "expiryTime": 1740738782323,  "status": "INACTIVE",  "createdAt": 1738319582329  },  {  "merchantId": "1000003",  "expiryTime": 1769852751550,  "status": "INACTIVE",  "createdAt": 1738316751554  },  {  "merchantId": "1000003",  "expiryTime": 1790758190990,  "status": "INACTIVE",  "createdAt": 1738313390994  },  {  "merchantId": "1000003",  "expiryTime": 1769845917190,  "status": "INACTIVE",  "createdAt": 1738309917196  },  {  "merchantId": "1000003",  "expiryTime": 1790751856412,  "status": "INACTIVE",  "createdAt": 1738307056426  },  {  "merchantId": "1000003",  "expiryTime": 1740726208571,  "status": "INACTIVE",  "createdAt": 1738307008597  }  ],  "count": 45,  "total": 45  } |

##### Validation - None

##### Business Logic

* + **Step 1: Key controller**

Request reach key controller and call Key service.

* + **Step 2: Key service**

Service calls key validator.

* + **Step 3: Key validator**

Validator checks for mandatory fields, on validation Key DAO will be called.

* + **Step 4: Key DAO**

DAO callsKMS client service to fetch list of merchant keys.

* + **Step 5: Key service**

Send Successful response with list of keys key.

##### Database table - NA

##### Information

* + Log Request

Log all incoming requests, including headers and body parameters (excluding sensitive information).

* + Log Response

Log all responses, including generated token details and any error messages except the token.

#### Download keys for a merchant

##### API definition and implementations

* **Endpoint**: /merchant/v1/key/download/encryption/{mId}
* **Method:** GET
* **Description:** API to generate and download a pwd-protected Merchant Encryption key in PDF for a given Merchant ID
* **Request details:**
* Path parameters

mId – Merchant identifier

* + - * Headers: Requires valid Merchant User Login Token.

**Authorization:** Bearer <Merchant login token>

* **Request body:** Empty
* **Response:** Success: 200 – PDF file.
* ENCRYPTION\_1000003\_1742905824119.pdf
* **Response**: Error 200 – Ok

{

"status":0,

"error": [

{

"errorCode": String,

"errorMessage": String

}

]

}

##### Validation

* Validate mId, should be valid and active.

##### Business Logic

* **Step 1:** **Key controller**

Request reach key controller and call Key service.

* **Step 2:** **Key service**

Service calls key validator.

* **Step 3:** **Key validator**

Validator checks for mandatory fields, on validation Key DAO will be called.

* **Step 4:** **Key DAO**

DAO calls KMS client service to fetch merchant keys.

* **Step 5:** **Key service**

Send Successful response in PDF format, file will have API key and encryption key.

##### Database table - NA

##### Information

* + Log Request

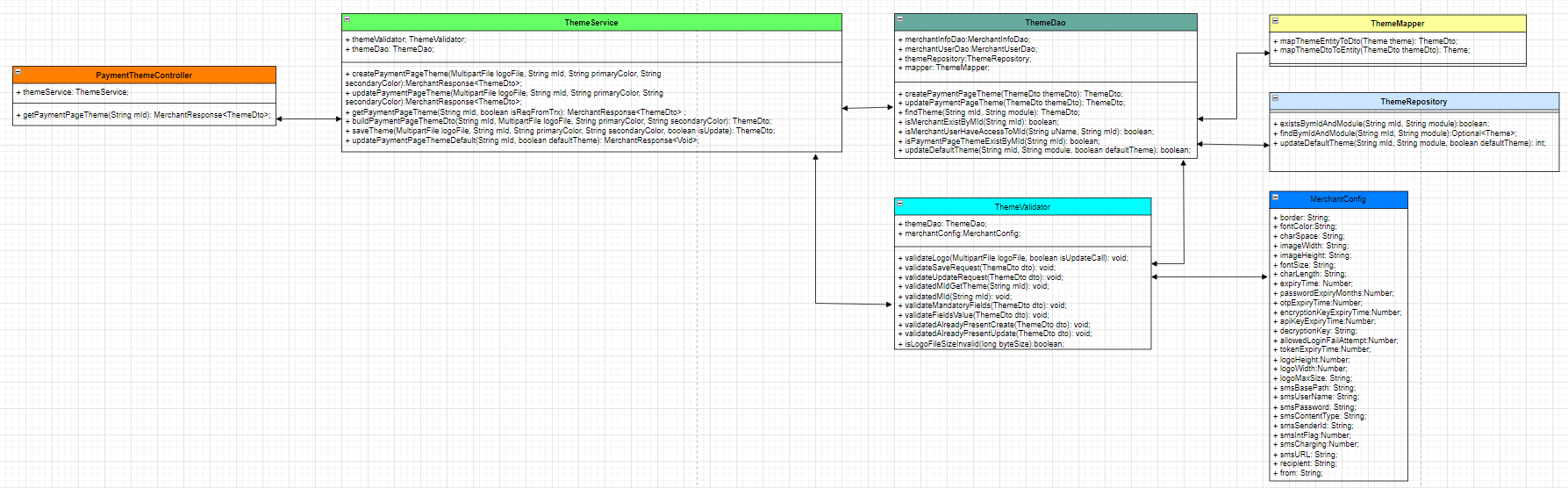
Log all incoming requests, including headers and body parameters (excluding sensitive information).

* + Log Response

Log all responses, including generated token details and any error messages except the token.

### Payment Page Theme

#### Class diagram



#### Create configured page theme

##### API Definition and implementation

* **Endpoint**: /merchant/v1/theme/payment/{mid}
* **Method**: POST
* **Description**: Merchant users can create a payment page theme by logo, primary color and secondary color.
* **Request Details:**

Content-Type: multipart/form-data

Authorization (String): Bearer <Access Token>

* **Request body:**

{

"logo": "file":”Path”,

   "primaryColor": "text":”Hex Color Code”,

   "secondayColor": "text":”Hex Color Code”

}

* **Response:**
* **Example** **response –** Success 200 ok

{

    "status": 1,

    "data": [

        {

            "logo": "Base64 Image",

            "primaryColor": "Hex Color Code",

            "secondaryColor": "Hex Color Code",

        }

    ],

    "count": 1

}

* **Example Response –** Error 200 ok

{

  "errors": [

    {

      "errorCode":"ERROR\_CODE ",

      "errorMessage":"ERROR\_MSG"

    },

  ],

"status":0

}

Sample response

|  |
| --- |
| {  "status": 0,  "errors": [  {  "errorCode": "1004",  "errorMessage": "The theme with MId is already present."  }  ]  } |

##### Validation

* Check logo file present or not and type should be one of

[x-icon, png, jpg,jpeg, png, svg+xml]

* Check logo file has valid size and dimension.
* Check primary and secondary colour hex code
* Check if theme exists for merchant.
* The API can only be accessed by users authenticated via the access token.
* Only one theme can be created.

##### Business logic

* **Step 1: Theme controller**

Request reach them controller and call theme service.

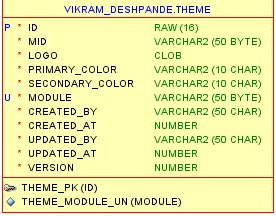
* **Step 2: Theme service**

Service validates request for mandatory fields.

* Check mandatory fields logo and merchant id.
* On Successful validations, save theme in database.
* **Step 3: Theme service**

Send Success response otherwise send custom exception

##### Database table: THEME



##### Information

* If valid, extract user details and set them in the security context.
* If invalid, reject the request with an unauthorized response.
* Log Request

Log all incoming requests, including headers and body parameters (excluding sensitive information) to the database.

* Log Response

Log all responses, including generated token details and any error messages except the token.

#### Update Configured Page Theme

##### API Definition and implementation

* **Endpoint**: /merchant/v1/theme/payment/{mid}
* **Method**: PUT
* **Description**: Merchant users can update a payment page theme by logo, primary color and secondary color.
* **Request Details:**
* Path parameters

mId – Merchant identifier

logo - Logo file for the Payment Page Theme

Content-Type: multipart/form-data

Authorization (String): Bearer <JWT Token>

* **Request body:**

{

"logo": "file":”Path”,

   "primaryColor": "text":”Hex Color Code”,

   "secondayColor": "text":”Hex Color Code”

}

* **Response:**
* **Example** **response –** Success 200 ok

{

    "status": 1,

    "data": [

        {

            "logo": "Base64 Image",

            "primaryColor": "Hex Color Code",

            "secondaryColor": "Hex Color Code",

        }

    ],

    "count": 1

}

* **Example Response –** Error 200 ok

{

  "errors": [

    {

      "errorCode":"ERROR\_CODE ",

      "errorMessage":"ERROR\_MSG"

    },

  ],

"status":0

}

Sample Success response

|  |
| --- |
| {  "status": 1,  "data": [  {  "primaryColor": "#3cb371",  "secondaryColor": "#ffa500",  "logo": "",  "module": "PAYMENT",  "defaultTheme": **false**,  "mId": "1000003"  }  ],  "total": 1  } |

##### Validation

* Check for existing theme, if not found raise custom exception.
* Check logo file present or not and type should be one of

[x-icon, png, jpg,jpeg, png, svg+xml]

* Check logo file has valid size and dimension.
* Check primary and secondary colour hex code
* Check if theme exists for merchant.
* The API can only be accessed by users authenticated via the access token.

##### Business logic

* **Step 1: Theme controller**

Request reach them controller and call theme service.

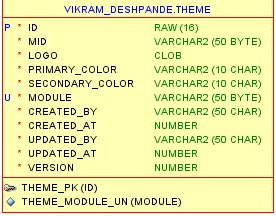
* **Step 2: Theme service**

Service validates request for mandatory fields.

* Check mandatory fields merchant id.
* On Successful validations, update payment as default in database.
* **Step 3: Theme service**

Send Success response otherwise send custom exception.

##### Database table: THEME



##### Information

* If valid, extract user details and set them in the security context.
* If invalid, reject the request with an unauthorized response.
* Log Request

Log all incoming requests, including headers and body parameters (excluding sensitive information) to the database.

* Log Response:

Log all responses, including generated token details and any error messages except the token.

#### Get Payment theme

##### API Definition and implementation

* **Endpoint:** /merchant/v1/theme/payment/{mid}
* **Method:** GET
* **Description:** This API allows the merchant users to view the payment page theme based on the selection criteria including logo, primary color, and secondary color.
* **Request** **Details**
* Content-Type: multipart/form-data
* Authorization (String): Bearer <JWT Token>
* Path parameters

mId – Merchant identifier

isReqFromTrx – Is required for transaction

* **Response**:
* **Example response** – Success 200 ok

{

    "status": 1,

    "data": [

        {

“mId”: string,

"primaryColor": "Hex Color Code",

            "secondaryColor": "Hex Color Code",

            "logo": "Base64 Image",

           “module”: string,

“defaultTheme”: boolean

        }

    ],

    "count": 1,

    "total": 1

}

* **Example Response** – Error 200 ok

{

  "errors": [

    {

      "errorCode":"ERROR\_CODE ",

      "errorMessage":"ERROR\_MSG"

    },

  ],

"status":0

}

Sample response

|  |
| --- |
| {  "status": 1,  "data": [  {  "primaryColor": "#00ff40",  "secondaryColor": "#a04c4c",  "logo": "",  "module": "PAYMENT",  "defaultTheme": **false**,  "mId": "1000003"  }  ],  "total": 1  } |

##### Validation

* Check if MID is valid, or else raise custom exception if invalid.
* Check merchant user has access on that MID.
* Check transaction flag(Request data) and fetch payment page.If both true return payment page theme else configured one.
* The API can only be accessed by users authenticated via the Merchant Access Token.

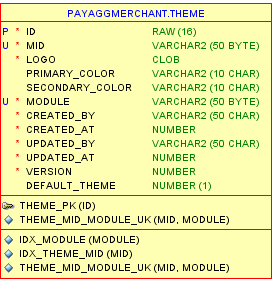
##### Business logic

* **Step 1: Theme controller**

Request reach theme controller and call theme service.

* **Step 2: Theme service**
* Fetch theme by mid and module transaction flag true.
* Send configured theme if transaction flag is false.
* Success response otherwise send custom exception.

##### Database Table: THEME



##### Information

* If valid, extract user details and set them in the security context.
* If invalid, reject the request with an unauthorized response.
* Log Request
* Log all incoming requests, including headers and body parameters (excluding sensitive information) to the database.
* Log Response
* Log all responses, including generated token details and any error messages except the token.

#### Update Payment Page Theme

##### API Definition and implementation

* **Endpoint**: /merchant/v1/theme/payment/{mid}/default
* **Method**: PUT
* **Description**: This API allows the merchant users to update the payment page theme including logo, primary color, and secondary color .
* **Request Details:**
* **Path parameters**

mId – Merchant identifier

Content-Type: multipart/form-data

Authorization (String): Bearer <JWT Token>

* **Request body:** None
* **Response:**
* **Example** **response –** Success 200 ok

{

    "status": 1,

“total”: 1,

    "data": null,

    "count": 1

}

* + - * **Example Response –** Error 200 ok

{

  "errors": [

    {

      "errorCode":"ERROR\_CODE ",

      "errorMessage":"ERROR\_MSG"

    },

  ],

"status":0

}

Sample response

|  |
| --- |
| {  "status": 1,  "total": 1  } |

##### Validation

* Check if merchant is valid with an active status.

##### Business logic

* **Step 1: Theme controller**

Request reaches controller and calls the theme service.

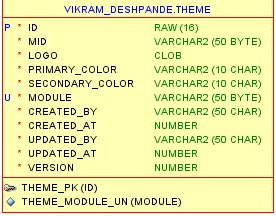
* **Step 2: Theme service**

Service validates request for mandatory fields.

* Check mandatory merchant id.
* On Successful validations, set DefaultTheme true for given MID
* **Step 3: Theme service**

Send Success response otherwise send custom exception

##### Database Table: THEME



##### Information

* If valid, extract user details and set them in the security context.
* If invalid, reject the request with an unauthorized response.
* Log Request:

Log all incoming requests, including headers and body

parameters (excluding sensitive information) to the database.

* Log Response:

Log all responses, including generated token details and any error messages except the token.

#### Reset Payment Page Theme

##### API Definition and implementation

* **Endpoint**: /merchant/v1/theme/payment/{mid}/reset
* **Method**: PUT
* **Description**: This API allows the merchant users to reset thepayment page theme including logo, primary color, and secondary color.
* **Request Details:**
* **Path parameters**

mId – Merchant identifier

Content-Type: multipart/form-data

Authorization (String): Bearer <JWT Token>

* **Request body:** None
* **Response:**
* **Example** **response –** Success 200 ok

{

    "status": 1,

“total”: 1,

    "data": null,

    "count": 1

}

* **Example Response –** Error 200 ok

{

  "errors": [

    {

      "errorCode":"ERROR\_CODE ",

      "errorMessage":"ERROR\_MSG"

    },

  ],

"status":0

}

Sample response

|  |
| --- |
| {  "status": 1,  "total": 1  } |

##### Validation

* Check if merchant is valid with an active status.

##### Business logic

* **Step 1: Theme controller**

Request reach them controller and call theme service.

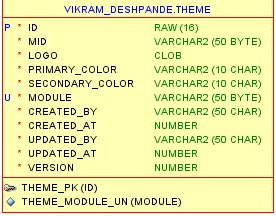
* **Step 2: Theme service**

Service validates request for mandatory fields.

* Check mandatory field merchant id.
* On Successful validations, set default theme true to falsefor given MID.
* **Step 3: Theme service**

Send Success response otherwise send custom exception

##### Database Table: THEME



##### Information

* If valid, extract user details and set them in the security context.
* If invalid, reject the request with an unauthorized response.
* Log Request

:

Log all incoming requests, including headers and body

parameters (excluding sensitive information) to the database.

* Log Response
* Log all responses, including generated token details and any error messages except the token.

#### Get Payment Page Theme

##### API Definition and implementation

* **Endpoint:** /merchant/v1/paymenttheme /{mid}
* **Method:** GET
* **Description:** This API allows merchant users to view the payment page based on the selection criteria including logo, primary color, and secondary color.
* **Request Details**
* Content-Type: multipart/form-data
* Authorization (String): Bearer <JWT Token>
* Path parameters

mId – Merchant identifier

* **Response**:
* **Example response** – Success 200 ok

{

    "status": 1,

    "data": [

        {

“mId”: string,

"primaryColor": "Hex Color Code",

            "secondaryColor": "Hex Color Code",

            "logo": "Base64 Image",

           “module”: string,

“defaultTheme”: boolean

        }

    ],

    "count": 1,

    "total": 1

}

* **Example Response** – Error 200 ok

{

  "errors": [

    {

      "errorCode":"ERROR\_CODE ",

      "errorMessage":"ERROR\_MSG"

    },

  ],

"status":0

}

Sample response

|  |
| --- |
| {  "status": 1,  "data": [  {  "primaryColor": "#3cb371",  "secondaryColor": "#ffa500",  "logo": "",  "module": "PAYMENT",  "defaultTheme": **false**,  "mId": "1000003"  }  ],  "total": 1  } |

##### Validation

* Check if MID is valid, or else raise custom exception if invalid.
* Check merchant user has access on that MID.
* Check transaction flag(Request data) and fetch payment page.If both true return payment page theme else configured one.
* The API can only be accessed by users authenticated via the Merchant Access Token.

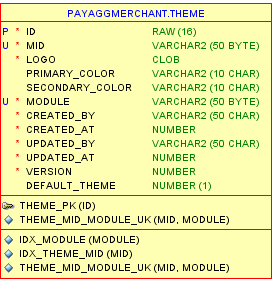
##### Business logic

* **Step 1: Theme controller**

Request reach theme controller and call theme service.

* **Step 2: Theme service**
* Fetch theme by mid and module transaction flag true.
* Send configured theme if transaction flag is false.
* Success response otherwise send custom exception.

##### Database Table: THEME



##### Information

* If valid, extract user details and set them in the security context.
* If invalid, reject the request with an unauthorized response.
* Log Request

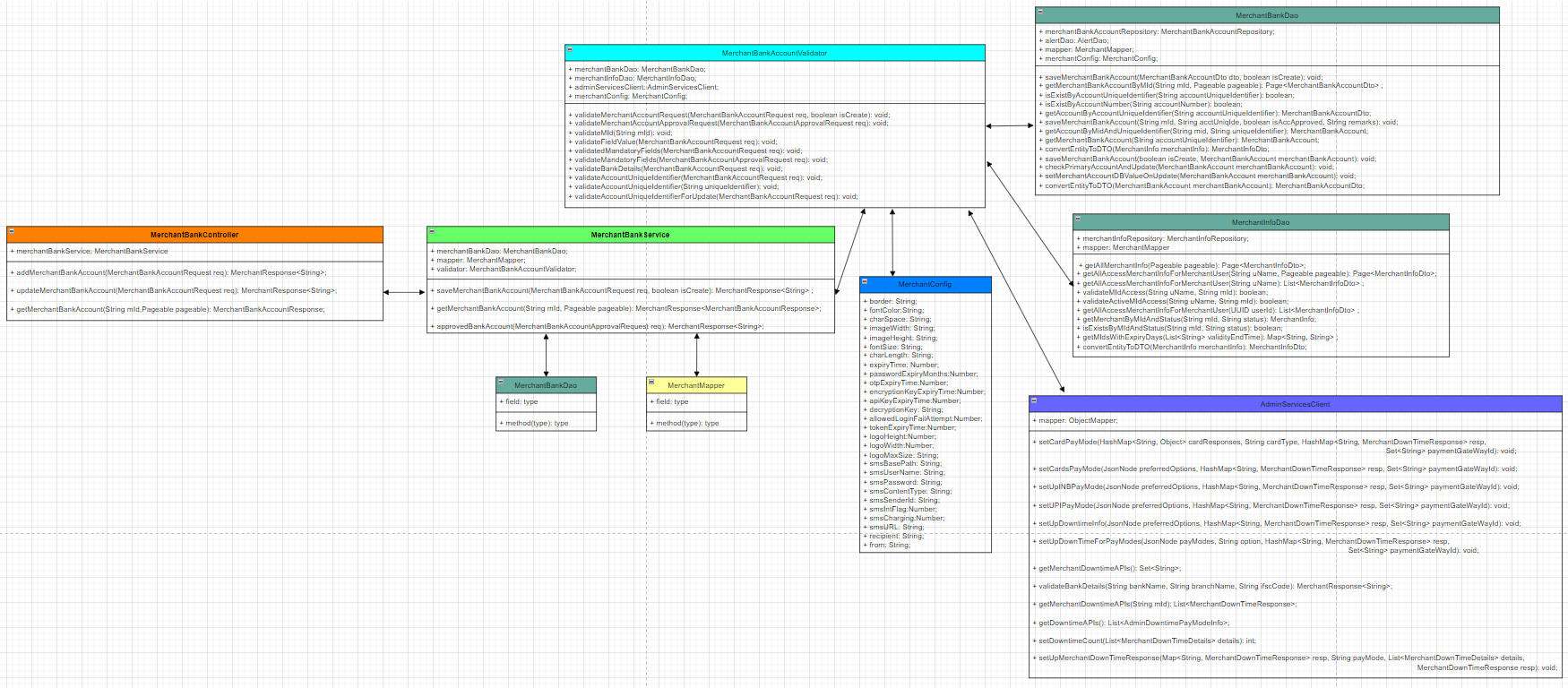
Log all incoming requests, including headers and body parameters (excluding sensitive information) to the database.

* Log Response

Log all responses, including generated token details and any error messages except the token.

### Merchant Portal – Bank Accounts

#### Class diagram



#### Bank Accounts Listing

##### API Definition and implementation

* **Endpoint:** /merchant/v1/bank/ {mId}
* **Method:** GET
* **Description:** This API helps an Admin or Super Admin to retrieve a list of bank accounts details associated with a specific merchant.
* **Request details:**
* Request header: Token contains valid username.
* Parameter: mId- merchant Identifier, pageable

{

"Authorization": Bearer <Token with username and role>

}

* **Response details:**
* Success response – 200 ok

{

[

{

"id": UUID,

"mid": UUID,

"bankCode": string,

"branchCode": string,

"ifscCode": string,

”isPrimary”: number (0,1),

"status": number(0,1)

"addressLine1": string,

"addressLine2": string,

"state": string,

"city": string,

"country": string

"pincode": number,

"mobileNumber": number,

"phoneNumber": number,

"primaryEmail": string,

"secondaryEmail": string,

"createdAt": number,

"createdBy": string,

"updatedAt": number,

"updatedBy": string,

"approvedAt": number

"approvedBy": string,

"rmName": string

}

],

status: 1

}

* **Example Response** – Error 200 ok

{

  "errors": [

    {

      "errorCode":"ERROR\_CODE ",

      "errorMessage":"ERROR\_MSG"

    },

  ],

"status":0

}

Sample response

|  |
| --- |
| {  "status": 1,  "data": [  {  "accountHolderName": "Test",  "accountUniqueIdentifier": "sbi345",  "accountType": "SAVING",  "accountNumberMasked": "xxxxxxx0342",  "bankName": "CANARA BANK ",  "branchName": "KOLSHET,THANE",  "ifscCode": "CNRB0000255",  "status": "APPROVAL\_PENDING",  "primary": **false**,  "approved": **true**,  "mid": "1000003"  },  {  "accountHolderName": "Sbi",  "accountUniqueIdentifier": "sbi1",  "accountType": "SAVING",  "accountNumberMasked": "xxxxxxxx1234",  "bankName": "THE SARASWAT CO-OPERATIVE BANK LTD",  "branchName": "VIMAN NAGAR",  "ifscCode": "SRCB0000342",  "status": "APPROVAL\_PENDING",  "primary": **false**,  "approved": **false**,  "mid": "1000003"  }  ],  "count": 2,  "total": 2  } |

##### Validation

* The API can only be accessed by users authenticated via the Merchant Access Token.
* Check mId is active and has access for merchant information.

##### Business logic

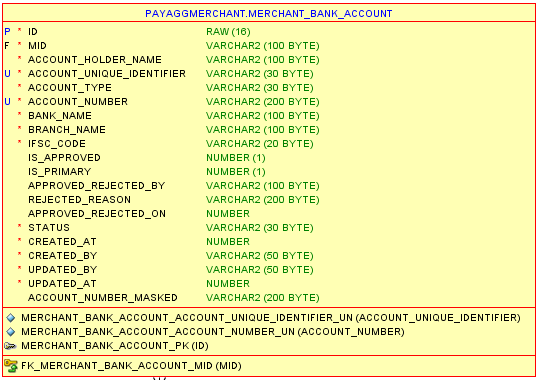
* **Step 1: Merchant bank controller**

Request reach Merchant controller and call Merchant service.

* **Step 2: Merchant bank service**
* Fetch all accessible merchant information for a merchant user.
* If no data is found, it fetches the merchant info data accessible by the super admin merchant user.
* On successful validation, fetch bank details for merchant identifier.
* **Step 3: Merchant service**

Send successful response otherwise send custom exception.

##### Database table – MERCHANT\_BANK\_ACCOUNT



##### Information

* Log Request

Log all incoming requests, including headers and body parameters (excluding sensitive information) to the database.

* Log Response Log all responses, including generated token details and any error messages except the token.

#### Bank accounts add / link

##### API Definition and implementation

* **Endpoint:** /merchant/v1/bank/account
* **Method:** POST
* **Description:** This API adds a bank account details including bank name, bank account type, account number, branch name, and IFSC Code associated with a specific merchant.
* **Request details**
* Request Body:

{

"mId":"string",

"accountHolderName":"string",

"accountUniqueIdentifier":"string",

"accountType":"string",

"accountNumber":"string",

"bankName":"string",

"branchName":"string",

"ifscCode":"string",

“isPrimary": boolean

}

* **Request header:** Token contains valid username.

{

"Authorization": Bearer <Token with username and role>

}

Sample Request:

|  |
| --- |
| {  "accountHolderName": "Test",  "accountNumber": "AAAAAAAAAAAAAAAAAAAAAHJUJzXJ0VpDxGYE+GoCQ8XUddFcwVvc6qEelw==",  "accountType": "SAVING",  "accountUniqueIdentifier": "123456789661",  "bankName": "SBItest",  "branchName": "Belapur",  "ifscCode": "SBIN0011513",  "primary": "True",  "mid": "1000003"  } |

* **Sample Response details:**
* Success response – 200 ok

{

“message”: “Add Merchant Bank Account Successfully”,

“status”: 1

}

* **Example Response** – Error 200 ok

{

  "errors": [

    {

      "errorCode":"ERROR\_CODE ",

      "errorMessage":"ERROR\_MSG"

    },

  ],

"status":0

}

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Description** | **Mandatory** | **Data Type** |
| mId | Merchant ID | Yes | string |
| AccountHoldername | Account holder name | Yes | string |
| accoutUniqueindentifier | Account unique identifier | Yes | string |
| accountType | Account type | Yes | String |
| AccountNumber | Account number | Yes | string |
| bankName | Bank name | Yes | string |
| BranchName | Branch name | Yes | string |
| ifscCode | IFSC code | Yes | string |
| isPrimary | Is primary | No | Boolean |

##### Validation

* The API can only be accessed by users authenticated via the Merchant Access Token.
* Check for uniqueness of account number for account to be created.
* Also check account type should be saving or current.
* Check Bank request payload for bank name, branch name, IFSC code,

account number and merchant identifiers.

##### Business Logic

* **Step 1: Merchant controller**

Request reaches Merchant controller and calls the Merchant service.

* **Step 2: Merchant service**

Service calls validator to check the merchant id.

* **Step 3: Merchant Validator**

Validates if merchant id is present, mandatory fields and values. Also check if the account already exists, bank details.

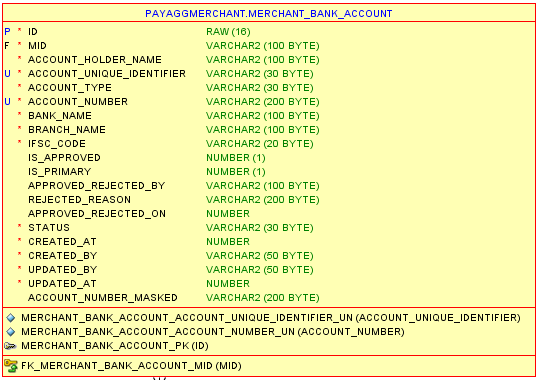
* **Step 4: Merchant DAO**

Saves bank account details by mid.

* **Step 5: Merchant service**

Send successful response otherwise send custom exception.

##### Database Table – MERCHANT\_BANK\_ACCOUNT



##### Information

* + Log Request:

Log all incoming requests, including headers and body parameters (excluding sensitive information) to the database.

* + Log Response:

Log all responses, including generated token details and any error messages except the token.

#### Bank Accounts Update

##### API Definition and implementation

* **Endpoint:** /merchant/v1/bank/account
* **Method:** PUT
* **Description:** This API updates the merchant's bank account details including bank name, bank account type, account number, branch name, and IFSC Code in the EPay system.
* **Request details**
* Request Body:

{

"mId":"string"

"accountHolderName":"string"

"accountUniqueIdentifier":"string"

"accountType":"string"

"accountNumber":"string"

"bankName":"string"

"branchName":"string"

"ifscCode":"string"

“isPrimary": boolean

}

* Request header: Token contains valid username.

{

"Authorization": Bearer <Token with username and role>

}

* **Sample Request details**

|  |
| --- |
| {  "accountHolderName": "Test",  "accountNumber": "AAAAAAAAAAAAAAAAAAAAAHJUJzXJ0VpDxGYE+GoCQ8XUddFcwVvc6qEelw==",  "accountType": "SAVING",  "accountUniqueIdentifier": "123456789661",  "bankName": "SBItest",  "branchName": "Belapur",  "ifscCode": "SBIN0011513",  "primary": **false**,  "mid": "1000003"  } |

* **Sample Response details**
* Success response – 200 ok

{

“message”: “Updated Merchant Bank Account Successfully”,

“status”: 1

}

* **Example Response** – Error 200 ok

{

  "errors": [

    {

      "errorCode":"ERROR\_CODE ",

      "errorMessage":"ERROR\_MSG"

    },

  ],

"status":0

}

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Description** | **Mandatory** | **Data Type** |
| mId | Merchant ID | Yes | string |
| AccountHoldername | Account holder name | Yes | string |
| accoutUniqueindentifier | Account unique identifier | Yes | string |
| accountType | Account type | Yes | String |
| AccountNumber | Account number | Yes | string |
| bankName | Bank name | Yes | string |
| BranchName | Branch name | Yes | string |
| ifscCode | IFSC code | Yes | string |
| isPrimary | Is primary | No | Boolean |

##### Validation

* The API can only be accessed by authenticated users via the Merchant Access Token.
* Check for uniqueness of account number for account to be updated.
* Also check account type should be saving or current.
* Check Bank request payload for bank name, branch name, IFSC code,

account number and merchant identifiers.

##### Business logic

* **Step 1: Merchant controller**

Request reaches Merchant controller and calls the Merchant service.

* **Step 2: Merchant service**

Service call validator to check merchant id.

* **Step 3: Merchant Validator**

Validates if merchant id is present, mandatory fields and values. Also check if the account already exists, bank details.

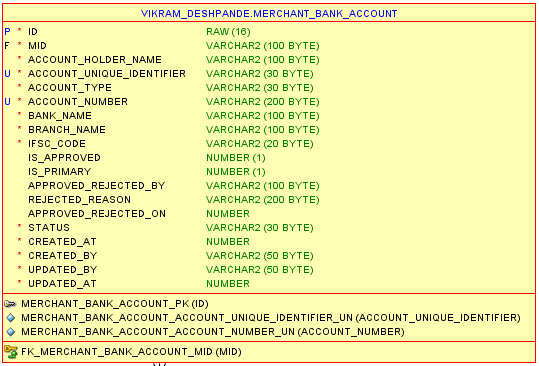
* **Step 4: Merchant DAO**

Saves bank account details by mid.

* **Step 5: Merchant service**

Send successful response otherwise send custom exception.

##### Database Table – MERCHANT\_BANK\_ACCOUNT



##### Information

* Log Request:

Log all incoming requests, including headers and body parameters (excluding sensitive information) to the database.

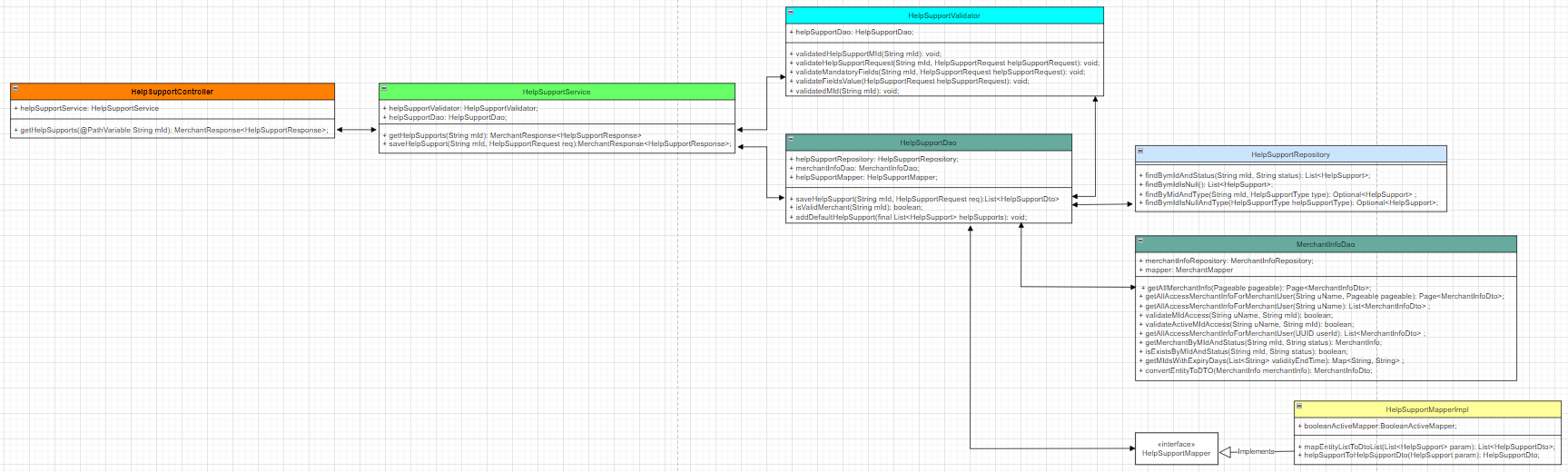
* Log Response:

Log all responses, including generated token details and any error messages except the token.

## Help & Support

### Get Help & Support

#### Class diagram



#### API Definition and implementation

* **Endpoint**: /merchant/v1/help/{mid}
* **Method**: GET
* **Description**: This API allows a merchant user to get the help and support contact details, i.e., phone number and email address. .
* **Request Details**
  + **Parameter: mId-** merchant Identifier

**Content-Type:** application/json

**Authorization** (String): Bearer <JWT Token>

* **Response**
* **Example response** – Success 200 ok

{

    "status": 1,

    "data": [

        {

            "type": "PHONE\_NUMBER",

            "value": "String"

        },

{

            "type": "EMAIL",

            "value": "String"

        }

    ],

    "count": 1

}

* **Example Response** – Error 200 ok

{

  "errors": [

    {

      "errorCode":"ERROR\_CODE ",

      "errorMessage":"ERROR\_MSG"

    },

  ],

"status":0

}

Sample response example

|  |
| --- |
| {  "status": 1,  "data": [  {  "helpSupports": [  {  "type": "PHONE\_NUMBER",  "value": "+91-22-20876156",  "status": **true**  },  {  "type": "EMAIL",  "value": "[support.sbiepay@sbi.co.in](mailto:support.sbiepay@sbi.co.in)",  "status": **true**  },  {  "type": "IMPORTANT\_NOTICE",  "value": "New Feature : Check the health status of all payment channels in one place.",  "status": **true**  },  {  "type": "IMPORTANT\_NOTICE",  "value": "New Feature : Real-time transaction tracking now available in Dashboard.",  "status": **true**  },  {  "type": "IMPORTANT\_NOTICE",  "value": "Feature Enhancement: Faster settlement reports with detailed insights.",  "status": **true**  },  {  "type": "IMPORTANT\_NOTICE",  "value": "Notice: RBI mandates additional authentication for high-value transactions.",  "status": **true**  },  {  "type": "IMPORTANT\_NOTICE",  "value": "Notice: NPCI announces new limits 5000 Rs. for UPI transactions.",  "status": **true**  }  ]  }  ],  "total": 7  } |

##### Validation

* Header should have the valid JWT token.
* Checks if MID is valid and active, if not returns error message.
* The API can only be accessed by authenticated users via the Merchant Access Token.

##### Business logic

* **Step 1: HelpSupport controller**

Request reaches HelpSupport controller and calls the HelpSupport service.

* **Step 2: HelpSupport service**

Service validates request for mandatory fields, call validator to check fields.

* **Step 3: HelpSupport Validator**
* Validates if merchant id is present.
* **Step 4: Theme** **Service**

Call HelpSupport DAO to get help support record.

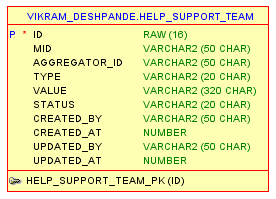
* **Step 5: HelpSupport DAO**

Get help and support by mid and type.

* **Step 7: HelpSupport service**

Send successful response otherwise send custom exception.

##### Database Table – HELP\_SUPPORT\_TEAM



##### Information

* If valid, extract user details and set them in the security context.
* If invalid, reject the request with an unauthorized response.
* Log Request

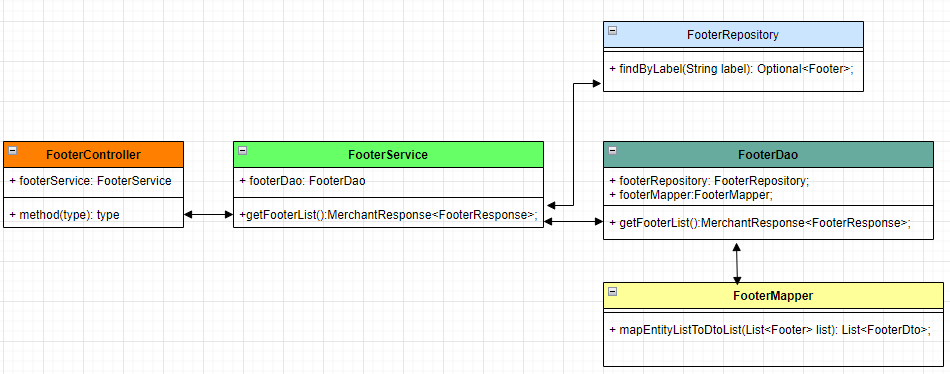
Log all incoming requests, including headers and body parameters (excluding sensitive information) to the database.

* Log Response

Log all responses, including generated token details and any error messages except the token.

### Merchant Portal - Footer info

#### Class diagram



#### API Definition and implementation

* **Endpoint**: /merchant/v1/footer
* **Method**: GET
* **Description:** This API displays thefooter list including About Us, Terms and Conditions, Privacy Policy, Disclosure, and Disclaimer for a specific merchant.**Request Details**

**Content-Type**: application/json

* **Authorization (String):** Bearer <JWT Token>
* **Request body:** Empty
* **Response**
* **Example response:** Success 200 Ok (Updated content)

{

"status": 1,

"data": [

{

"listOfLabels": [

{

"url": "www.aboutus.com"

},

{

"url": "www.termandconditions.com"

},

{

"url": "www.Privacy.com"

},

{

"url": "www.disclosure.com"

},

{

"url": "www.disclaimer.com"

}

]

}

]

}

* **Example Success response**

|  |
| --- |
| {  "status": 1,  "data": [  {  "listOfLabels": [  {  "label": "About Us",  "value": "#"  },  {  "label": "Terms & Conditions",  "value": "#"  },  {  "label": "Privacy Policy",  "value": "#"  },  {  "label": "Disclosure",  "value": "#"  },  {  "label": "Disclaimer",  "value": "#"  }  ]  }  ],  "total": 5  } |

* **Example response:** Error – 200 ok

{

"errors": [

{

"errorCode":"ERROR\_CODE ",

"errorMessage":"ERROR\_MSG"

},

],

"status":0

}

##### Validation

* Header should have the valid JWT token.
* JWT token should have the Merchant as a role.
* The API can only be accessed by authenticated users via the Merchant Access Token.

##### Business logic

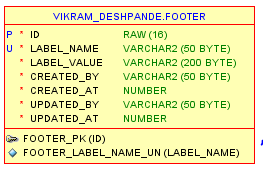
* **Step 1: Footer controller**

Request reaches AboutUs controller and calls the AboutUs service.

* **Step 2: Footer service**
* Service validates request for mandatory fields, call validator to check fields.
* Call AboutUs DAO to get help support record.
* **Step 3: AboutUs service**

Send successful response otherwise send custom exception.

##### Database Table - FOOTER



##### Information

* If valid, extract user details and set them in the security context.
* If invalid, reject the request with an unauthorized response.
* Log Request

Log all incoming requests, including headers and body parameters (excluding sensitive information) to the database.

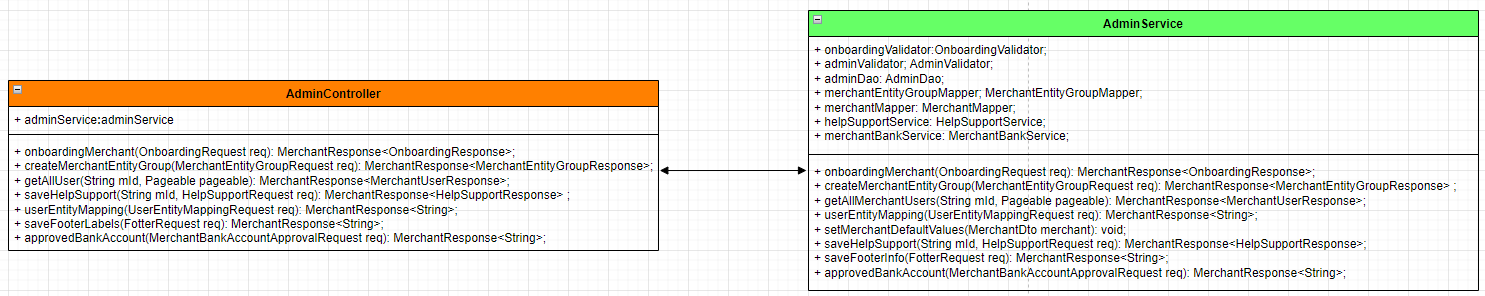
* Log Response

Log all responses, including generated token details and any error messages except the token.

# SD\_MS – SBI Admin Portal – Integration



## Class diagram



## Merchant Portal – Merchant Admin Entity Creation

### API Definition and implementation

* **Endpoint**: /merchant/v1/admin/entity
* **Method**: POST
* **Description:** This API allows in merchant entity creation by grouping multiple mIds together. Each such merchant entity can be managed by a common single Super Admin.
* **Request Details:**

Content-Type: application/json

Authorization (String): Bearer <JWT Token>

* **Request body:**

{

"entityId": " IRCTC-1",

" m**i**ds": [

"10000079"

]

}

* **Response** 
  + **Example** **response –** Success 200 ok

{

"status": 1,

"data": [

{

"entityId": "IRCTC-1",

"mids": [

"10000079"

]

}

],

"count": 1

}

* + **Example Response –** Error 200 ok

{

"errors": [

{

"errorCode":"ERROR\_CODE ",

"errorMessage":"ERROR\_MSG"

},

],

"status":0

}

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Description** | **Mandatory** | **Data Type** |
| entityId | entityId for the request. | Yes | String |
| mIds | list of mids | Yes | List<String> |

#### Validation

* The API can only be accessed by authenticated users via the Merchant Access Token.
* Only active MIDs can be associated with an entity.
* Each MID can belong to only one entity.

#### Business logic

* **Step 1: Admin controller**

Request reaches merchant user admin controller and calls the merchant user service.

* **Step 2: Admin service**

Service validates request for mandatory fields, call DAO to check user.

* **Step 3: Admin Validator**

Validates entity if exists and merchant ids will be checked if any existing association.

* **Step 4: Admin service**

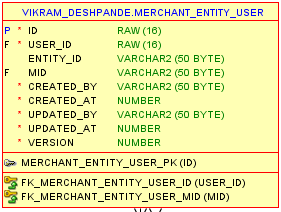
Calladmin DAO to save list of merchants associated to entity.

* **Step 5: Admin DAO**

Save merchant IDs associated to be associated to merchant entity.

* **Step 6: Admin service –** Send Success response otherwise send custom exception.

#### Database Table: MERCHANT\_ENTITY\_USER



#### Information

* + If valid, extract user details and set them in the security context.
  + If invalid, reject the request with an unauthorized response.
  + Log Request
  + Log all incoming requests, including headers and body parameters (excluding sensitive information) to the database.
  + Log Response
  + Log all responses, including generated token details and any error messages except the token.

## Merchant Portal – Admin Onboarding

### API Definition and implementation

* **Endpoint**: /merchant/v1/admin/onboarding
* **Method**: POST
* **Description**: This API allows you to add an Admin during the merchant onboarding process.
* **Request Details:** 
  + - Content-Type: application/json
    - Authorization (String): Bearer <Merchant User - JWT Token>
    - Request body:

{

"merchant": {

"mid": "129810",

"merchantName": "Copier Intl pvt ltd",

"businessName": " Copier Intl pvt ltd ",

"brandName": "Copier1",

"businessCategory": "banking",

"categoryCode": "String",

"addressline1": "String",

"addressline2": "String",

"state": "String",

"city": "String",

"country": "String",

"pincode": "123456",

"mobileNumber": "8806302443",

"phoneNumber": "0228802356",

"primaryEmail": "abc@xylko.com",

"secondaryEmail": "abc@x0pox.com",

"merchantUrl": "String",

"validityStartTime": 1735046816200,

"validityEndTime": 1766582750000,

"onboardingTime": 1735046816200,

"encryptedAlgo": "String",

"rmName": "String",

"bankCode": "String",

"branchCode": "String",

"gstNumber": "String",

"isChargebackAllowed": true,

"aggregator": "String",

"notification": "String"

},

"user": {

"userName": "RakeshG",

"firstName": "V",

"middleName": "V",

"lastName": "V",

"email": "abc@xyo0px.com",

"primaryPhone": "String",

"secondaryPhone": "1234567890",

"mobilePhone": "1234567890",

"officePhone": "1234567890",

"addressline1": "String",

"addressline2": "String",

"state": "String",

"city": "String",

"country": "String",

"pincode": "124563"

}

}

**Response:** Success 200 – Ok

{  
 "data": [  
 {  
 "merchant":{  
 "id": UUID  
 "mid":"String",  
 "merchantName":"String",  
 "businessName":"String",  
 "brandName":"String",  
 "businessCategory":"String",  
 "categoryCode":"String",  
 "addressline1":"String",  
 "addressline2":"String",  
 "state":"String",  
 "city":"String",  
 "country":"String",  
 "pincode":"String",  
 "mobileNumber":"String",  
 "phoneNumber":"String",  
 "primaryEmail":"String",  
 "secondaryEmail":"String",  
 "merchantUrl":"String",

"status":"String",  
 "validityStartTime":"long",  
 "validityEndTime":"long",  
 "onboardingTime":"long",  
 "encryptedAlgo":"String",  
 "rmName":"String",  
 "bankCode":"String",  
 "branchCode":"String",  
 "gstNumber":"String",  
 "isChargeBackAllowed":"String",  
 "agrregatorId":"String",  
 "notification":"String"  
 },  
 "user":{  
 "id": UUID  
 "userName":"String",  
 "firstName":"String",  
 "middleName":"String",  
 "lastName":"String",  
 "email":"String",  
 "primaryPhone":"String",  
 "secodaryPhone":"String",  
 "mobilePhone":"String",  
 "addressline1":"String",  
 "addressline2":"String",  
 "state":"String",  
 "city":"String",  
 "country":"String",  
 "pincode":"String"  
 }  
 }  
 ],  
 "status":1,  
 "count": 1,  
 "total": 1

}

* + **Example Response –** Error 200 ok

{

"errors": [

{

"errorCode":"ERROR\_CODE ",

"errorMessage":"ERROR\_MSG"

},

],

"status":0

}

Sample request

|  |
| --- |
| {  "merchant": {  "addressline1": "123 Main Street",  "aggregator": "SBIEPAY",  "bankCode": "BANK123",  "branchCode": "BR123",  "city": "Los Angeles",  "country": "USA",  "mId": "MID6",  "merchantName": "John Doe Enterprises",  "mobileNumber": "9876543210",  "onboardingTime": 1672531200000,  "pincode": "123456",  "primaryEmail": "[primary@example.com](mailto:primary@example.com)",  "rmName": "Alice Johnson",  "state": "California",  "validityEndTime": 1747366148271,  "validityStartTime": 1737366148271,  "businessName": "John's Electronics",  "brandName": "J-Electro",  "businessCategory": "Retail",  "categoryCode": "R123",  "addressline2": "Suite 456",  "phoneNumber": "9876543210",  "secondaryEmail": "[secondary@example.com](mailto:secondary@example.com)",  "merchantUrl": "<https://www.j-electro.com>",  "status": "ACTIVE",  "encryptedAlgo": "AES256",  "gstNumber": "27ABCDE1234F1Z5",  "notification": "EMAIL",  "createdBy": "admin",  "createdAt": 1672531200000,  "updatedBy": "admin",  "updatedAt": 1672627600000,  "isChargebackAllowed": **true**  },  "user": {  "email": "[test304@gmail.com](mailto:test304@gmail.com)",  "firstName": "John",  "mobilePhone": "9619517305",  "userName": "ManualTest",  "emailMasked": "[test304@gmail.com](mailto:test304@gmail.com)",  "mobilePhoneMasked": "\*\*\*\*\*\*\*304",  "remark": "Verified user",  "roleName": "Admin/SuperAdmin/User",  "status": "ACTIVE",  "passwordExpiryTime": 1704067200000,  "createdBy": "Neeraj",  "createdAt": 1672531200000,  "updatedBy": "Neeraj",  "updatedAt": 1672627600000,  "lastSuccessLogin": "1672627600000",  "firstLogin": "True"  }  } |

Sample response

|  |
| --- |
| {  "status": 1,  "data": [  {  "merchant": {  "merchantName": "John Doe Enterprises",  "businessName": "John's Electronics",  "brandName": "J-Electro",  "businessCategory": "Retail",  "categoryCode": "R123",  "addressline1": "123 Main Street",  "addressline2": "Suite 456",  "state": "California",  "city": "Los Angeles",  "country": "USA",  "pincode": "123456",  "mobileNumber": "9876543210",  "phoneNumber": "9876543210",  "primaryEmail": "[primary@example.com](mailto:primary@example.com)",  "secondaryEmail": "[secondary@example.com](mailto:secondary@example.com)",  "merchantUrl": "<https://www.j-electro.com>",  "status": "ACTIVE",  "validityStartTime": 1737366148271,  "validityEndTime": 1747366148271,  "onboardingTime": 1672531200000,  "encryptedAlgo": "AES256",  "rmName": "Alice Johnson",  "bankCode": "BANK123",  "branchCode": "BR123",  "gstNumber": "27ABCDE1234F1Z5",  "aggregator": "SBIEPAY",  "notification": "EMAIL",  "createdBy": "Neeraj1",  "createdAt": 1742974857469,  "updatedBy": "Neeraj1",  "updatedAt": 1742974857469,  "mId": "MID6",  "isChargebackAllowed": **true**  },  "user": {  "userName": "ManualTest",  "firstName": "John",  "email": "[test304@gmail.com](mailto:test304@gmail.com)",  "emailMasked": "t\*\*\*\*\*04@\*\*\*\*\*om",  "mobilePhone": "9619517305",  "mobilePhoneMasked": "\*\*\*\*\*\*\*305",  "remark": "Verified user",  "status": "ACTIVE",  "passwordExpiryTime": 1750923657493,  "createdBy": "Neeraj1",  "createdAt": 1742974857497,  "updatedBy": "Neeraj1",  "updatedAt": 1742974857497,  "lastSuccesssLogin": 1672627600000,  "firstLogin": **true**  }  }  ],  "total": 1  } |

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Description** | **Mandatory** | **Data Type** |
| mId | entityId for the request. | Yes | String |
| merchantName | Merchant name | Yes | String |
| businessName | Business Name | No | String |
| brandName | Brand Name | No | string |
| businessCategory | Business category | No | string |
| categoryCode | Category code | No | string |
| addressline1 | Address line 1 | Yes | string |
| addressline2 | Address line 2 | No | string |
| state | State | Yes | String |
| city | City name | Yes | String |
| country | country | Yes | String |
| pincode | Pin code | Yes | String |
| mobileNumber | Mobile number | Yes | String |
| phoneNumber | Phone number | No | String |
| primaryEmail | Primary email address | Yes | String |
| secondaryEmail | Secondary email address | No | string |
| merchantUrl  status | Merchant website URL  Merchant Status | No | String  MerchantStatusACTIVE, INACTIVE, BLOCKED, EXPIRED; |
| validityStartTime | Validity Start Time | Yes | Long |
| validityEndTime | Validity End Time | Yes | Long |
| onboardingTime | Onboarding time | Yes | Long |
| encryptedAlgo | Encryption algorithm used | No | String |
| rmName | Relationship Manager's name | Yes | String |
| bankCode | Bank code | Yes | String |
| branchCode | Branch code | Yes | String |
| gstNumber | GST number | No | String |
| isChargebackAllowed | Chargeback allowed flag | No | Boolean |
| aggregator | Aggregator name | Yes | String |
| notification | Notification preferences | No | String |
| createdBy | Created by user | No | string |
| createdAt | Creation timestamp | No | Long |
| updatedBy | Updated by user | No | String |
| updatedAt | Updated timestamp | No | Long |
| id | UUID unique Identifier | No | UUID |
| parentUserId | UUID parent user Identifier | No | UUID |
| version | Version | No | int |
| userName | Unique username | Yes | String |
| firstName | User first name | Yes | String |
| email | User email address | Yes | String |
| emailMasked | User mask email address | No | String |
| mobilePhone | User mobile number | Yes | String |
| mobilePhoneMasked | Mask mobile number | No | String |
| remark | Remarks or notes | No | String |
| role | User Role ID | No | UUID |
| roleName | Role Name | No | String |
| status | User Status | No | Enum userStatus  ACTIVE, BLOCKED |
| password | Encrypted Password | No | String |
| passwordExpiryTime | Password expiry time |  | Long |
| loginFailAttempt | Login failure attempt count | No | int |
| isFirstLogin | User doing first Login | No | Boolean |
| lastSuccessLogin | User last Success login | No | Long |
| email | User email address | No | String |

#### Validation

* Check if MID already associated, if yes return error message
* Check the status of merchant ID.
* The API can only be accessed by users authenticated via the Merchant Access Token.
* Only active MIDs can be associated with an entity.
* Each MID can belong to only one entity.

#### Business L

* **Step 1: Admin controller**

Request reaches merchant user admin controller and calls the admin user service.

* **Step 2: Admin service**

Service calls validator check onboarding request.

* **Step 3: Admin validator**

Validator checks for mandatory fields, merchant exists by MID.

* **Step 4: Admin service**

Service set default values like notification, encryption algo and status if not set already. Once done call DAO.

* **Step 5: Admin DAO**

Save merchant info and call merchant user DAO

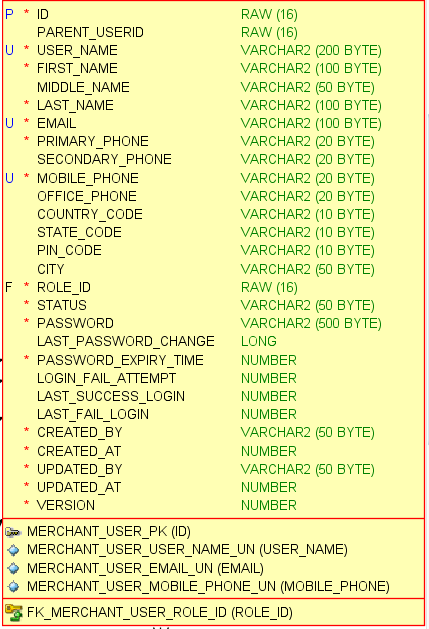
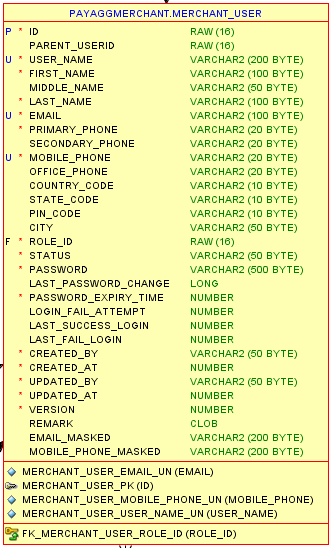
* **Step 6: Merchant user DAO**

In DAO, password will be created, along with default values will be assigned to Merchant user.

* **Step 7: Admin service**

Send success response otherwise send custom exception.

#### Database Table - MERCHANT\_USER and MERCHANT\_INFO



#### Information

* + If valid, extract user details and set them in the security context.
  + If invalid, reject the request with an unauthorized response.
  + Log Request

Log all incoming requests, including headers and body parameters (excluding sensitive information) to the database.

* + Log Response

Log all responses, including generated token details and any error messages except the token.

## Merchant Portal: Account Setting - Admin User Entity Mapping

### API Definition and implementation

* **Endpoint**: /merchant/v1/admin/entity/mapping
* **Method:** POST
* **Description:** This APIallows a merchant user to be mapped with an entity and grant a Super Admin role to that user. This provides the enhanced permissions to that user to manage entity-specific operations efficiently.
* **Request details:** 
  + **Path Parameter**: type (values: API or Encryption)
  + **Authorization**: Bearer <Merchant Access Token (Admin Portal Authorization) >
  + **Request object**:

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Description** | **Mandatory** | **Data Type** |
| userName | Unique merchant for the request. | Yes | string |
| entityId | Map a user to a specific entity | Yes | string |

* + **Request body:**

{   
 "userName": "string",   
 "entityId": "string"   
}

**Response details**

* Successs – 200 Ok

{

"data": [

"UserObject": {},

"EntityObject": {}

],

"status":1,

"count": 1,

"total": 1

}

* Error / exception – 200 Ok

{

"errors": [

{

"errorCode":"ERROR\_CODE ",

"errorMessage":"ERROR\_MSG"

},

],

"status":0

}

Sample request

|  |
| --- |
| {  "entityId": "ENTITY001",    "userName": "ManualTest"  } |

Sample response

|  |
| --- |
| {  "status": 1,  "data": [  "role has been updated Successfully"  ],  "total": 1  } |

#### Validation

##### Request Data Validation

* Check if entity id is valid.
* In request user id or user should not blank, also check merchant user in database.
* The API can only be accessed by authenticated users via the Merchant Access Token.
* Only active MIDs can be associated with an entity.
* One MID can belong to only one entity.

#### Business Logic

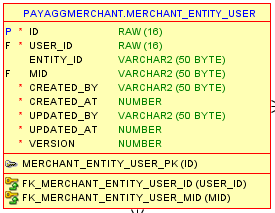
* **Step 1: Admin controller**

Request reaches merchant user admin controller and calls the merchant user service.

* **Step 2: Admin service**
* Call DAO to fetch merchant user.Check for role of merchant user.
* User with super admin role, associate entity with user.
* For user with admin role, update user role to super admin, delete existing menu permission and assign default menu permission.
* **Step 3: Admin service**

Send successs response otherwise send custom exception.

#### Database Table - MERCHANT\_ENTITY\_USER



#### Information - None

## Merchant Portal - List of User for Merchant for SBI Admin

### API Definition and Implementation

* **Endpoint:** /merchant/v1/admin/users/{mId}
* **Method: GET**
* **Description:** This API allows the SBI Admins to get a list of users associated with a given merchant. A merchant with an Admin access privileges can perform this task via the SBI Admin Portal.
* **Request details** 
  + **Authorization:** Bearer <Merchant Access Token (Admin Portal Authorization)
  + **Path Parameter:** The Merchant ID, used to identify the merchant account.
* **Query Parameters** 
  + **page**: (Integer) Optional. The page number for pagination. Defaults to 0 if not provided.
  + **size**: (Integer) Optional. The size of the page. Default value is 50, with a minimum value of 50 and a maximum value of 100 (configured in application properties).
* **Response details**
* **Response –** Success – 200 Ok

{

"status": 0,

"data": [

{

"userName": "string",

"firstName": "string",

"middleName": "string",

"lastName": "string",

"email": "string",

"primaryPhone": "string",

"mobilePhone": "string",

"countryCode": "string",

"stateCode": "string",

"pinCode": "string",

"city": "string",

"role": "string",

"status": "string",

"createdAt": 0,

"updatedAt": 0

}

],

"count": 0,

"total": 0,

}

Sample response

|  |
| --- |
| {  "status": 1,  "data": [  {  "userName": "TestOne",  "firstName": "Test",  "email": "[Testone@abc.in](mailto:Testone@abc.in)",  "emailMasked": "T\*\*\*\*\*ne@\*\*\*\*\*in",  "mobilePhone": "9874561230",  "mobilePhoneMasked": "\*\*\*\*\*\*\*230",  "roleName": "USER",  "status": "ACTIVE",  "remark": "[STATUS UPDATE from INACTIVE to ACTIVE ]Remark:fghh",  "createdAt": 1741260759885,  "updatedAt": 1741349172749,  "firstLogin": **true**  },  {  "userName": "Rakesh",  "firstName": "Rakesh Kumar ",  "email": "[rakesh@gmail.inmmmmmmmmmoo](mailto:rakesh@gmail.inmmmmmmmmmoo)",  "emailMasked": "r\*\*\*\*sh@\*\*\*\*oo",  "mobilePhone": "9956265729",  "mobilePhoneMasked": "\*\*\*\*\*\*\*729",  "roleName": "ADMIN",  "status": "ACTIVE",  "remark": "Verified user",  "createdAt": 1740477993427,  "updatedAt": 1742570815526,  "firstLogin": **false**  }  ],  "count": 2,  "total": 2  } |

* Response – Error 200

{

"errors": [

{

"errorCode": "string",

"errorMessage": "string"

}

]

}

### Validation

* The API can only be accessed by authenticated users via the Merchant Access Token.

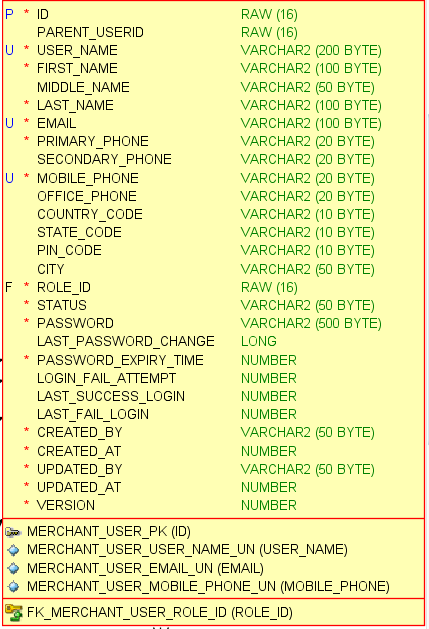
### Business Logic

* **Step 1: Admin controller**

Request reaches merchant user admin controller and calls the merchant user service.

* **Step 2: Admin service**
* Calladmin DAO to retrieve list of merchant users associated to mId.
* Send success response with list of otherwise send custom exception.

### Database Table – MERCHANT\_USER



#### Information

* + If valid, extract user details and set them in the security context.
  + If invalid, reject the request with an unauthorized response.
  + Log Request

Log all incoming requests, including headers and body parameters (excluding sensitive information) to the database.

* + Log Response

Log all responses, including generated token details and any error messages except the token.

## Update Help & Support

### API Definition and Implementation

* **Endpoint**: /merchant/v1/admin/help/{mid}
* **Method**: PUT
* **Description**: This API allows merchant users to update help and support information for the associated mId, including the phone number and email Id. ~~Merchant users can update help and support information by mid which includes phone number and email~~.
* **Request Details:**

Content-Type: multipart/form-data

Authorization (String): Bearer <JWT Token>

* **Request body:**

{

"supportTeams": [

{

"type": "PHONE\_NUMBER",

"value": "xxx-xxxxxxx"

},

{

"type": "EMAIL",

"value": "support.sbiepay@sbi.co.in"

}

]

}

* **Response**:
* **Example response** – Success 200 ok

{

    "status": 1,

    "data": [

        {

            "type": " EMAIL",

            "value": "String"

        }, {

            "type": "PHONE\_NUMBER",

            "value": "String"

        }

    ],

    "count": 1

}

* **Example Response** – Error 200 ok

{

  "errors": [

    {

      "errorCode":"ERROR\_CODE ",

      "errorMessage":"ERROR\_MSG"

    },

  ],

"status":0

}

Sample request

|  |
| --- |
| {  "helpSupports": [  {  "aggregatorId": "SBIEPAY",  "type": "EMAIL",  "value": "[test@gmail.com](mailto:test@gmail.com)",  "status": "true",  "mid": "MID6"  }  ]  } |

Sample response

|  |
| --- |
| {  "status": 1,  "data": [  {  "helpSupports": [  {  "type": "EMAIL",  "value": "[test@gmail.com](mailto:test@gmail.com)",  "status": **true**,  "mid": "MID6"  }  ]  }  ],  "total": 1  } |

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Description** | **Mandatory** | **Data Type** |
| mId | Merchant ID for the request. | No | String |
| aggregatorId | aggregatorId | No | String |
| type | Valid HelpSupportType are PHONE\_NUMBER, EMAIL | No | HelpSupportType |
| value | Notes or value | No | String |
| Status | status | No | Boolean |

### Validation

* Header should have the valid JWT token.
* Checks if MID is present and valid, if not returns an error message.
* Check Email and Phone with regex pattern.
* The API can only be accessed by users authenticated via the Merchant Admin Access Token.
* After token user can update the help and support information.

### Business logic

* **Step 1: Help Support controller**

Request reaches help & support controller and calls the help support service.

* **Step 2: Help Support service**

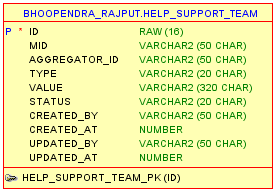
Service validates request for mandatory fields, call DAO to check user.

* **Step 3: Help Support Validator**

Validate phone number and email with pattern. Call admin DAO to check merchant id exists.

* **Step 4: HelpSupport service**
* Call DAO to update the help support information.
* Send successful response with support information otherwise send custom exception.

### Database Table - HELP\_SUPPORT\_TEAM



### Information

* If valid, extract user details and set them in the security context.
* If invalid, reject the request with an unauthorized response.
* Log Request

Log all incoming requests, including headers and body parameters (excluding sensitive information) to the database.

* Log Response

Log all responses, including generated token details and any error messages except the token.

## Merchant Portal - Footer

### API Definition and Implementation

* **Endpoint**: /merchant/v1/admin/footer
* **Method**: POST
* **Description**: This API helps you to update the footer information for the specific merchant.
* **Request Details:**
* **Content-Type**: application/json
* **Authorization** (String): Bearer <JWT Token>
* **Request body:**

{

“footerLabels”: [

{

“label”: “string”,

“value”: “string”

}

]

}

* **Response**:
* **Example response** – Success 200 ok

{

    " label": “string”,

“url”: string

}

* **Example Response** – Error 200 ok

{

  "errors": [

    {

      "errorCode":"ERROR\_CODE ",

      "errorMessage":"ERROR\_MSG"

    },

  ],

"status":0

}

Sample request

|  |
| --- |
| {  "footerLabels": [  {  "label": "testLabel",  "value": "test"  }  ]  } |

Sample response

|  |
| --- |
| {  "status": 1,  "data": [  "Updated Successfully"  ],  "total": 1  } |

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Description** | **Mandatory** | **Data Type** |
| label | Message or label | Yes | String |
| value | value | Yes | String |

### Validation

* The API can only be accessed by users authenticated via the Merchant Admin Access Token.
* Raise custom exception if footer list is empty.
* Process footer list to validate for empty or blank field values.

### Business Logic

* **Step 1: Admin controller**

Request reach help & admin controller’s Footer API and call admin service.

* **Step 2: Admin service**

Service validator called.

* **Step 3: Admin Validator**

Validates request for Footer options.

* **Step 4: Admin service**

Call DAO to update the help support information.

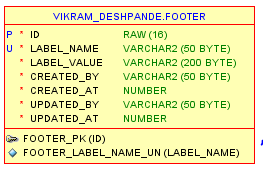
* **Step 5: Admin DAO**

Fetch and update footer option. If not found Create option.

* **Step 6: Admin service**

Send successful response otherwise send custom exception.

### Database Table – FOOTER



### Information

* + Log Request:

Log all incoming requests, including headers and body parameters (excluding sensitive information) to the database.

* + Log Response:

Log all responses, including generated token details and any error messages except the token.

## Merchant Portal – Bank Accounts Listing

### API Definition and Implementation

* **Endpoint:** /merchant/v1/admin/bank/approval
* **Method:** PUT
* **Description: This API displays a list of bank accounts which are associated to the merchant including the account holder name, account identifier and remarks**. ~~API will fetch list of bank accounts associated to merchant.~~
* **Request details:**
* Request header: Token contains valid username.

{

"Authorization": Bearer <Token with username and role>

}

* **Request body details:**

{

"mId":"string",

"isApproved":"boolean",

"accountName":"string",

"accountUniqueIdentifier":"string",

"remarks":"string"

}

* **Example Response** – Success 200 ok, response will be updated payload like request payload.
* **Example Response** – Error 200 ok

{

  "errors": [

    {

      "errorCode":"ERROR\_CODE ",

      "errorMessage":"ERROR\_MSG"

    },

  ],

"status":0

}

Sample response

|  |
| --- |
| { "status": 1,  "data": [  "Bank account updated Successfully"  ],  "total": 1  } |

Sample Request

|  |
| --- |
| {  "accountName": "testAccount",  "accountUniqueIdentifier": "123456789661",  "remarks": "BankAccountApproved",  "approved": **true**,  "mid": "1000003"  } |

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Description** | **Mandatory** | **Data Type** |
| mId | Merchant Id | Yes | String |
| isApproved | value | No | Boolean |
| accountName | accountName | Yes | String |
| accountUniqueIdentifier | accountUniqueIdentifier | Yes | String |
| remarks | Remark or Notes | No | String |

### Validation

* Check MID is present and valid, if not return error message,
* Check mandatory fields mid, account holder name, account identifier. Remark is mandatory if bank account is not approved.

### Business Logic

* **Step 1: Admin controller**

Request reach Admin controller and call admin service.

* **Step 2: Admin service**

Service call’s Merchant service validator to check login user.

* **Step 3: Merchant service**

Service call’s Merchant service validator to check login user.

* **Step 4: Merchant bank account Validator**

Validate merchant id is present. Also, the user should be an admin or super admin to add / update bank account details. On successful validation call DAO.

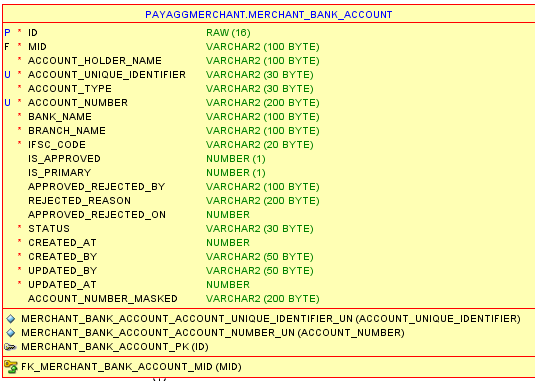
* **Step 5: Merchant info DAO**

Update bank details along with remark as per approval status and set alert for merchant.

* **Step 6: Merchant service**

Send successful response otherwise send custom exception.

### Database Table - MERCHANT\_BANK\_ACCOUNT



### Information

* Log Request:

Log all incoming requests, including headers and body parameters (excluding sensitive information) to the database.

* Log Response:

Log all responses, including generated token details and any error messages except the token.

# Scheduler Service



## Password Expiry Scheduler

This API displays a list of all expired passwords. A scheduler runs every five minutes to select such passwords to make a list. Business logic

**Step-1: Password expiry scheduler**

Scheduler evokes every 30 minutes; password expiry service is called.

**Step-2: Password expiry service**

Call DAO to find records expired captcha records.

**Step-3: Captcha DAO**

Call Password expiry repository to get expired records.

**Step-4: Captcha repository**

Find expired records and delete entries.

### Information

* Log scheduler requests, including start time end time.
* Also log record found and updated.

## Account Expiration Scheduler

This API allows to run a scheduler every 30 minutes to find out a list of all the accounts expiring Account expiration scheduler is a cache cleanup service to delete expired records. This schedule runs every 30 minutes.

### Business Logic

**Step-1: Account expiration Scheduler**

Scheduler evokes every 1 minute; alert service is called to generate notification for user.

**Step-2: Alert service**

Call DAO to find list of expired accounts.

**Step-3: Alert DAO**

Call repository to get expired account list. Use expired account list to find associated merchant identifier from.

**Step-4: Alert service**

Create alerts if merchants are found with expired accounts.

**Step-5: Alert DAO**

Save alerts.

### Information

* Log scheduler requests, including start time end time.
* Also log record found and updated.

# Admin Services



## Merchant – Current Downtime for Paymodes

### API Implementation

* **Endpoint:** /admin/v1/payMode/downtime
* **Method:** GET
* **Description: This API allows you to view the payment modes that are currently unavailable for transactions.** ~~Provides list of paymodes not available or experiencing issue.~~
* **Request body:** None
* **Path parameters:** None
* **Response:** Success 200 Ok

{

    “status”: 1,

    “data”: [

       {

            "srNo":"string"

     "gatewayId":"string"

     "status":"string"

     "startTimestamp":"string"

     "endTimestamp":"string"

     "errorMessage":"string"

     "payModeCode":"string"

      }

    ],

   “count”: 1,

   “length”: 1

}

* **Response:** Error 200 Ok

{

   "errors": [

      {

         "errorCode":"ERROR\_CODE ",

         "errorMessage":"ERROR\_MSG"

      },

    ],

   "status":"0"

}

Sample response

|  |
| --- |
| {  "status": 1,  "data": [  {  "srNo": "3238",  "gatewayId": "140",  "status": "L",  "startTimestamp": "2024-11-08 15:15:18",  "endTimestamp": "2024-11-08 15:20:18",  "errorMessage": "TestApprove-08112024",  "payModeCode": "WALLET"  },  {  "srNo": "1021",  "gatewayId": "76",  "status": "L",  "startTimestamp": "2021-04-15 11:12:19",  "endTimestamp": "2021-04-15 14:17:19",  "errorMessage": "ok",  "payModeCode": "NB"  } |

|  |  |
| --- | --- |
| Error Code | Error Description |
| Respective generic error code and description with field Name | |

### Validation

* Check cache and database if not found raise custom exception.

### Business Logic

* **Step-1: Admin controller**

API provides listing of paymodes downtime.

* **Step-2: Merchant paymode service**
* Query cache to fetch downtime API. If not available query database.

If found in database update cache otherwise raise custom exception.

* Process downtime data and send the same as response.

## Merchant - Paymode Details

### API Implementation

* **Endpoint:** /admin/v1/merchant/payMode/{mId}
* **Method:** POST
* **Description: This API allows to view merchant information along with associated payment modes details.**
* **Request body:** None
* **Path parameters:**
* **mId –** Merchant identifier
* **Response:** – Success 200 Ok

{

    “status”: 1,

    “data”: [

       "merchantInfo": {

          "merchantId":"string",

          "isActive":"string",

          "merchantName":"string",

          "countryCode":"string",

          "currency":"string",

          "preferredPayMode":"string",

          "preferredBank":"string",

          "theme":JsonNode,

          "returnURL":"string",

          "accessTokenExpiryTime":"number",

          "transactionTokenExpiryTime":"number",

          "status":"string",

          "apiKey":"string",

          "secretKey":"string",

          "mek":"string",

          "kek":"string",

          "aek":"string",

          "currencyCodes":"Lis of currency"

          "maxAtrnCount":"number",

          "orderExpiryTime":"number",

          "merchantVolVelFlag":"string",

      },

      "paymodes":{

     "preferredOptions": {

    "upi": {

     "apps": List<String>,

                 "vpa": boolean,

                 "drawerIntent": boolean,

                 "self": {

    "bankName": string,

                    "aggregatorGatewayMapId": string,

                    "payproc": string,

                    "payproctype": string,

                    "downtime": string,

 },

                 "bhimQr:{

    "bankName": string,

                    "aggregatorGatewayMapId": string,

                    "payproc": string,

                    "payproctype": string,

                    "downtime": string,

 }

},

"cards": {

   "cc": [

      {

    "bankName": string,

                    "aggregatorGatewayMapId": string,

                    "payproc": string,

                    "payproctype": string,

                    "downtime": string,

 }

   ],

   "dc": [

      {

    "bankName": string,

                    "aggregatorGatewayMapId": string,

                    "payproc": string,

                    "payproctype": string,

                    "downtime": string,

 }

   ],

   "pc": [

      {

    "bankName": string,

                    "aggregatorGatewayMapId": string,

                    "payproc": string,

                    "payproctype": string,

                    "downtime": string,

 }

   ]

},

"inb": [

  {

"bankName":"string"

                "bankId":"string"

                "aggregatorGatewayMapId":"string"

                "payproc":"string"

                "payproctype":"string"

                "cardType":"string"

                "downtime" boolean,

                "popular": boolean

             }

]

 },

 "otherPaymentOptions": {

    "upi": {

     "apps": List<String>,

                 "vpa": boolean,

                 "drawerIntent": boolean,

                 "self": {

    "bankName": string,

                    "aggregatorGatewayMapId": string,

                    "payproc": string,

                    "payproctype": string,

                    "downtime": string,

 },

                 "bhimQr:{

    "bankName": string,

                    "aggregatorGatewayMapId": string,

                    "payproc": string,

                    "payproctype": string,

                    "downtime": string,

 }

},

"cards": {

   "cc": [

      {

    "bankName": string,

                    "aggregatorGatewayMapId": string,

                    "payproc": string,

                    "payproctype": string,

                    "downtime": string,

 }

   ],

   "dc": [

      {

    "bankName": string,

                    "aggregatorGatewayMapId": string,

                    "payproc": string,

                    "payproctype": string,

                    "downtime": string,

 }

   ],

   "pc": [

      {

    "bankName": string,

                    "aggregatorGatewayMapId": string,

                    "payproc": string,

                    "payproctype": string,

                    "downtime": string,

 }

   ]

},

"inb": [

  {

"bankName":"string"

                "bankId":"string"

                "aggregatorGatewayMapId":"string"

                "payproc":"string"

                "payproctype":"string"

                "cardType":"string"

                "downtime" boolean,

                "popular": boolean

             }

]

 }

  }

    ],

   “count”: 1,

   “length”: 1

}

Sample response

|  |
| --- |
| {  "status": 1,  "data": [  {  "merchantInfo": {  "isActive": "Y",  "merchantName": "MahaOnlineLimited",  "countryCode": "IN",  "currency": "INR",  "preferredPayMode": "UPI",  "preferredBank": "STATE BANK OF INDIA",  "accessTokenExpiryTime": 30,  "transactionTokenExpiryTime": 20,  "maxAtrnCount": 3,  "orderExpiryTime": 30,  "merchantVolVelFlag": "N",  "merchantMultiAccountFlag": "N",  "numberOfAttemptsApplicable": "Y",  "numberOfAttempts": "5",  "isRefundApplicable": "Y",  "refundWindowDays": 30,  "merchantBusinessUrl": "<https://uat.sbiepay.sbi>, <https://dev.epay.sbi/demo/>",  "mId": "1000003"  },  "payModes": {  "preferredOptions": {  "upi": {  "apps": [],  "vpa": **false**,  "drawerIntent": **false**,  "self": **null**,  "BHIMQR": **null**  },  "cards": {  "cc": [],  "dc": [],  "pc": []  },  "inb": [  {  "bankName": "STATE BANK OF INDIA",  "bankId": **null**,  "aggregatorGatewayMapId": **null**,  "payproc": **null**,  "payproctype": **null**,  "cardType": **null**,  "downtime": **null**,  "popular": **null**  }  ]  },  "otherPaymentOptions": {  "upi": {  "apps": [  "PhonePe",  "GPay",  "AmazonPay"  ],  "vpa": **true**,  "drawerIntent": **true**,  "self": {  "bankName": "SBIePay Test Bank",  "aggregatorGatewayMapId": "133",  "payproc": "SELF",  "payproctype": "SELF",  "downtime": **false**  },  "BHIMQR": {  "bankName": "SBIePay Test Bank",  "aggregatorGatewayMapId": "182",  "payproc": "BHIMQR",  "payproctype": "BHIMQR",  "downtime": **false**  }  },  "cards": {  "cc": [  {  "bankName": "State Bank of India New PG",  "aggregatorGatewayMapId": "171",  "payproc": "RUPAY",  "payproctype": "RUPAY",  "downtime": **false**  },  {  "bankName": "State Bank of India New PG",  "aggregatorGatewayMapId": "139",  "payproc": "VISA",  "payproctype": "VISA",  "downtime": **false**  },  {  "bankName": "State Bank of India New PG",  "aggregatorGatewayMapId": "138",  "payproc": "MASTER",  "payproctype": "MASTER",  "downtime": **false**  }  ],  "dc": [  {  "bankName": "State Bank of India WIBMO PG",  "aggregatorGatewayMapId": "226",  "payproc": "MASTER",  "payproctype": "MASTER",  "downtime": **false**  },  {  "bankName": "State Bank of India WIBMO PG",  "aggregatorGatewayMapId": "228",  "payproc": "VISA",  "payproctype": "VISA",  "downtime": **false**  },  {  "bankName": "State Bank of India WIBMO PG",  "aggregatorGatewayMapId": "227",  "payproc": "RUPAY",  "payproctype": "RUPAY",  "downtime": **false**  }  ],  "pc": [  {  "bankName": "State Bank of India New PG",  "aggregatorGatewayMapId": "163",  "payproc": "MASTER",  "payproctype": "MASTER",  "downtime": **false**  },  {  "bankName": "State Bank of India New PG",  "aggregatorGatewayMapId": "172",  "payproc": "RUPAY",  "payproctype": "RUPAY",  "downtime": **false**  },  {  "bankName": "State Bank of India New PG",  "aggregatorGatewayMapId": "164",  "payproc": "VISA",  "payproctype": "VISA",  "downtime": **false**  }  ]  },  "inb": [  {  "bankName": "Mehsana Urban Co. Op. Bank Ltd - Corporate",  "bankId": "119",  "aggregatorGatewayMapId": "115",  "payproc": "SELF",  "payproctype": "SELF",  "cardType": **null**,  "downtime": **false**,  "popular": **false**  }  ]  } |

* + **Response:** Error 200 Ok

{

   "errors": [

      {

         "errorCode":"ERROR\_CODE ",

         "errorMessage":"ERROR\_MSG"

      },

    ],

   "status":"0"

}

|  |  |
| --- | --- |
| Error Code | Error Description |
| Respective generic error code and description with field Name | |
| 1003 | Mid is not found |

### Validation

* Query cache if not available, check database. Otherwise raise appropriate exception.

### Business Logic

* + - **Step-1: Admin controller**

API provides merchant information with paymode details.

* + - **Step-2: Merchant paymode service** 
      * Query database to cache, if not available query database. No data found then raise custom exception.
      * Process data to arrange paymode-wise like CC, DC, PC, UPI and INB.
      * Prepare success response with processed data.

## Merchant – Notification Details

### API Implementation

**Endpoint:** /admin/v1/merchant/notification/{mId}

**Method:** POST

**Description: This API allows you to view the merchant details, including their notification.** ~~Get merchant details for notification.~~

**Request body:** None

**Path parameters:**

**mId –** Merchant identifier

**Response:**

**Success** – 200 Ok

{

"status": 1,

"data": [

"mId": string,

"emailAlertMerchant": string,

"emailAlertCustomer": string,

"smsAlertMerchant": string,

"smsAlertCustomer": string,

"communicationEmail": string,

"mobileNo": string,

"merchPushResponseFlag": string,

"merchPushResponseUrl": string,

"businessName": string,

"brandName": string,

"modifiedDate": date,

"modifiedBy": string,

"creationDate": date,

"createdBy": string

],

"count": 1,

"length": 1

}

**Error** – 200 Ok

{

"errors": [

{

"errorCode":"ERROR\_CODE ",

"errorMessage":"ERROR\_MSG"

},

],

"status":"0"

}

Sample response

|  |
| --- |
| {  "status": 1,  "data": [  {  "emailAlertMerchant": "Y",  "emailAlertCustomer": "Y",  "smsAlertMerchant": "Y",  "smsAlertCustomer": "Y",  "communicationEmail": "[ebms\_uat\_receiver@ebmsgits.sbi.co.in](mailto:ebms_uat_receiver@ebmsgits.sbi.co.in)",  "mobileNo": "8073012970",  "merchPushResponseFlag": "N",  "merchPushResponseUrl": "<https://test.timesofmoney.com/direcpay/secure/NewFile.jsp>",  "businessName": "MahaOnlineLimited",  "brandName": "MahaOnlineLimited",  "mId": "1000003"  }  ],  "total": 1  } |

|  |  |
| --- | --- |
| Error Code | Error Description |
| Respective generic error code and description with field Name | |
| 1003 | Merchant Notification View  Is not found |

#### 

### Validation

* Raise custom exception if data is not available in cache and database.

### Business Logic

**Admin controller**

Process request to get merchant notification details.

**Admin service**

Query cache for merchant notification details. If not found query database.

Update cache if data found in database. And send a successful response with notification details.

## Merchant – Save Paymode Details

### API Implementation

**Endpoint:** /admin/v1/merchant/payMode

**Method:** POST

**Description: This API allows to create and save payment modes for an active merchant Id.** ~~Save paymode details in cache~~

**Request body:**

{

"srNo": string,

"mid": string,

"payGatewayName": string,

"payGatewayId": string,

"currency": string,

"payModeCode": string,

"payProc": string,

"gatewayIssueCode": string,

"aggregatorGatewayMapId": string,

"status": string,

"isTpActive": string,

"recordStatus": string,

"timerWindow": string,

"upiQrVpa": string

}

Sample request

|  |
| --- |
| {  "srNo": "0",  "mid": "1000003",  "payGatewayName": "Bharat Bank",  "payGatewayId": "134",  "currency": "INR",  "payModeCode": "CC",  "payProc": "VISA",  "gatewayIssueCode": "NA",  "aggregatorGatewayMapId": "21",  "status": "A",  "isTpActive": "N",  "recordStatus": "A",  "timerWindow": "1",  "upiQrVpa": ""  } |

Sample response

|  |
| --- |
| {  "status": 1,  "data": [  "Pay mode details saved in cache Successfully."  ]  } |

|  |  |  |  |
| --- | --- | --- | --- |
| **Paymode Request** | | | |
| Properties | Mandatory | Description | Data Type |
| srNo | No | Serial number | NUMBER(20) |
| mid | Yes | Merchant ID | VARCHAR2(20) |
| payGatewayName |  | Pay gateway name | VARCHAR2(4000) |
| paymentGatewayId | No | Gateway ID | VARCHAR2(30) |
| currency | No | CURRENCY\_CODE | VARCHAR2(10) |
| payModeCode | No | Pay mode code | VARCHAR2(10) |
| payProc | No | Pay proc Id | VARCHAR2(12) |
| gatewayIssueCode | No | Gateway Issue code | VARCHAR2(50) |
| aggregatorGatewayMapId | No | Aggregator gateway map id | VARCHAR2(10) |
| status | No | status | CHAR(1) |
| isTpActive | No | ISTPVACTIVE | VARCHAR2(50) |
| recordStatus | No | RECORDSTATUS | VARCHAR2(50) |
| timerWindow | No | Timer window | VARCHAR2(10) |
| upiQrVpa | No | UPI payment Type | VARCHAR2(100) |

**Path parameters:** None

**Response:**

**Success** – 200 Ok

{

"status": 1,

"data": [

"Pay mode details saved in cache Successfully."

],

“count”: 1,

“length”: 1

}

**Error** – 200 Ok

{

"errors": [

{

"errorCode":"ERROR\_CODE ",

"errorMessage":"ERROR\_MSG"

},

],

"status":"0"

}

### Validation - None

### Business Logic

**Admin controller**

Process request to save paymode.

**Merchant paymode service**

Save paymode details in cache.

# Key Management Service



## Generates API Key ID and Key Secret

### API Implementation

**Endpoint: /**kms/v1/key/generation/api

**Method:** POST

**Description: This API allows to save API Key ID and Keysecret, hashes the API Keysecret and displays the API Keys.**.

**Request Body:**

{

"mid":"1000003",

"keyLength":"AES\_256",

"newKeyExpiryTime":12,

“expiry”:0

}

**Response –** Success 200 Ok 

{

"data": [

{

"Merchant-API-Key- Secret": "WlaSKTYsY8WI7kjnkRQd7BQ/WQw+MyDCp2za+l5GcCs=",

"Merchant-API-Key-Id": "J0Pq09ibXXXUkL8ACZlV"

}

],

"status": 1,

"count": 1

}

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Description** | **Mandatory** | **Data Type** |
| mId | merchant ID for the request. | Yes | String |
| keyLength | This is for encryption | No | Enum  AES\_128(128), AES\_192(192), AES\_256(256), AES\_512(512); |
| expiry | Expiry time in hours | No | Int |
| newKeyExpiryTime | NEW\_KEY\_EXPIRY\_TIME\_AT\_LEAST\_1 | No | int |

### Validation

* Validate the mId that should not empty and newKeyExpiryTime value min at least 1.

### Business Logic

**Step-1: KeyManagement controller**

KeyManagement controller the request containing necessary details for key generation and Service is called to further process request

**Step-2: KeyManagement Service**

This service generates apiKeyId , apiKeySecret and apiKeySecretHash and called APIKeyManagementDao for save ApiKeys into DB.

Send success response with Merchant-API-Key-Id and Merchant-API-Key-Secret.

### Information

* Log generated apiKeyId,apiKeySecret, apiKeySecrethash.
* Also log merchant Id and calculated expiryTime.

## Generates Encryption Keys for a given Merchant

### API Implementation

**Endpoint:** /kms/v1/key/generation/encryption

**Method:** POST

**Description: This API helps to generate API Keys for specific merchant id.**

**Request:**

{

"mid":"1000834",

"keyLength":"AES\_256",

"expiry":0,

"newKeyExpiryTime":12

}

**Response:** Success 200 Ok

{

"data": [

"7KgEO8kza3fu3WQKhoOcSdzVovXlH49mVRvUTiViqGc="

],

"status": 1

}

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Description** | **Mandatory** | **Data Type** |
| mId | merchant ID for the request. | Yes | String |
| keyLength | This is for encryption | No | Enum  AES\_128(128), AES\_192(192), AES\_256(256), AES\_512(512); |
| expiry | Expiry time in hours | No | Int |
| newKeyExpiryTime | NEW\_KEY\_EXPIRY\_TIME\_AT\_LEAST\_1 | No | int |

### Validation

Validate the mId that should not be empty and newKeyExpiryTime value minimum at least 1.

### Business Logic

**Step-1: KeyManagement controller**

Controller contain the details of the merchant for encryption key generation and keyManagementService is called to further process request .

**Step-2: KeyManagement Service**

This service is used to generate encrypted MEK key and calling EncryptionKeyManagementDao for saveEncryptionKeys Into DB.

Send Success response with mek key.

**Step-3: EncryptionKeyManagementDao**

EncryptionKeyManagementDao saves encryption keys (MEK and KEK) into the database. It generate a KEK encrypt it and then encrypt MEK using the KEK.

### Information

* Log generated mek key, calculated expiryTime. calculated newKeyExpiryTime, calculated keyExpiryTime, generated kek.
* Also log encrypted kek,mek,and new key status.

## Generates Encryption Keys for a given Merchant

### API Implementation

**Endpoint:** /kms/v1**/**key/validated/api

**Method:** POST

**Description: This API confirms the API Key for a given mechant id.**

**Request Body:**

{

"apiKey":"xPWOIOQ9E4L1CJEbAwUe",

"apiKeySecret":"LgKrrAUCSwSKBKIZIQT8CshCQZ7C2WNdgOnQwAexepA="

}

**Response:** Success 200 Ok

{

"data": [

"1000003"

],

"status": 1

}

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Description** | **Mandatory** | **Data Type** |
| apiKey | API Key | Yes | String |
| apiKeySecret | API KeySecret | Yes | String |

### Validation

* apiKey and apiKeySecret is mandatory and it should not be blank

### Business logic

**Step-1: KeyManagement controller**

Requestbody contain apiKey and apiKeySecret and it should not be blank and calling keyManagementService for validate the keys.

**Step-2: KeyManagement Service**

Service layer is used for return merchant id based on apiKey and apiKeySecret.

Get merchant id by apiKey and secrethash.

Send success response with merchant id.

### Information

* Log merchant id, api key.

## Get Encryption Keys for a given Merchant ID

### API Implementation

**Endpoint:** /kms/v1/key/encryption/ {mId}

**Method:** GET

**Description:** This API displays the encryption keys for a given merchant ID.

**Request:**

**Path Parameter:** The Merchant ID, used to retrieve the encryption key for a given merchant id.

**Response:** Success 200 Ok

{

"data": [

{

"mek": "AAAAAAAAAAAAAAAAAAAAABBc+QGfr7/d1iBPSMQ9hhy42b6911b218DSrCSRIb7m/FmjkbH1pN96UU/RUjeCUiB5DYHlrQPStddLvw==",

"kek": "AAAAAAAAAAAAAAAAAAAAABsUbUmttEY4jDZEalPSzZ+8FN/a0TwiUoWnTQhgiqLMSfOWZFXf/DN/eKA3nG5b4SntjJqJUUndNc0roA=="

}

],

"status": 1

}

### Validation: No

### Business Logic

**Step-1: KeyManagement controller**

The Merchant ID for which encryption keys need to be fetched and calling keyManagementService for get encryption keys.

**Step-2: KeyManagement Service**

Calling EncryptionKeyManagementDao for get keys based on merchant id.

Send success response with keys.

**Step-3: EncryptionKeyManagementDao**

Fetch MEK based on mId,keyType and status and .

Fetch KEK based on mId.

### Information

* Log merchant id, encryption key response.

## Download Encryption Keys

### API Implementation

**Endpoint:** /kms/v1/key/download/encryption/ {mId}

**Method: GET**

**Description:** This API allows downloading the decrypted encryption keys for a given Merchant id.

**Request:**

**Path Parameter:** The Merchant ID used for which the encryption keys are to be fetched for download.

**Response:** Success 200 Ok

{

"data": [

"7KgEO8kza3fu3WQKhoOcSdzVovXlH49mVRvUTiViqGc="

],

"status": 1,

"count": 1

}

### Validation: No

### Business Logic

**Step-1: KeyManagement controller**

The merchant id is used for which the encryption keys are to be fetched for download.

Calling keyManagementService for get merchant encryption keys for download.

**Step-2: KeyManagement Service**

Merchant Service layer is used for to decrypt encryption key for a given merchant id to be used for download

Calling EncryptionKeyManagementDao for decrypt encryption key.

Send Success response with decrypted keys.

**Step-3: EncryptionKeyManagementDao**

This dao layer fetched MEK and KEK from DB.

Decrypt KEK using AEK. Decrypt MEK using KEK and .

Return decrypted MEK.

### Information

* Log merchant id
* Log Fetched MEK and KEK from DB.
* Decryption of KEK completed.
* Decryption of MEK completed

## Fetches list of Keys

* Fetches list of keys associated with a given merchant id and key type.

### API implementation

**Endpoint:** /kms/v1/key/{keytype}/{mId}

**Method: GET**

**Description:This API display** list of keys including API, MEK, KEK which are associated with a given merchant id..

**Request:**

**Path Parameter:** Merchant ID for which the keys need to be fetched and KeyType(API,KEK,MEK) for type of keys to fetch and pageable for pagination details for fetching list of keys.

**Response:** Success 200 Ok

{

"data": [

{

"merchantId": "1000834",

"keyId": "vEbLBxTKzCmINmaEAEvS",

"expiryTime": 1772707415392,

"status": "ACTIVE",

"createdAt": 1741171415500

}

],

"status": 1,

"count": 1,

"total": 1

}

### Validation: No

### Business Logic

**Step-1: KeyManagement controller**

Fetching list of keys for merchant with id and key type (API,KEK,MEK) and calling keyManagementService for get list of keys.

**Step-2: KeyManagement Service**

This service is used for fetched the list of keys based on merchant id and key type (API,KEK,MEK) by calling apiKeyManagementDao.

Send success response with list of keys.

### Information

* Log merchant id and type

# Bank Service



## List active banks

### API Implementation

**Endpoint:** /admin/v1/bank

**Method:** POST

**Description: This API allows you to view a list of all active banks.** ~~API provides list of all active banks.~~

**Request body:** None

**Path parameters:** None

**Response –** Success 200 Ok

{

"status": 1,

"data": [

{

"bankId": string,

"bankname": string,

"bankAbbreviation": string

}

],

"count": 1,

"length": 1

}

Sample response

|  |
| --- |
| {  "status": 1,  "data": [  {  "bankId": "184",  "bankName": "THE SURAT DISTRICT COOPERATIVE BANK LIMITED",  "bankAbbreviation": "TSDCOBL"  },  {  "bankId": "180",  "bankName": "NKGSB COOPERATIVE BANK LIMITED",  "bankAbbreviation": "NCBL"  },  {  "bankId": "130",  "bankName": "THE SARASWAT CO-OPERATIVE BANK LTD",  "bankAbbreviation": "TSCBL"  },  {  "bankId": "33",  "bankName": "CREDIT AGRICOLE CORP N INVSMNT BANK",  "bankAbbreviation": "CACNIB"  },  {  "bankId": "47",  "bankName": "IDBI BANK LTD",  "bankAbbreviation": "IDBIB"  },  {  "bankId": "119",  "bankName": "THE KARAD URBAN CO-OP BANK LTD",  "bankAbbreviation": "TKUCBL"  },  {  "bankId": "136",  "bankName": "THE TAMILNADU STATE APEX COOPERATIVE BANK LIMITED",  "bankAbbreviation": "TTSACBL"  },  {  "bankId": "138",  "bankName": "THE THANE JANATA SAHAKARI BANK LTD",  "bankAbbreviation": "TTJSBL"  },  } |

**Response:** Error 200 Ok

{

"errors": [

{

"errorCode":"ERROR\_CODE ",

"errorMessage":"ERROR\_MSG"

},

],

"status":"0"

}

### Validation - None

### Business Logic

**Step-1: Bank details controller**

API processes requests to fetch list of active banks.

**Step-2: Bank details service**

Query cache for active bank listing. If empty query database.

Raise custom exception if no data available.

Process list of active banks, prepare success response with data.

## Get all Branches

### API Implementation

**Endpoint:** /admin/v1/bank/branch/{bankId}

**Method:** GET

**Description: This API provides a** **list of branches for a given bank**.

**Request body:** None

**Path parameters:**

**bankId –** Bank identifier

**Response –** Success 200 Ok

{

"status": 1,

"data": [

{

"ifscCode": string,

"branchName": string

}

],

"count": 1,

"length": 1

}

**Response:** Error 200 Ok

{

"errors": [

"errorCode":"ERROR\_CODE ",

"errorMessage":"ERROR\_MSG"

},

],

"status":"0"

}

Sample response

|  |
| --- |
| {  "status": 1,  "data": [  {  "branchName": "BANKAPASI",  "ifscCode": "ORBC0100481"  },  {  "branchName": "HALDI",  "ifscCode": "ORBC0100482"  } |

### Business Logic

**Step-1: Admin controller**

API Service is called to get all branches of bank.

**Step-2: Bank details service**

Query cache for a provided bank identifier. If not available query database.

If not found in both cache and database raise custom exception.

If found in database update cache with record retrieved from database.

Send success response with merchant id and status of merchant.

## Get bank details

### API Implementation

**Endpoint:** /admin/v1/bank/validate

**Method:** POST

**Description: This API helps the user to view bank details as per the IFSC code.** ~~API provides bank details~~.

**Request body:**

{

“ifscCode”: string,

“bankName”: string,

“branchName”: string

}

**Path parameters:** None

**Response –** Success 200 Ok

{

"status": 1,

"data": [

{

"bankId": string,

"bankname": string,

"branchName": string

"branchAddress": string

"city": string

"state": string

}

],

"count": 1,

"length": 1

}

Sample request

|  |
| --- |
| {  "ifscCode":"NKGS0000041",  "bankName":"NKGSB COOPERATIVE BANK LIMITED",  "branchName":"NASHIK"  } |

Sample response

|  |
| --- |
| {  "status": 1,  "data": [  "IFSC code validated Successfully"  ]  } |

**Response:** Error 200 Ok

{

"errors": [

{

"errorCode":"ERROR\_CODE ",

"errorMessage":"ERROR\_MSG"

},

],

"status":"0"

}

### Validation

* Validate ifsc code. If not found, raise custom exception.

### Business Logic

**Step-1: Bank details controller**

API provides bank details information

**Step-2: Bank details service**

Query database to fetch Bank details for ifsc code.

Send a successful response with bank details.

## Validate IFSC

### API Implementation

**Endpoint:** /admin/v1/bank/{ifscCode}

**Method:** POST

**Description:** This API allows user to validate bank details with the IFSC code, bank name and branch name **.**

**Request body:**

{

“ifscCode”: string,

“bankName”: string,

“branchName”: string

}

**Path parameters:** None

**Response –** Success 200 Ok

**Response:** Error 200 Ok

{

"errors": [

{

"errorCode":"ERROR\_CODE ",

"errorMessage":"ERROR\_MSG"

},

],

"status":"0"

}

Sample request

|  |
| --- |
| {{Dev}}/admin/v1/bank/NKGS0000041 |

Sample response

|  |
| --- |
| {  "status": 1,  "data": [  {  "bankId": "180",  "bankName": "NKGSB COOPERATIVE BANK LIMITED",  "branchName": "NASHIK",  "branchAddress": "BEMUTHA S NAVKAR HEIGHTS,SHARANPUR ROAD,NASHIK- 422 002",  "city": "NASHIK",  "state": "NASHIK"  }  ]  } |

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Description** | **Mandatory** | **Data Type** |
| ifscCode | IFSC code | Yes | String |
| bankName | Bank Name | Yes | String |
| branchName | Branch Name | Yes | String |

### Validation

* Validate ifsc code, bank name and branch name, raise custom for exception condition.

### Business Logic

**Step-1: Admin controller**

API to validate bank details. Service is called to further process request

**Step-2: Bank details service**

Query cache for a provided ifsc code, branch code identifier. If not available query database.

If not found in both cache and database raise custom exception.

If found update cache with record retrieved from database.

Send success response with merchant id and status of merchant.