IDM - Identity Module

System Documentation

Overview

IDM is an authentication and authorization module built on top of AWS Cognito and served as a service through API interface. It enables users to securely authenticate with multiple applications and websites by logging in only once, with just one set of credentials. Currently, each website or application maintains its own database of users that were registered on that application.

Users have one set of credentials and can have metadata shared with other applications or not through IDM User Partitions feature. IDM major features are: Registration, Authentication, Authorization and User Partitions. User information is captured in a single database, using a single set of credentials but shared among various BCCSA resources.

Application Features

Applications

Applications are the main entity on IDM system. Every application has Application Tokens and Roles; User are associated with an application role and Partitions ACLs are also associated with application roles.

Application Tokens

For each application, it should have one or more tokens related to it. This application tokens are used to calculate a AppID token used to identify the application on every request and make sure that application is accessing what it was supposed to.

Tokens can be enabled or disabled granularly, so if a code breach happens, that specific token be disabled or if abusive usage is detected in can also be throttled. For this reason, every application client should use its own Application Token, related with the same application. Example: Mobile application uses one, Web application uses other and API uses a third one.

Application Roles

Every application can have one or more roles. Roles are directly associated with one and just one application and can't be moved from application to application, they need to be deleted and created on the destination app.

A role dictate what that role can make within that application.

Role options:

- Registration enabled: This option enables self-registering of users on that role. If not enabled, only System administrators will be allowed to link users into that role.
- MFA Required: Currently not implemented. This will force all users from that role to use MFA.
- Super role: When enabled the users that are part of this role will be able to access all users from the application this role is part of. Read or Write access depends on the option below. If not enabled, this will make the default behaviour and users will only be able to access their own information.
    - Read Only: This option is only considered when Super Role is enabled. If enabled, it will allow users from this role to read all information from users of that application but not write.
- Allow federation: Currently not implemented. This will allow users to login with other identity providers such as Facebook and Google.

Note:

- Changes on those preferences in the UI or via API, makes immediate effect on registration calls.
- Sometimes roles are referred here on this document as claims.

Partition ACLs

Partitions ACLs are the relation between a partition namespace, an application role and the access level (Read or Read/Write). That said, we create partition ACLs to grant authorization for a specific role into a namespace and what that role can do on that namespace.

Namespaces are never pre-created to be used by ACLs, however the opposite is true, ACLs grant authorization to namespaces that will be created for each user on demand. So if we want to make usage of partitions system on IDM, we need to create an ACL for that namespace and role and then read/write it for each user.

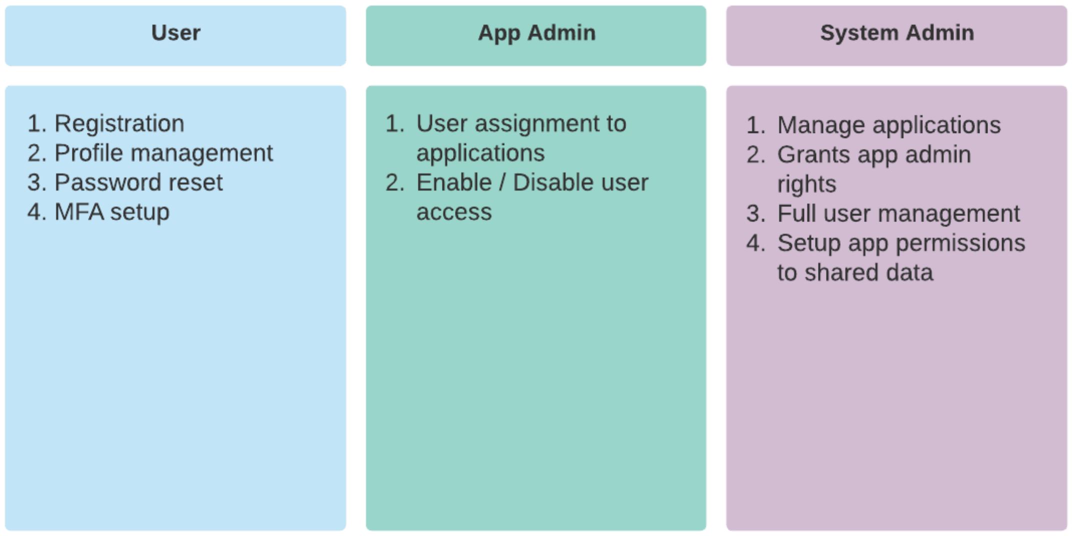
Role property, super role, is directly related with partitions and only users with Super Role enabled on their role will be able to make the access specified on the ACL for all users on their application. If not enabled, an user will be able to make the access specified on the ACL only for its own information.

Management Features

IDM UI

IDM UI itself is an IDM registered application that makes usage of the same API calls available for all applications. It allows user, application and system management through its UI and also makes popup based authentication for other applications, not discussed here in this document for the reason of being deprecated and highly discouraged for compatibility purposes.

IDM Roles



User	App Admin	System Admin
1. Registration	1. User assignment to applications	1. Manage applications
2. Profile management	2. Enable / Disable user access	2. Grants app admin rights
3. Password reset		3. Full user management
4. MFA setup		4. Setup app permissions to shared data

There are 3 main distinct roles on IDM that are fundamental for the understanding of the system.

- **System Admin**: Users responsible for managing the IDM system. Allowed to create/delete applications and its roles & tokens, manage partitions ACLs, create/delete/list all users and its roles in all applications.
- **App Admin**: Users responsible and authorized to manage users associated with that specific application, also able to create users with the role from application they administrate.
- **Users**: Normal users are allowed to self-register, manage their own personal information and access applications they are part of.

Important Note: The access of the above roles on the consumer applications are specific to every application implementation, for example, if the IDM User Admin role is allowed on the application claim conditional this user will be able to access it and make the actions implemented on the API, however this is **extremely discouraged** and consumers applications should allow only application specific roles to access and make API actions.

IDM API

IDM API is the only interface allowed to communicate with Cognito and the database containing the user information. Therefore, all Authentication and Registration calls must be done directly with IDM via public API.

Documentation can be seen and tested **here**

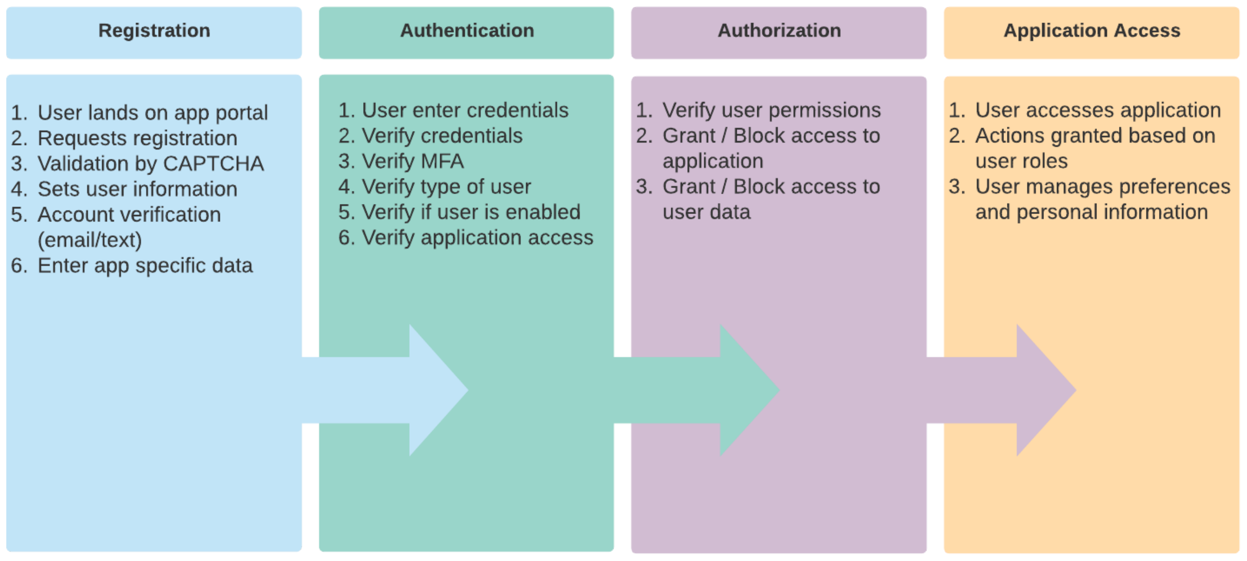
Notes:

- All API calls are done through secure protocol (HTTP/SSL) with valid certificate and data sent on plain-text, no base64 encoding.
- API communication format is `application/json` and do not accept any other at this moment. Therefore, content-type and accept headers are not required but will be allowed if specified.
- Every API call must be identified with what application is making that call. This can be done by setting the `AppAuth` parameters on the query URL or header with the AppID token calculated by the Application access token and Application access secret. Please, see more about AppID on its section at Application Features -> Application Tokens
- Most of the API calls require Authorization header to be set. Please, check API documentation to better understand what calls will need it. If not provided, 401 status code will be returned.

- API errors are based on body code and not status code, so every error will return 4xx code and the body will contain further information. Please, check the Error model to understand error structure.
- 5xx status codes are not expected and should be reported when happened.
- Bugs and support with API calls **can't** be done **without** transaction ID, returned on every API call response. This is crucial for our team to identify the transaction and what happened. So please, make sure to include it when reporting a bug or asking for support.
- Headers & Query parameter keys are case insensitive.

User Features

Main user features are briefly illustrated on the flow below.

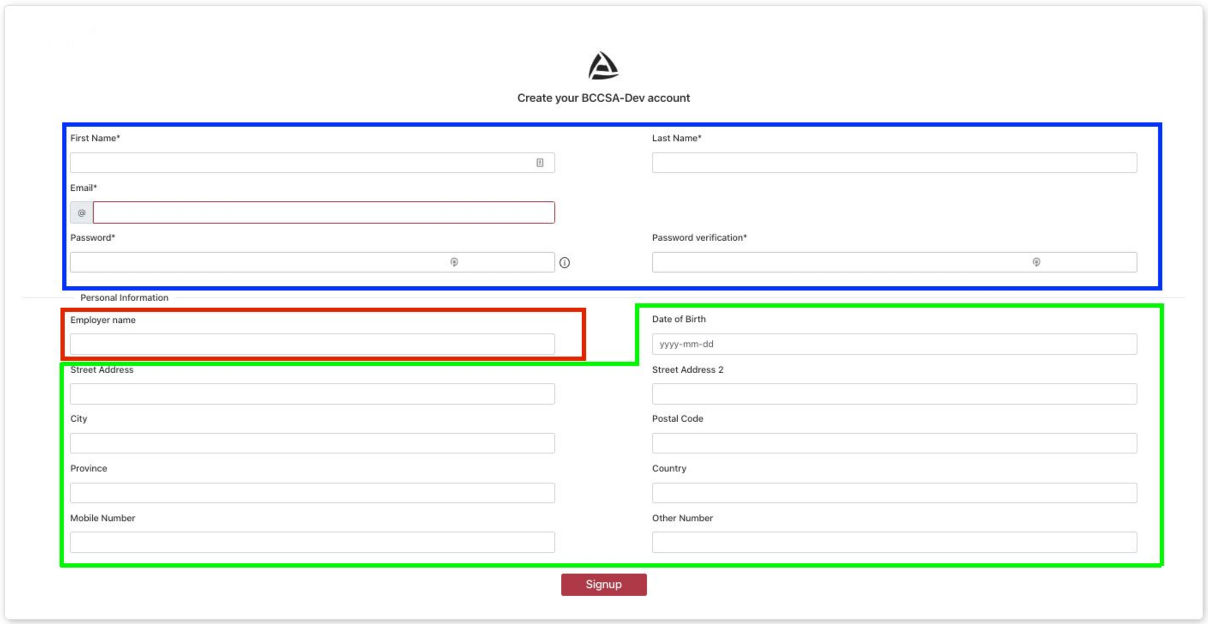

Registration
1. User lands on app portal
2. Requests registration
3. Validation by CAPTCHA
4. Sets user information
5. Account verification (email/text)
6. Enter app specific data
Authentication
1. User enter credentials
2. Verify credentials
3. Verify MFA
4. Verify type of user
5. Verify if user is enabled
6. Verify application access
Authorization
1. Verify user permissions
2. Grant / Block access to application
3. Grant / Block access to user data
Application Access
1. User accesses application
2. Actions granted based on user roles
3. User manages preferences and personal information

Registration

Currently, registration is solely done on IDM UI but also exposed by the API for applications that wish to make its own registration process. This process rather that just create a user entity and confirming it, can also attach users to existing application roles.

The registration involves gathering user information, sending confirmation email to validate the user email (done by IDM), attaching user into specified roles (Role with registration enabled only), confirming it through IDM UI or API call and then entering on the application for the first time to agree with that application terms or whatever is required at first user launch.

The existing process implemented on IDM UI, involves collecting user basic and business informations required by BCCSA applications and stored into user partitions. If registration call succeeds, Registration token is returned by the API and when merged with the second registration code send on the user email, can be used to confirm the account via API or IDM UI.



Create your BCCSA-Dev account

First Name*	Last Name*
Email*	
Password*	Password verification*

Personal Information

Employer name	Date of Birth (yyyy-mm-dd)
Street Address	Street Address 2
City	Postal Code
Province	Country
Mobile Number	Other Number

Signup

The screenshot above shows the information collected by IDM UI sectioned by where it will be saved. On the blue box we have the basic user information stored at user level and accessible by any application after the successful authentication. On the red box, we have professional information, stored on the user professional partition and accessible for all applications that have at least READ access into this partition. And last but not least on the green box we have the user personal information, also stored on the user personal partition and accessible for applications that have at least READ access on this partition.

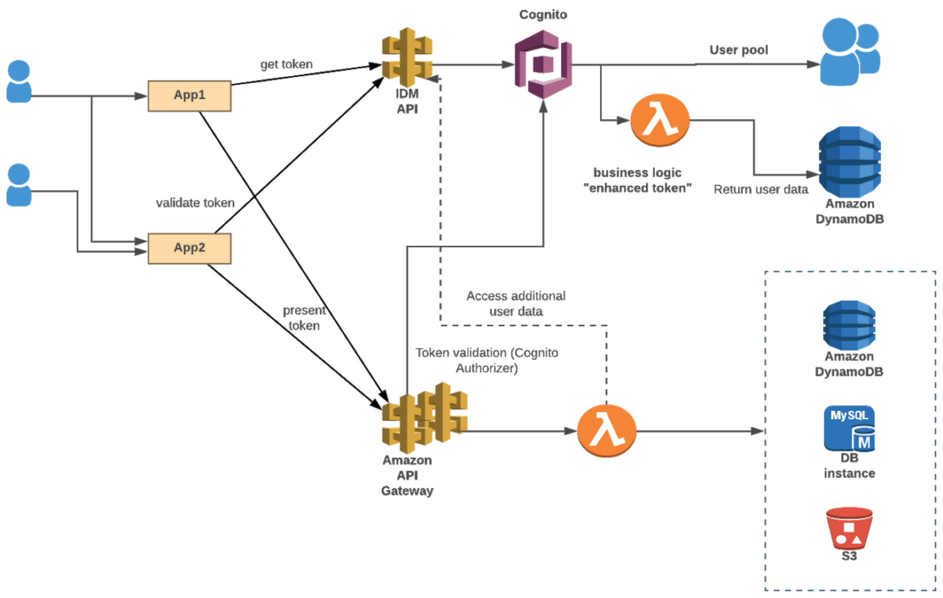
Dummy account registration is encouraged to be done to better understand the registration process.

Note: Registration confirmation email sends IDM UI link on it and normally is opened on a new tab when directly clicked. Therefore, the consumer application is not notified when the user registration is confirmed and should present a login page until the user is successfully authenticated to prevent errors on this process.

Authentication

Below is a flow describing 2 applications that are under the same apex domain and therefore share the authentication token. When **App1** authenticate with IDM API it saves the token and other relevant information on the local cookie under `*.mydomain.com` to be later used with any other application that is server through any subdomain of the apex. When authorization is completed with success, IDM also returns the basic user information and all partitions that the caller application has access within the authorization token, so everything is done in one, compact call.

On this case, **App2** is launched after successfully authentication on **App1**, so it checks for the authorization token, from now on called IDM Token, is set, if so, it optionally call IDM to validate that token or just opt to use it for authorizing the subsequent API calls on the application API. **App2** Will make use of the user information set by the **App1** and will not need any additional API calls to IDM.



Authorization

Following the authentication flow above, we are authenticated and now will make authorized API calls to the application API or IDM API with the IDM Token retrieved on the authentication call set as the `Authorization` header in the JWT format, `Authorization: Bearer <IDM Token>`.

When **App2** API is called, the authorization header is received by the API, that if using API Gateway will take advantage of API Gateway Cognito Authorizer to validate the token with Cognito User Pool and decode it, launching the Lambda function with an already validated and decoded token, so the Lambda will just check the claims available on the token, to see if any of then are allowed to make the specified action on the API. If not API Gateway is used, the API will need to make the heavy lift of receiving the IDM Token, decoding it, validating it via Cognito User Pool public JWKS (available at `https://cognito-idp.REGION.amazonaws.com/USER_POOL_ID/.well-known/jwks.json`), checking if is not expired, and then authorizing the action with the available claims on the decoded token. Attention, JWKS is a very important step to avoid accepting IDM or Generic JWT tokens issued by other User Pool or attackers.

Please, refer to JWT documentation for further information.

User Partitions

User partitions are a crucial part of the user information and can be simplified as information buckets identifier by a namespace and a user. It gives the flexibility of saving user information that may be useful for other applications on a fine-grained controlled namespace but also considered part of the user data. These partitions can be later retrieved by other applications that have access to it or rewritten by other applications that have RW access on that specific namespace.

Partitions have an encouraged limit of 200Kb but are hardly limited to 390Kb on the average and will not be allowed to be saved if this limit exceeds.

User Reflection

User reflection is an important IDM feature that guarantees that user information is synchronized between apps even when the user updates his information in one app but also exists on another app.

Whenever a user is updated via IDM API call, IDM will publish an AWS SNS message into a protected SNS Topic, so applications can subscribe to this topic to make sure its databases are up to date with the latest basic user information.

Important Note: If SNS topic wasn't provided, please, request SNS User Reflection topic for your system administrator

ES6/NodeJS Module

Public registered on the official npm registry, `@ikonintegration/idmclient` is a ES6 nodejs module compatible with browser environment that abstracts most of the core logic describe here on this document and also shortcut API calls and required parameters to consume IDM without much effort. It's strongly recommended that this module is used on the integration or at least the developer uses it as a base to make its own implementation.

Best security practices

- As stated before, API does not accept plan HTTP calls, therefore all HTTP+SSL call should check against the API provided certificate and its authority. No self-signed certificates will be provided.
- It's encouraged that Application Access token and Application Access secret be stored in an obfuscated form on the front end and not visible by the client.
- Production Application tokens should never be stored on the application repository. Refer to AWS Secrets and inject them at build time for safer process.
- IDM Token should be stored on browser cookies that are restricted to known domains.
- Never store IDM Token at server sessions or at the same location Application tokens reside. This can lead to attacks where the attacker behaves like the registered application and can make actions on the user's behalf.
- Never load external resources from unknown resources, XSS vulnerability.
- User password should never be saved or stored, it should be transferred to API call and then cleared from memory.

Tokens

IDM Token (JWT)

IDM Token, here referred as IDM Token, JWT and authentication token, is the token returned by IDM API after successful authentication and then served as the authorization hash used on the application and

IDM API calls. This token can be decoded for retrieving very basic user information and its claims (roles) of all applications that user has.

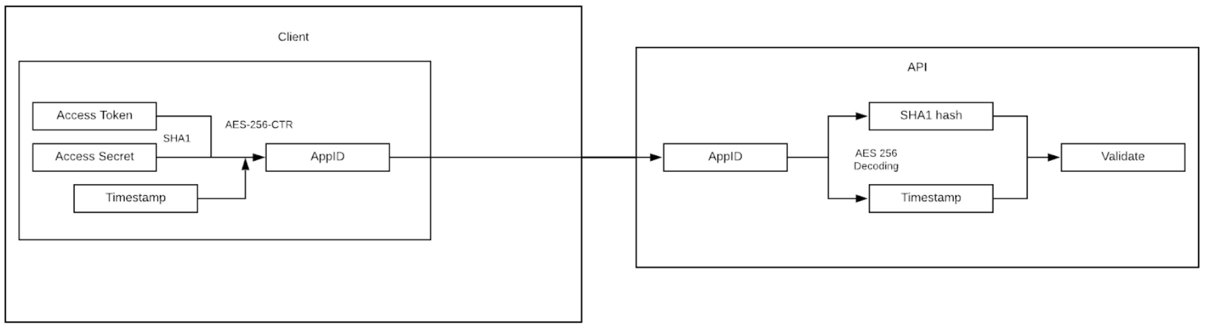
A token has a default validity of one hour. And when expired, renewal token, returned by the API authorization call should be used within validation API call to retrieve a new valid token.

Basic knowledge of JSON web token methodology is strongly encouraged before getting the hands on IDM.

App ID Token

AppID token is a generated token sent on every IDM API request to identify the application and validate it's usage.

This token is sent in the header or query parameter `AppAuth` and calculated by the Application Access key and secret. The generated token includes the current unix timestamp and API allows tokens generated at maximum of 300 seconds before the request. Therefore, it should be calculated/generated at every request to avoid using expired tokens.



Above the token generation and validation process is briefly illustrated. For more detailed explanation, please, check below.

1-

- Access token and secret SHA1 is generated from the string `{“token”:”AppToken”, “secret”:”AppSecret”}` where AppToken and AppSecret is replaced with the actual keys. This first token is known as `verificationToken` and will be used on the next step.
- Another string is created from `{“token”:”verificationToken”,”timestamp”: 158267906300 }`. Where verificationToken is the token generated on the previous step and timestamp is the unix timestamp in **milliseconds** not in seconds as the usual timestamp formatting.

- Create AES-256-CTR of the string created on the previous step with a random 16 bytes initialization vector (IV) and salted with the key provided by the system administrator and referenced as AppID rotative key. If no key was provided, please, ask the system administrator.
- And finally but not least, format your hash as `IV:AES`. Where IV is your 16 bytes random initialization vector and AES is your previously generated AES hash. This is your **AppID** valid for the next 300 seconds.

Important Note: This algorithm is implemented on IDM ES6/Nodejs module on the class AppID, file AppID.js. Developers are extremely encouraged to use the existing implementation or base their code into this existing class.

Setup

1. Define application name, what roles and what are their behaviour on the application, what partitions and informations will be saved on IDM and request it to be created on the development environment by your system administrator.
2. Application access token and secret, partitions namespaces, roles IDs, API endpoint, SNS Reflection Topic arn and admin account will be provided by you system administrator.