

Intercept Everywhere

Web Services Architecture

Version: 1.0

Revision History

|  |  |  |
| --- | --- | --- |
| **Revision** | **Added by** | **Details** |
| 1.0 | Ramana Mylavarapu | Initial release |
| 1.1 | Ramana Mylavarapu  Atif Goheer | * Updated the architecture diagram, description * Registration flow * Updated the document with Web Services JSON templates and response codes |
|  |  |  |

Table of Contents

[2 Overview 4](#_Toc466570938)

[3 Architecture 4](#_Toc466570939)

[4 Interaction with Web Services 5](#_Toc466570940)

[5 User Management 9](#_Toc466570941)

[Add User 9](#_Toc466570942)

[Reset Password 9](#_Toc466570943)

[Disable User 9](#_Toc466570944)

[Delete User 9](#_Toc466570945)

[List Users 10](#_Toc466570946)

[6 System Services Configuration Management 10](#_Toc466570947)

[Add Service 10](#_Toc466570948)

[Modify Service 10](#_Toc466570949)

[Delete Service 10](#_Toc466570950)

[Show Service 10](#_Toc466570951)

[List Services 10](#_Toc466570952)

[7 User Sign Up, Registration & Login 11](#_Toc466570953)

[These calls are initiated by the App. 11](#_Toc466570954)

[Signup 11](#_Toc466570955)

[Signup Confirmation 11](#_Toc466570956)

[Registration 11](#_Toc466570957)

[User Login 11](#_Toc466570958)

[User Logout 11](#_Toc466570959)

[Forgot password Request 12](#_Toc466570960)

[Forgot password Confirmation 12](#_Toc466570961)

[8 User Services Configuration 12](#_Toc466570962)

[These calls are initiated by the App. 12](#_Toc466570963)

[Add Service 12](#_Toc466570964)

[Modify Service 12](#_Toc466570965)

[Delete Service 13](#_Toc466570966)

[9 API Calls 14](#_Toc466570967)

[9.1 App Registration 14](#_Toc466570968)

[10 User APIs 14](#_Toc466570969)

[User Service Configuration 20](#_Toc466570970)

[11 Miscellaneous 24](#_Toc466570971)

[a) Encryption (Perfect Forward Secrecy) 24](#_Toc466570972)

[b) Authentication 24](#_Toc466570973)

[c) Password Protection 24](#_Toc466570974)

[d) API Versioning 25](#_Toc466570975)

[e) Communication 25](#_Toc466570976)

# Overview

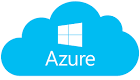
This document provides architecture of Everywhere Web Services framework for phase-1 of the project.

# Architecture

Refer to following architecture diagram. The solution includes the following components:

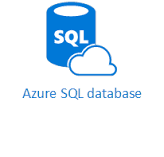
1. Web Portal (Customer self-service portal)
2. Admin Portal (Backend Administration portal)
3. Web Services
4. SQL Database
5. Redis

The solution is deployed in Azure and use Azure SQL Database for persistence data storage.



Admin Portal

Web Services



Web Portal



**Mobile Apps**



Web Services

Web Services

Video Flow

Video Switch Platform

Web portal for customers to sign up and do live streaming from mobile devices by logging in from the applications like gopro.

Admin portal is used for System Service Configuration and User Management. In this phase the portal functionality is realized using Web Services calls. Following are the Admin Web services:

1. System Service Configuration
2. User Management

The Apps in mobile devices and Web client use the following Web Services:

1. User sign up, Registration & Login
2. User Services Configuration

SQL database is used for persistent data storage. Redis is used for storing session token and runtime user information for performance optimization.

# Interaction with Web Services

Web Services will have the following configuration to support registration of Apps, Video Switches and Admins.

* Default 64 bit Key for Apps called “Default\_App\_User\_Key”
* Default 64 bit Key for Apps called “Default\_Web\_User\_Key”
* Default 64 bit Key for Video Switches called “Default\_Video\_Switch\_User\_Key”
* Default 64 bit Key for System Admin Users called “Default\_Admin\_Key”

The following flow diagrams explain the interaction procedure with Web Service by Mobile Apps and Video Platform for securely accessing the web services and session persistence.

#### Web Service Interactions

A default User Key which is specific App, Web portal, admin portal and video switch is used by to establish initial handshake by registering with the Web Service (Hereafter called Server). The server then provides a unique registration token which will be used by the client for further interactions with the Server. The following registration flow is same for Apps, Web and Admin users. The purpose of registration is to allow access to the Web Services only through this user key initially and registration token after that. This will help in preventing attacks on the Web Services and help mapping the user with the GUID, support multiple sessions by same user and have visibility of the user accessing the service right from the registration.

App

Web Service

App establishes HTTPS Session with the Server

App sends Registration Request with Default Key and UUID of the device

Server returns a Registration Token

[App uses this Registration Token for subsequent interactions with the Server]

Registration of the App [User not logged-in]

App

Web Service

App establishes HTTPS Session with the Server

App sends Registration Token, UUID of the device and User Credentials

Server returns a Session Token along with login results

[App uses this Session Token for subsequent interactions with the Server]

User login

App

Web Service

App establishes HTTPS Session with the Server

App sends Session Token, UUID of the device and Web Service Request

Server returns an updated Session Token along with Web Service response

[App uses this Session Token for subsequent interactions with the Server]

App Accessing Web Services

#### Video Switch-Web Service Interactions

* The Video Switches will have the following configurations:

1. Default\_Video\_Switch\_User\_Key for them to send with credentials for login.
2. Send Session Token with each subsequent session Request.

Video Switch

Web Service

Switch establishes HTTPS Session with the Server

Switch sends Login Request with Default Key and Credentials

Server returns a Session Token and Login results

[Switch uses this Session Token for subsequent interactions with the Server]

Video Switch Login

Web Service

Switch establishes HTTPS Session with the Server

Switch sends Session Token and Web Service Request

Server returns an updated Session Token along with Web Service response

[Switch uses this Session Token for subsequent interactions with the Server]

Video Switch Accessing Web Services

Video Switch

#### Admin Portal -Web Service Interactions

* The Admin portal will have the following configurations:

1. Default\_Admin\_Key and credentials for it to login with the Web Service
2. Admin Credentials: We will create default admin user by inserting the user into the database.

Admin Portal

Web Service

Admin Portal establishes HTTPS Session with the Server

Switch sends Login Request with Default Key and Credentials

Server returns a Session Token and Login results

[Admin uses this Session Token for subsequent interactions with the Server]

Admin Login

Web Service

Admin Portal establishes HTTPS Session with the Server

Switch sends Session Token and Web Service Request

Server returns an updated Session Token along with Web Service response

[Admin uses this Session Token for subsequent interactions with the Server]

Admin Accessing Web Services

Admin Portal

# User Management

These calls are initiated through Admin portal by user type “Admin”. Initially we will use default admin credentials for admin to login.

## Add User

Requires:

* User ID
* User Type (AppUser/VideoSwitch/Admin)

## Reset Password

Requires:

* User ID

System will send Forgot Password Code via email to the user’s Email ID.

## Disable User

Requires:

* User ID
* Notify User (YES/NO)

## Delete User

Requires:

* User ID
* Notify User (YES/NO)

## List Users

Requires:

* User State (ALL/SIGNEDUP/REGISTERED/REGISTRATION\_ INCOMPLETE)

# System Services Configuration Management

These calls are initiated through Admin portal by user type “Admin”. Initially we will use default admin credentials for admin to login.

## Add Service

Requires:

* Service Name
* Authentication Method (CRENTIALS/oAUTH/SAML)
* Service Provider Info

## Modify Service

Requires:

* Service Name
* Authentication Method (CRENTIALS/OAUTH/SAML)
* Service Provider Info

## Delete Service

Requires:

* Service Name

## Show Service

Requires:

* Service Name

## List Services

Requires:

None

# User Sign Up, Registration & Login

## These calls are initiated by the App.

## Signup

Requires:

* Email ID
* Name
* Password
* Date of Birth
* Device Information (optional)
* Geo-location (optional)

### 

### Signup Confirmation

* System will send Confirmation Code via email to the Email ID specified
* Pending registration will expire if not activated within 1 week

## Registration

* User enters Confirmation Code to confirm the sign up and continue with registration process.
* Application will ask user’s permission to query the following information “Device & Location Information”:
* Device make, model, UUID, its video/audio capabilities, etc.
* Registration requires:
* Confirmation Code
* App steps through Service Configuration (this can be made mandatory or optional and can be prompted to the user to completed).

## User Login

Requires:

* Email ID
* Password

## User Logout

Requires:

* Email ID

## Forgot password Request

* System will send Forgot Password Code via email to the Email ID specified

Requires:

* Email ID

## Forgot password Confirmation

* User enters Confirmation Code to confirm the sign up and continue with registration process.

Requires:

* Email ID
* Confirmation Code
* New Password

# User Services Configuration

## These calls are initiated by the App.

## Add Service

* App shows list of services and user selects each of one of them.
* As per Authentication Method of the Service, App prompts the user to enter Credentials (if the method is CREDENTIALS) or makes users to authenticate with the Service (if it is OAUTH) and provides authentication info.
* System validates the credentials/oAuth artifacts with the Service and provides the results in response to this call.

Requires:

* User ID
* Service Name
* Authentication Info: Credentials or OAUTH artifacts

## Modify Service

* User may want to change Service Credentials or reauthenticates with the service through oAuth (if the credentials happened to have changed)

Requires:

* User ID
* Service Name
* Authentication Info: Credentials or OUATH artifacts

## Delete Service

Requires:

* User ID
* Service Name

# API Calls

# App Registration

* + 1. **Register an Application**
* /api/v1/app/register
* Method: POST
* Parameters
  + appName {required}
  + uuid {required}
  + version {required}
  + os {required}
  + defaultKey {required}
* Result

|  |  |
| --- | --- |
| **Http Status Code** | **Message** |
| **201** | **Registration Successful**   * **Result:** * {"registration\_token": "HUccNuPGMJcTfLzh0D0Ec9jCVEnqp84dUb1hOiJYeksRrsXkHe"} |
| 400 | Required field error -- Verified   * **Result:**   {  "Message": "The request is invalid.",  "ModelState": {"model.os": ["The Operating System is required"]}  } |
| 500 | Oops! An error occurred while registering |
| 409 | Sorry, this Application is already registered -- Verified   * **Result:**   {"Message": "App already exists."} |

# User APIs

* 1. **Check Email Available for Registration**
* /api/v1/user/check
* Method: POST
* Parameters
  + email {required}
* Result

|  |  |
| --- | --- |
| **Http Status Code** | **Message** |
| **404** | **Email Address Available --** Verified   * **Result:**   Raw: HTTP/1.1 404 Not Found |
| 409 | Email Address In Use   * **Result:**   Raw: HTTP/1.1 409 Conflict |
| 500 | Oops! An error occurred |

* 1. **Register User**
* /api/v1/user/register
* Method: POST
* Parameters
  + email {required}
  + password {required}
  + name {required}
  + dob
  + geolocation
* Result

|  |  |
| --- | --- |
| **Http Status Code** | **Message** |
| **201** | **Registration Successful**   * **Result:**   HTTP/1.1 201 Created |
| 400 | Required field error   * **Result:**   {  "Message": "The request is invalid.",  "ModelState": {"model.email": ["Email address is required"]}  } |
| 500 | Oops! An error occurred while registration |
| 409 | Sorry! User already exists   * **Result:**   {"Message": "User already exists."} |

* 1. **Login User**
* **/api/v1/authenticate**
* Method: POST
* Parameters
  + email{required}
  + password{required}
  + key{required} [User = App Registration Token, Admin/Video Switch = Respective Default Token]
  + uuid {required for app user only}
* Result

|  |  |
| --- | --- |
| **Http Status Code** | **Message** |
| 200 | **Success**  *{*  *"token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJlbWFpbCI6ImFiaWQuYWxpQGlxdmlzLmNvbSIsInJvbGUiOiJVc2VyIiwic3ViIjoiOTkwNGJkZDAtZjljZS00YzE4LTk1ODAtM2M5Mzg4OTIxYzZiIiwic2lkIjoiMGE3MTMyYTktMDAxZS00YTc4LTkyMDAtYmU3NTlhZjgwNzgzIiwibmJmIjoxNDk0OTQyNDk1LCJpYXQiOjE0NzkzMDQwOTUsImV4cCI6MTQ3OTMyMjA5NX0.g3ZjUJlxbAw2OmaQpEmsvgQxrceaFur2WdGx5KTx1f4",*  *"id": "4"*  *}* |
| 400 | Required field error   * **Result:**   {  "Message": "The request is invalid.",  "ModelState": {"loginInfo.email": ["The email address is required"]}  } |
| 500 | Oops! An error occurred while login |
| 401 | Incorrect Login Information   * **Result:**   Raw: HTTP/1.1 401 Unauthorized |

* 1. **Reset User**
* **/api/v1/reset**
* Method: POST
* Parameters
  + Email{required}
* Result

*{*

*"error": false,*

*“message”:”* *email successfully sent”*

*"user": {*

*"message": "* system will send forgot password code via email ID*"*

*}*

*}*

*{error:true, error\_code, message}*

|  |  |
| --- | --- |
| **Http Status Code** | **Message** |
| 200 | Success |
| 400 | Required field error |
| 500 | Oops! An error occurred while reset |
| 404 | Account with given Email does not exist |

* 1. **LogoutUser**
* **/api/v1/signout**
* Method: GET
* Parameters
  + N/A
* Result

|  |  |
| --- | --- |
| **Http Status Code** | **Message** |
| 200 | Success   * **Result:**   HTTP/1.1 200 OK |
| 500 | Oops! An error occurred while logout |

* 1. **Forgot password confirmation**
* /api/v1/resetpass
* Method: POST
* Parameters
  + emailID{email}{required}
  + confirmationCode{required}
  + newPassword{required}
* Result

*{*

*"error": false,*

*“message”:”* *password successfully reset”*

*"user": {*

*“status”:”true”*

*}*

*}*

*{error:true, error\_code, message}*

|  |  |
| --- | --- |
| **Http Status Code** | **Message** |
| 200 | Success |
| 400 | Required field error |
| 500 | Oops! An error occurred while reset the password |
| 404 | emailID does not exist |
| 417 | Confirmation code is not valid |

**User Management**

* 1. **Delete User**
* **/api/v1/users/{id}**
* Method: DELETE
* Parameters
  + id{required}
  + notifyUser
* Result

No Response

|  |  |
| --- | --- |
| **Http Status Code** | **Message** |
| 204 | Success   * **Result Updated:**   HTTP/1.1 204 No Content |
| 400 | Required field error |
| 500 | Oops! An error occurred while delete the user |
| 404 | User with ‘id’ does not exist |

* 1. **Disable User**
* /api/v1/users/disable/{id}
* Method: POST
* Parameters
  + id{required}
  + notifyUser{required}
* Result

*No Response*

|  |  |
| --- | --- |
| **Http Status Code** | **Message** |
| 204 | Success   * **Result Updated:**   HTTP/1.1 204 No Content |
| 400 | Required field error |
| 500 | Oops! An error occurred while disable the user |
| 404 | User with ‘id’ does not exist |

* 1. **GetUser Info**
* **/api/v1/users/{id}**
* Method: GET
* Parameters
  + id{required}
* Result

*[{*

*"id": 4,*

*"email": "abid.ali@iqvis.com",*

*"firstName": "Abid",*

*"lastName": ""*

*}]*

|  |  |
| --- | --- |
| **Http Status Code** | **Message** |
| 200 | Success |
| 400 | Required field error |
| 500 | Oops! An error occurred while listing up the users |
| 404 | Provided user id is not valid |

# User Service Configuration

* 1. **Register Service**
* /api/v1/user/service
* Method: POST
* Parameters
  + serviceName {required}
  + AuthenticationMethod {required}
  + ServiceProviderInfo {required}
  + Usertoken
  + userID
* Result

|  |  |
| --- | --- |
| **Http Status Code** | **Message** |
| 201 | Created   * **Result:**   HTTP/1.1 201 Created |
| 400 | Required field error |
| 500 | Oops! An error occurred while registering |
| 409 | Sorry, this Service is already registered |

* 1. **Login Service**
* /api/v1/service
* Method: POST
* Parameters
  + ServiceName {required}
  + usertoken
* Result

*{*

*"error": false,*

*“message”:”User* *successfully logged in for this service”*

*"service": {*

*“serviceName”:”someservice”,*

*“*AuthenticationMethod*”:”credential/oAuth/SAML”,*

*“*ServiceProviderName*”:”someprovider”,*

*"access\_toknen":*

*"tqusavaVPuxsuSKFdﬂAil76JkqunSpSuTMM7AYS\_eo\_sLoaaius2\_VMu1uﬂrLM6Yr-3nyqu1palrdﬂ*

*SOZNVHStquZIles-*

*MIPBOOOICSMB LlVlFbQCstﬂknat‘ Lrs958¢viﬂlszeprI rfI\_2ﬂ-G Ft-l‘SJVSRdRXxHXkL iOHEBTO‘I’-*

*72th1R1F\_1mlﬂfky‘~'o\_C¢ﬁsaiHZQS-95qa00932f26yux-hWIII-MS",*

*"issued": "27/10/2016",*

*"expires": "03/11/2016"*

*“created\_by”:””,*

*“updated\_by”:””*

*}*

*}*

*{error:true, error\_code, message}*

|  |  |
| --- | --- |
| **Http Status Code** | **Message** |
| 200 | Success |
| 400 | Required field error |
| 500 | Oops! An error occurred while login |
| 404 | Sorry, service does not exist |
| 401 | Sorry, provided token is not valid |

* 1. **Update Service**
* /api/v1/user/service/UpdateService
* Method: PUT
* Parameters
  + ServiceName {required}
  + AuthenticationMethod {required}
  + ServiceProviderName {required}
  + userID
* Result

|  |  |
| --- | --- |
| **Http Status Code** | **Message** |
| 204 | Success   * **Result Updated:**   HTTP/1.1 204 No Content |
| 400 | Required field error |
| 500 | Oops! An error occurred while updating service |
| 404 | Sorry, service does not exist |

* 1. **Delete Service**
* /api/v1/user/service/DeleteService
* Method: DELETE
* Parameters
  + ServiceName {required}
  + UserID
* Result

*{*

*"error": false,*

*“message”:”* *service is successfully deleted”*

*"service": {*

*“status”:”true”,*

*}*

*}*

*{error:true, error\_code, message}*

|  |  |
| --- | --- |
| **Http Status Code** | **Message** |
| 200 | Success   * **Result:**   HTTP/1.1 200 OK |
| 400 | Required field error |
| 500 | Oops! An error occurred while delete the service |
| 404 | Sorry, service does not exist |

* 1. **Show Service**
* /api/v1/user/service/GetService
* Method: GET
* Parameters
  + ServiceName {required}
* Result

|  |  |
| --- | --- |
| **Http Status Code** | **Message** |
| 200 | Success   * **Result:**   [{  "ID": "00000000-0000-0000-0000-000000000000",  "name": "service1",  "authenticationMethod": "someMethod",  "serviceProviderInfo": "url"  }] |
| 400 | Required field error |
| 500 | Oops! An error occurred while getting the service information |
| 404 | Sorry, service does not exist |

* 1. **List Service**
* /api/v1/user/service/GetServices
* Method: GET
* Parameters
  + null
* Result

*[{*

*"ID": "00000000-0000-0000-0000-000000000000",*

*"name": “serivce1”,*

*"authenticationMethod": "someMethod",*

*"serviceProviderInfo": "url"*

*}]*

|  |  |
| --- | --- |
| **Http Status Code** | **Message** |
| 200 | Success   * **Result:**   HTTP/1.1 200 OK |
| 400 | Required field error |
| 500 | Oops! An error occurred while listing the service |
|  |  |

# Miscellaneous

# Encryption (Perfect Forward Secrecy)

*All communication with the Restful endpoints takes over TLS connection.* To ensure most secure communication, encryption algorithms that support “Perfect Forward Secrecy” will be prioritized on the Server. No special settings or connection method is required on the client side.

# Authentication

After the client successfully authenticates with the Restful, the service will return a Json Web Token (JWT). The token will contain various info regarding the client e.g. name, UUID, email, role/type, and token expiry date.

The client must send the token in the HTTP header of any subsequent request to the endpoint to identify itself correctly. The client needs to refresh token by calling the respective API before it expired. The client can also call the logout API to force expire a token. This will result in immediate invalidation of the token.

# Password Protection

All Passwords stored in Database are hashed using PBKDF2. The min number of iterations should be at least 20,000. It can be increased to any amount e.g. 100K, however, that would take more CPU on the server side, so an appropriate number needs to be chosen based off of expected traffic and hardware capability.

# API Versioning

API version will be included in the URI – as per current industry practice.

https://www.domainname.com/api/**v1**/resourceName<parameters>

# Communication

All communication between the clients and web services will use JSON formatted data and HTTP status codes.

As per industry practice, all APIs that require sensitive information being sent from client to web service will use POST verb. GET, DELETE and PUT will be used in appropriate times.

1. **Glossary**

*Client*: Client of the Restful endpoint. Could be a user, video switch or admin

*User*: Client of the Service with non-admin privileges

*Admin*: Client of the Service with admin privileges

*Video Switch*: Client of the Service for automated

*JWT*: Json Web Token (<https://jwt.io/>)

PBKDF2: Password Hashing Algorithm <https://en.wikipedia.org/wiki/PBKDF2>