Identity Auth REST Service

# Background

Identity Auth REST service can be used to authenticate an Individual. MOSIP supports below types of authentications –

* Pin based – OTP, Static Pin
* Demographic based – Name, Address, DOB, Gender, Phone and EmailID
* Biometric based – Fingerprint, IRIS and Face

## Target users -

TSP can use Identity Auth service to authenticate an Individual by using one or more of the above types of authentication supported.

## Key requirements -

* TSP can authenticate an Individual using one or more of the above mentioned authentication types
* TSP will send Individual’s UIN/VID to enable authentication of Individual
* TSP will send muaCode and msaCode to authenticate and authorize a TSP to authenticate an Individual
* Check Individual’s UIN/VID for authenticity and validity
* Inform authentication status (success/failure) to the Individual in the form of message and/or email

## Key non-functional requirements –

* Log each stage of authentication process:
  + Log all the exceptions along with error code and short error message
  + As a security measure, Individual’s UIN or PI/PA should not be logged
* Audit :
  + Audit all transaction details during authentication process in database
  + Individual’s UIN or PI/PA details should not be audited
  + Audit any invalid UIN or VID incidents
* Exception :
  + Any failure in authentication/authorization of TSP and validation of UIN and VID needs to be handled with appropriate error code and message in Auth Response
  + Any error in Individual authentication also should be handled with appropriate error code and message in Auth Response
* Security

<<TBD>>

# Solution

Identity Auth REST service addresses the above requirements as explained below.

1. TSP to construct a **POST** request with below details and send to Request URL **identity/auth**

Sample Request Body –

{

"id" : "mosip.identity.auth",

"ver" : "1.0",

"indId" : "1234567890",

"indIdType" : "V",

"authType" :

{

"address" : false,

"fullAddress" : true,

"personalIdentity" : false,

"otp" : false,

"pin" : false,

"bio" : false

},

"muaCode" : "tspLevel1ID",

"txnID" : "txn12345",

"reqTime" : "2018-10-17T07:22:57.086+0000",

"reqHmac" : "<SHA of request element>",

"key" : "<encrypted\_encoded\_session\_key>",

"matchInfo" :

[

{

"authType" : "fullAddress",

"ms" : "P",

"mt" : 60

}

],

"pinInfo" :

{

"value" : "123456",

"type" : "OTP"

},

"request" :

{

//JSON request as per the id object schema defined by the country

"identity": {

"name": [

{

"language": "ar",

"value": "ابراهيم"

},

{

"language": "fr",

"value": "Ibrahim"

}

],

"addressLine1": [

{

"language": "ar",

"value": "عنوان العينة سطر 1"

},

{

"language": "fr",

"value": "exemple d'adresse ligne 1"

}

],

"fullAddress": [

{

"language": "ar",

"value": "فاس-الدار البيضاء"

},

{

"language": "fr",

"value": "Casablanca"

}

],

"leftEye": [

{

"value": "encoded\_left\_eye\_image"

}

],

"rightIndex": [

{

"value": "encoded\_right\_index\_image"

}

]

}

}

}

1. Authenticate and Authorize TSP <<TBD>>
2. Validate “reqTime” for incoming Identity Auth Requests for valid format and timestamp < 24 hours (configurable value) from current time
3. Integrate with kernel UIN Validator and VID Validator to check UIN/VID for validity. Validate UIN/VID for authenticity in AuthDB
4. Once the above validations are successful, Identity Auth request is then validated based on specific authentication types as described below.
   1. OTP Auth – OTP value sent to the Individuals are validated
   2. Demo Auth – Input Demo fields are validated against stored Individual’s Demo Fields
   3. Pin Auth – Static Pin generated by the Individuals is validated against the input pin
   4. Bio Auth – Fingerprint/IRIS/Face minutiae stored is validated against input bio minutiae
5. Retrieve mode of communication with Individual using admin config to send authentication success/failure information
6. When the Individual is successfully authenticated based on one or more of the above authentication types, a sms/email notification is sent to them using Kernel’s SmsNotifier and EmailNotifier to their stored phone/email respectively.
7. Respond to TSP with below success Identity Auth response -

{

"status" : true,

"err": [],

"txnID": "txn12345",

"resTime": "2018-10-17T13:40:19.590Z",

"info":

{

"indIdType": "V",

"reqTime": "2018-10-17T07:22:57.086+0000",

"ver": "1.0",

"matchInfo":

[

{

"authType": "fullAddress",

"ms": "P",

"mt": 60

}

],

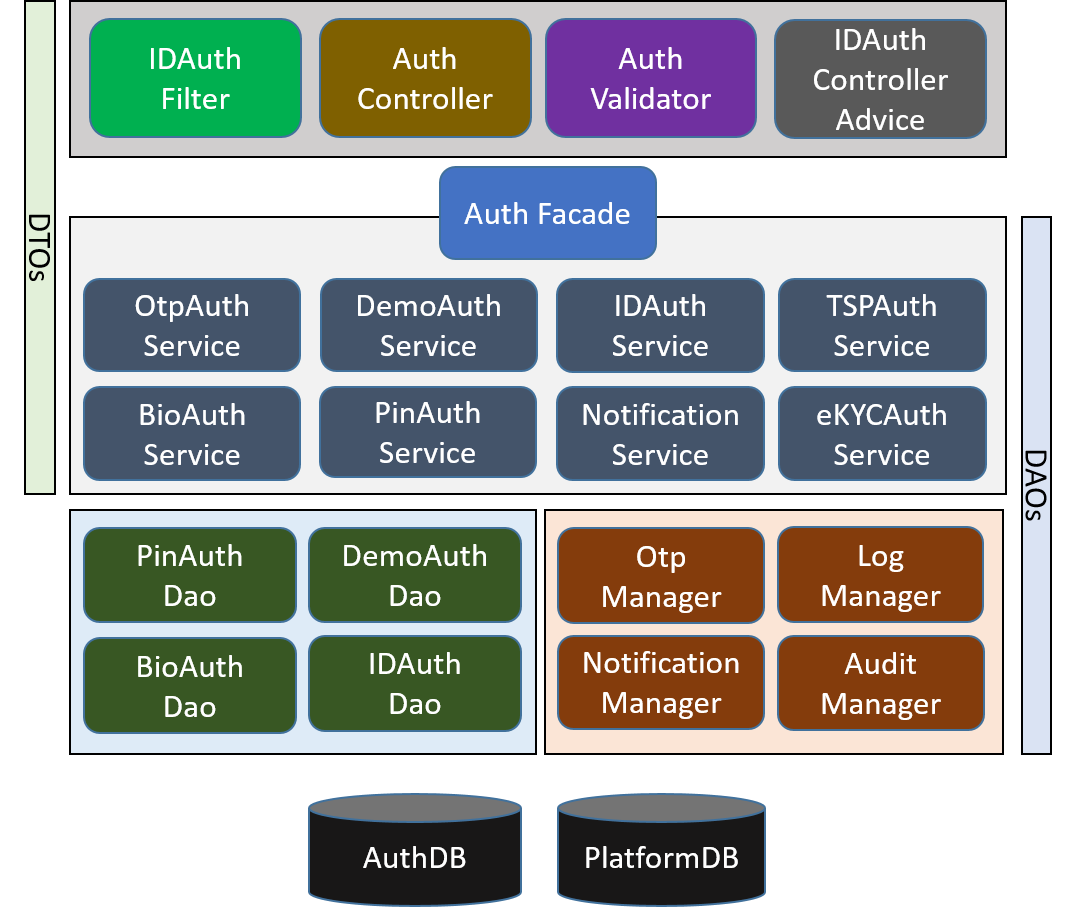
"usageData": "0xaf100000af100000"

}

}

## Component Diagram

Below component diagram shows all the components that work together during authentication process.

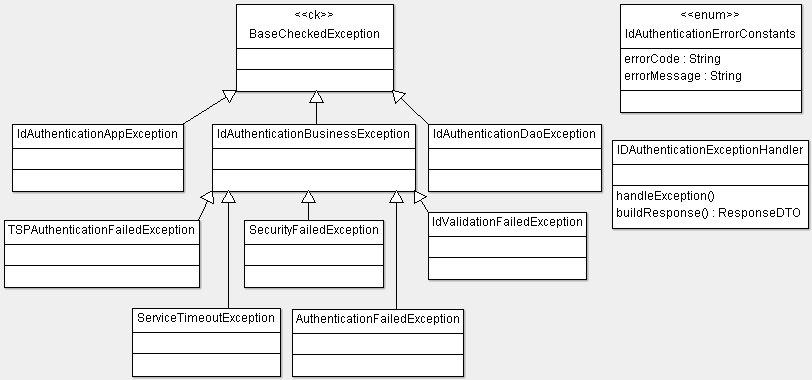


Below are the details on each of the above components:

* **IDAuthFilter** – A Spring Filter to intercept AuthRequest and perform TSP authentication/authorization
* **AuthController** – A Spring Controller to process all the types of authentications
* **AuthValidator** – A Spring Validator to perform basic validation on AuthRequest
* **IDAuthControllerAdvice** – A Spring Controller Advice to handle all IDA exceptions centrally and respond to TSP in the form of AuthResponse
* **AuthFacade** – A facade layer decide which authentication service to be invoked based on the auth request
* **OTPAuthService** – A Spring Service that contains business logic for authenticating an Individual using otp
* **DemoAuthService** – A Spring Service that contains business logic for authenticating an Individual using one or more input demo fields
* **BioAuthService** – A Spring Service that contains business logic for authenticating an Individual using one of the supported Bio Auth types like Fingerprint, IRIS and Face
* **NotificationService** – A Spring Service to determine what is the notification type associated with the Individual and then send the authentication success/failure notification
* **IDMasterAuthService** – A Spring Service that validates UIN and it matching VID
* **OTPManager** – A Manager class which integrates with Core-Kernel validator logic to validate input otp value

## Exception Handling:

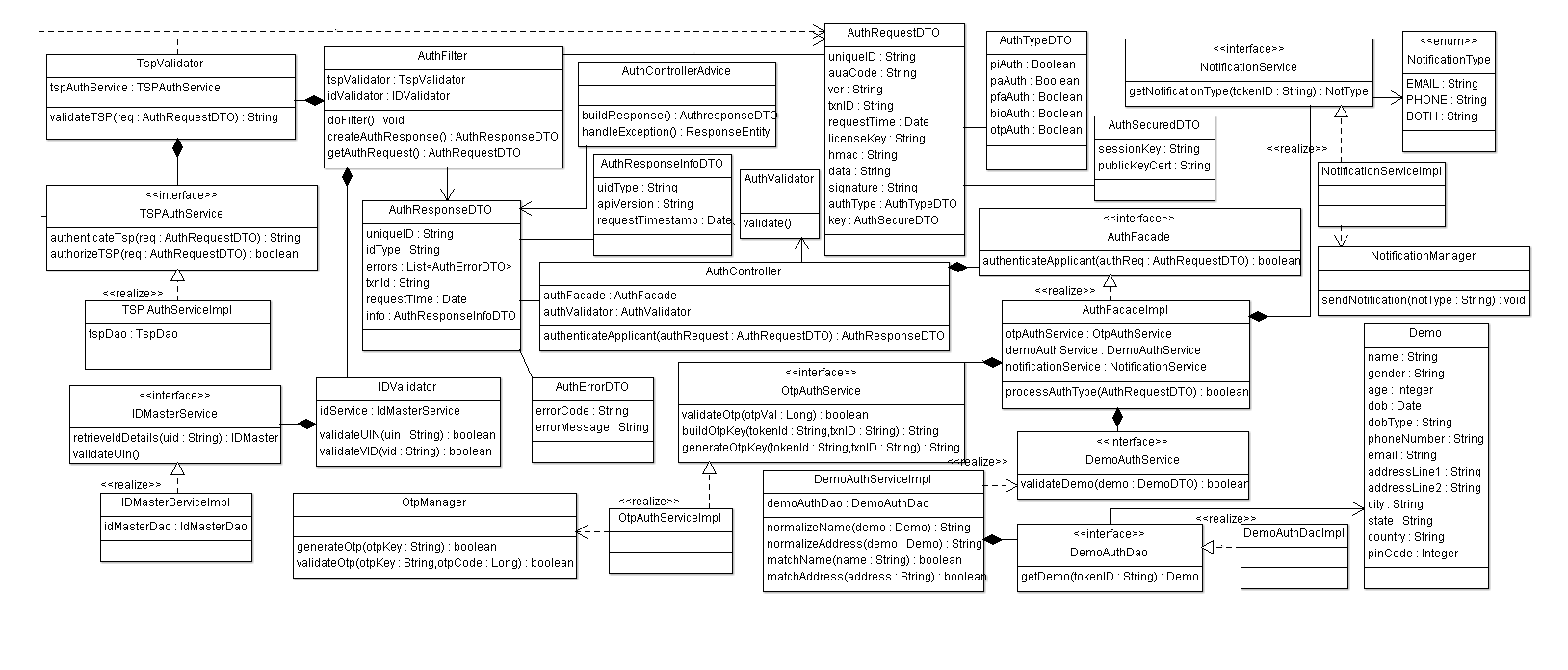
Below exception hierarchy shows how IDAuthentication will handle exceptions.



* **IDAuthDaoException** – This exception will wrap all database related exceptions thrown by Dao layer
* **IDAuthBusinessException** – This exception handles all business logic related issues, any business condition not met, issues in communicating with external APIs, etc. It also wraps all IDAuthenticationDaoException
* **IDAuthAppException** – This exception class will be thrown by Controller layer and wraps all IDAuthenticationBusinessException and throws only one type of exception to the end user
* **IDAuthExceptionHandler** – This is a generic exception handler class which centrally handles exceptions thrown by controller layer and creates a generic response object, which is then sent as the response to the user
* **IDAuthErrorConstants** – This is a common error constants class which contains the list of all error codes and error messages used by IDA product.

## Class Diagram:

The below class diagram shows relationship between all the classes which are required for common authentication service.

****

## Sequence Diagram: