Table of Contents

3M IoT Platform Guides

Welcome to the public documentation on guides that cover common activities of working with the platform.

Managing Devices

OVERVIEW

This is what a card might look like.

User Manual

FIRST SECTION

This is what a card might look like.

Overview of IoT Devices on Azure

Conceptually, there are two types of devices that can be used in Azure: 1) Simulated and 2) Real.
Simulated devices are used to test connectivity and offer a quick way to evaluate functionality without needing to procure any hardware.
There are two types of real devices: IoT Edge device and IoT device.

If you'd like to learn more about the types of devices that can work with Azure check out Microsoft's Device Catalog

Device Registration

Before you can use a real or simulated device, you must first register it. Use the following links to get started:

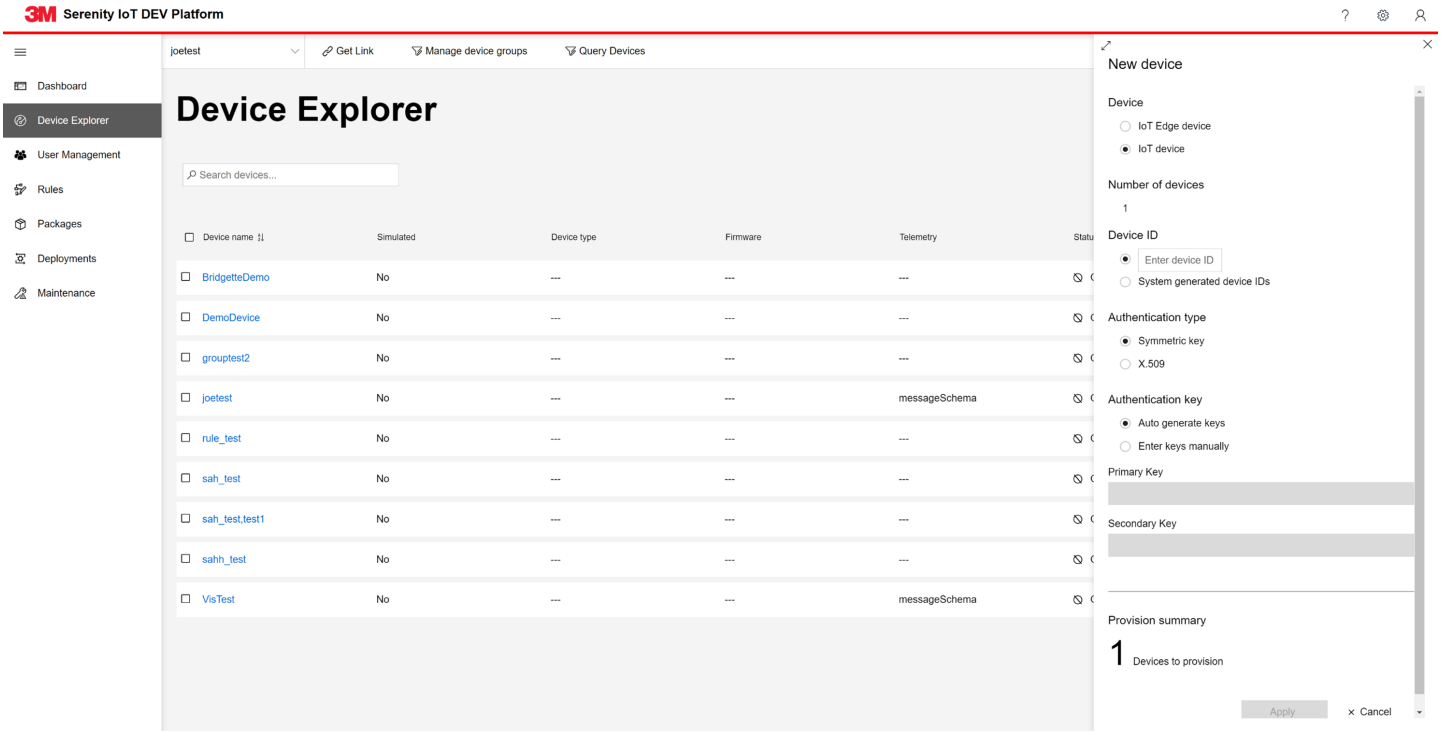
- Register a device
- Register a simulated device

How to Register a Device

For a device to connect to the solution accelerator, it must identify itself to IoT Hub using valid credentials. You have the opportunity to save the device connection string that contains these credentials when you add the device to the solution. You include the device connection string in your client application later in this tutorial.

To add a device to your Remote Monitoring solution, complete the following steps on the Device Explorer page in the solution:

1. Choose + **New device**, and then choose **Real** as the **Device type**:



1. Enter a Device ID you wish to use to reference this device or allow the system to generate one for you. The value must be unique and connect include spaces.

2. Select the desired authentication type and either have the system generate authentication keys for you or provide your own. CRSL generally uses a Symmetric Key but your organization might prefer x509. Chose whichever is the most approriate.

3. Select the preferred option for how to define keys. You have the choice between:

    - Providing your own
    - Auto Generating

How to Register a Simulated Device

User Management Overview

Describe all of what's covered in this guide.
Describe the audiences.
Put some cards in her for popular quick hits

User Registration Prerequisites

This document describes the prerequisites related to registering users in the 3M IoT Platform on Azure. The following describes the prerequisites for:

- **Invitee**: The person invited to use the application
- **Inviter**: The person inviting the new user

Invitee

The invitee must have the following to join a tenant:

- **Valid Email:** A valid email of the user to be added is required

In addition, their responsibilities should be determined prior to invite to determine the appropriate role, which is required to invite a user.

To learn more about the type of roles the user can be assigned to, see the Role and Permission documentation found here: TODO

Inviter

The following prerequisites are necessary to invite a user:

- **Tenant:** A user can only be invited to an existing tenant.
- **InviteUser Permissions:** The inviter must be assigned InviteUser permissions

InviteUser permissions is typically assigned through a role. The table below highlights which roles grant the permissions to be able to invite a user:

PERMISSIONS NAME	ADMIN	CONTRIBUTOR	READ ONLY
Invite users	Yes	No	No

> There is a cross-tenant roll called SysAdmin that also has permission to assign users.

A user can be invited to 3M IoT Platform from application.

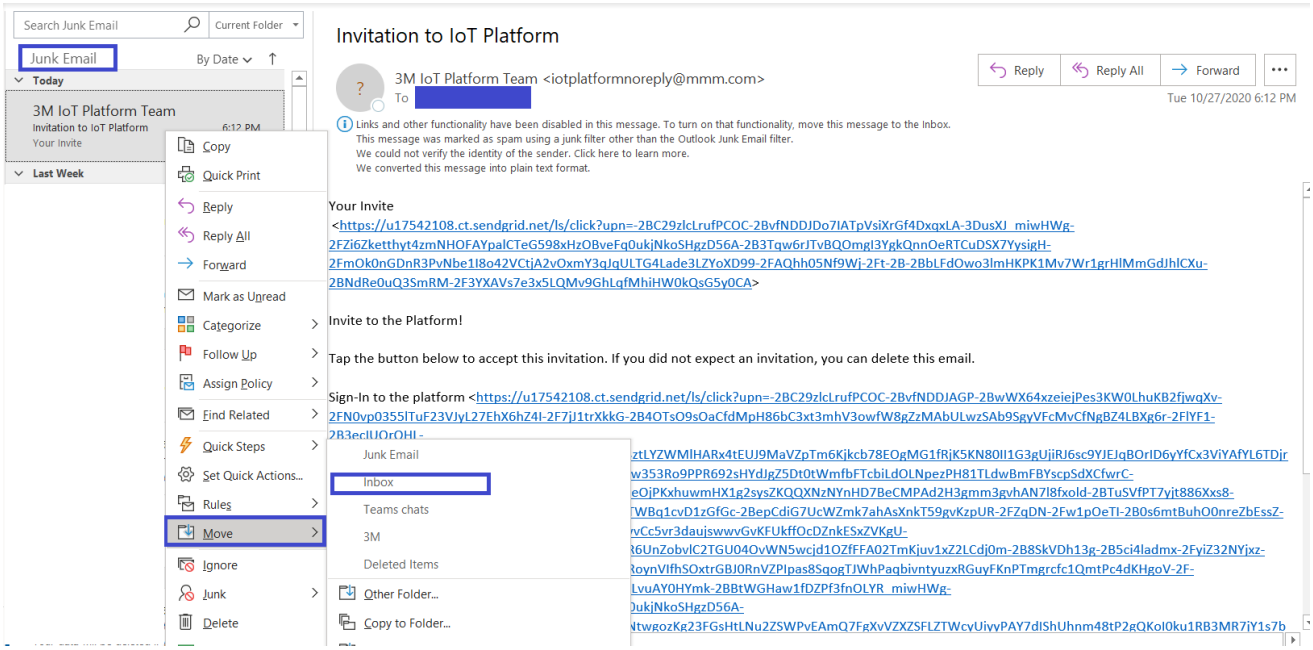
Refer steps from User Manual to invite a new user to the platform.

3M User Registration

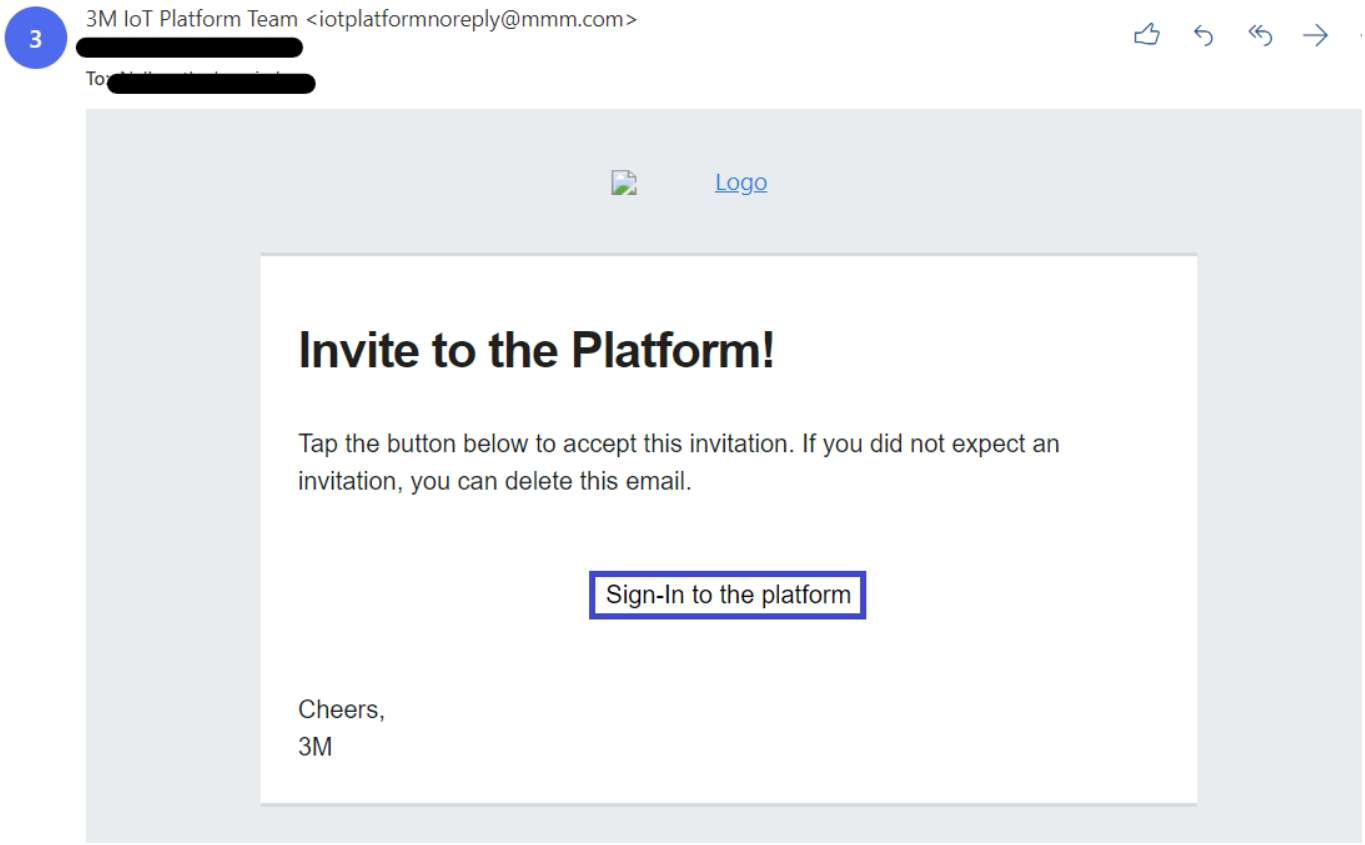
When a user is invited to a tenant, they will receive in email that must be used to complete the registration process so they can sign in. This document guides the invited user through the registration process so that they can access the intended resource, which might be the Serenity UI or a custom application built on top of Serenity.

3M Registration

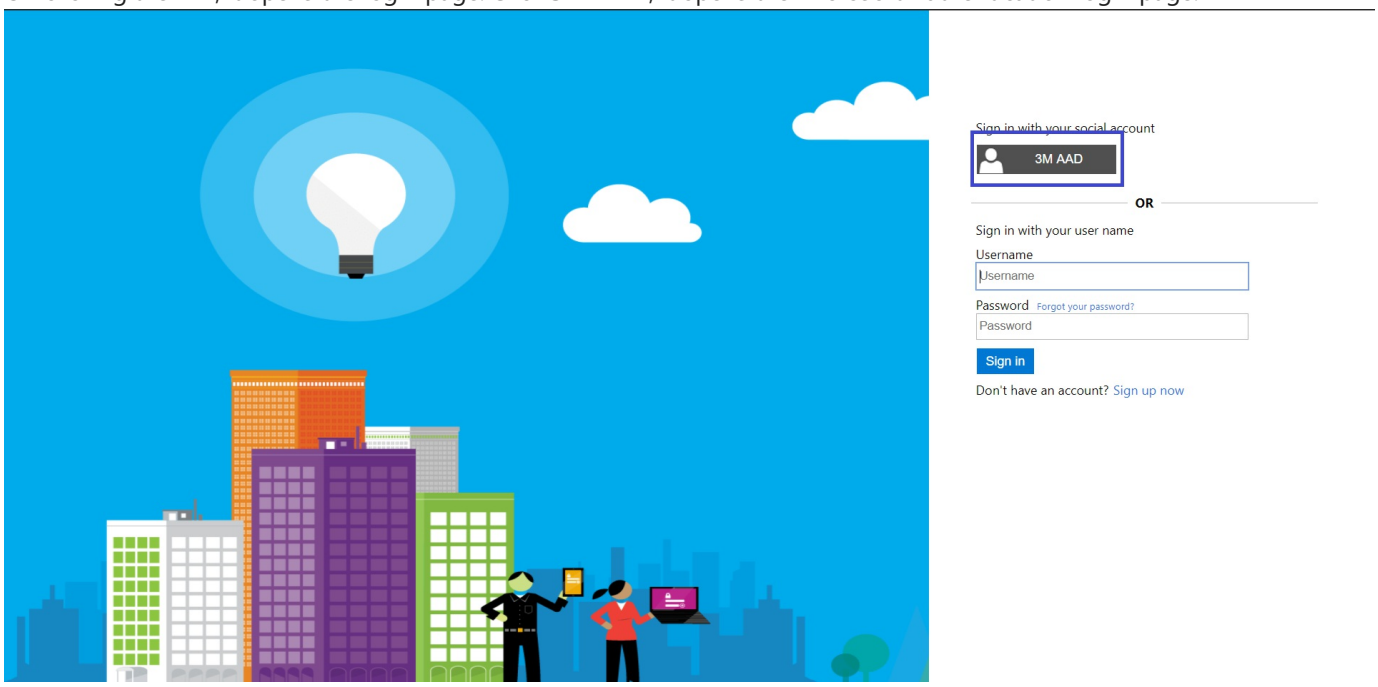
- Find the email invite
  - An email invite is sent to your mail address. Check for it in your inbox.
  - If the email invite is not found in your inbox, see if it ended up in the Junk folder
  - Once found, right click on the email and select Move -> Inbox(any other folder of your choice). This allows enabling the links in the email.



- Next, open the mail and click **Sign-In to the platform**.

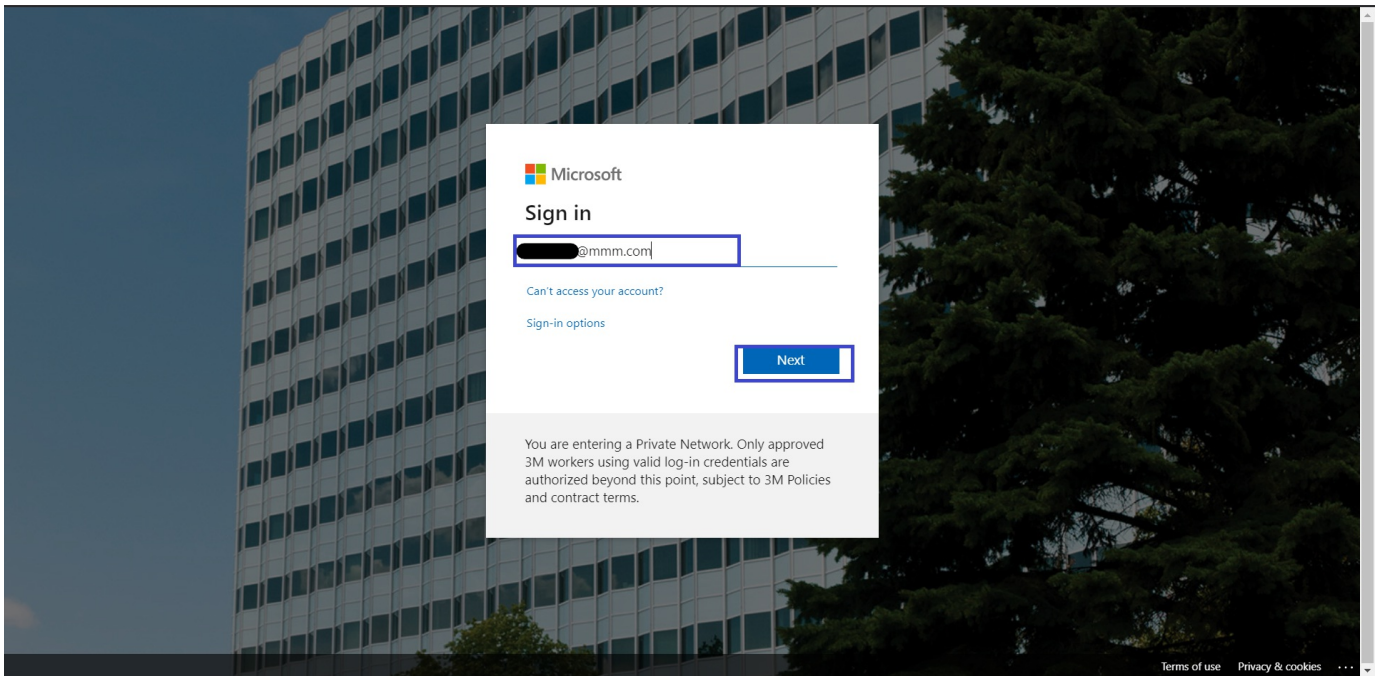


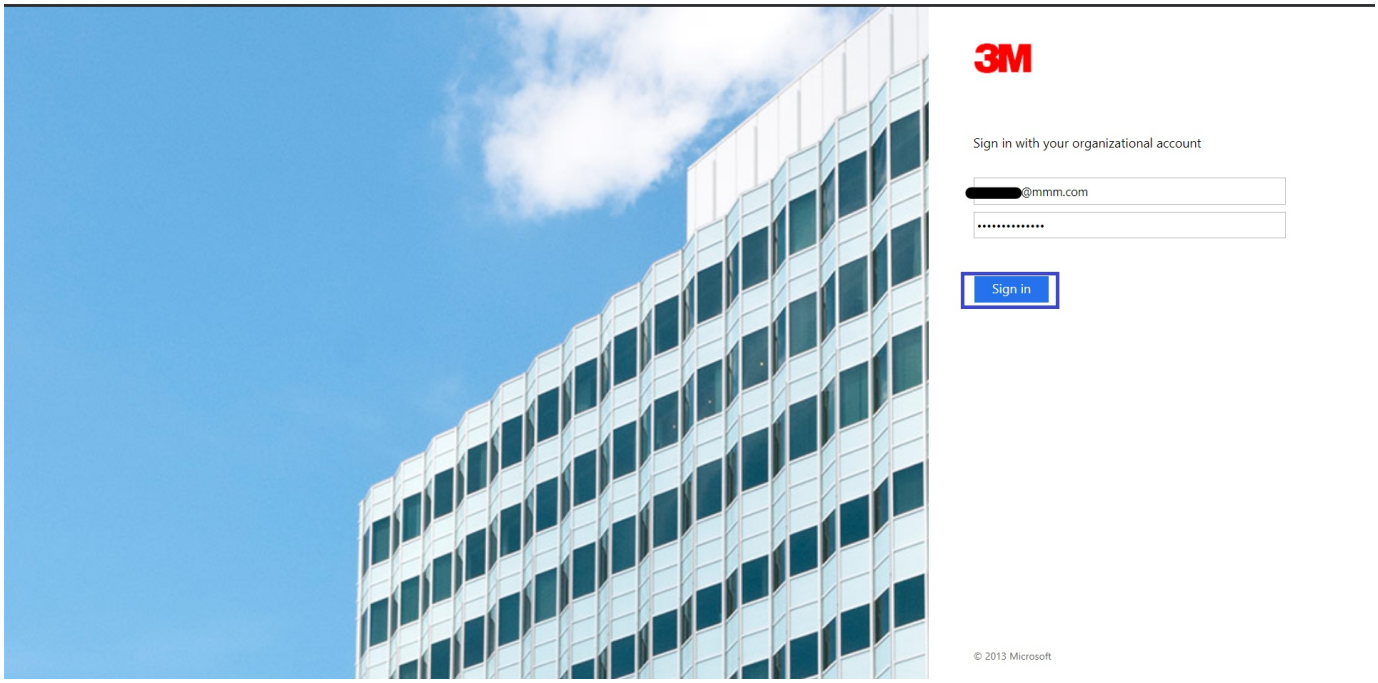
- On clicking the link, it opens the login page. Click **3M AAD**, it opens the Microsoft Authentication login page.



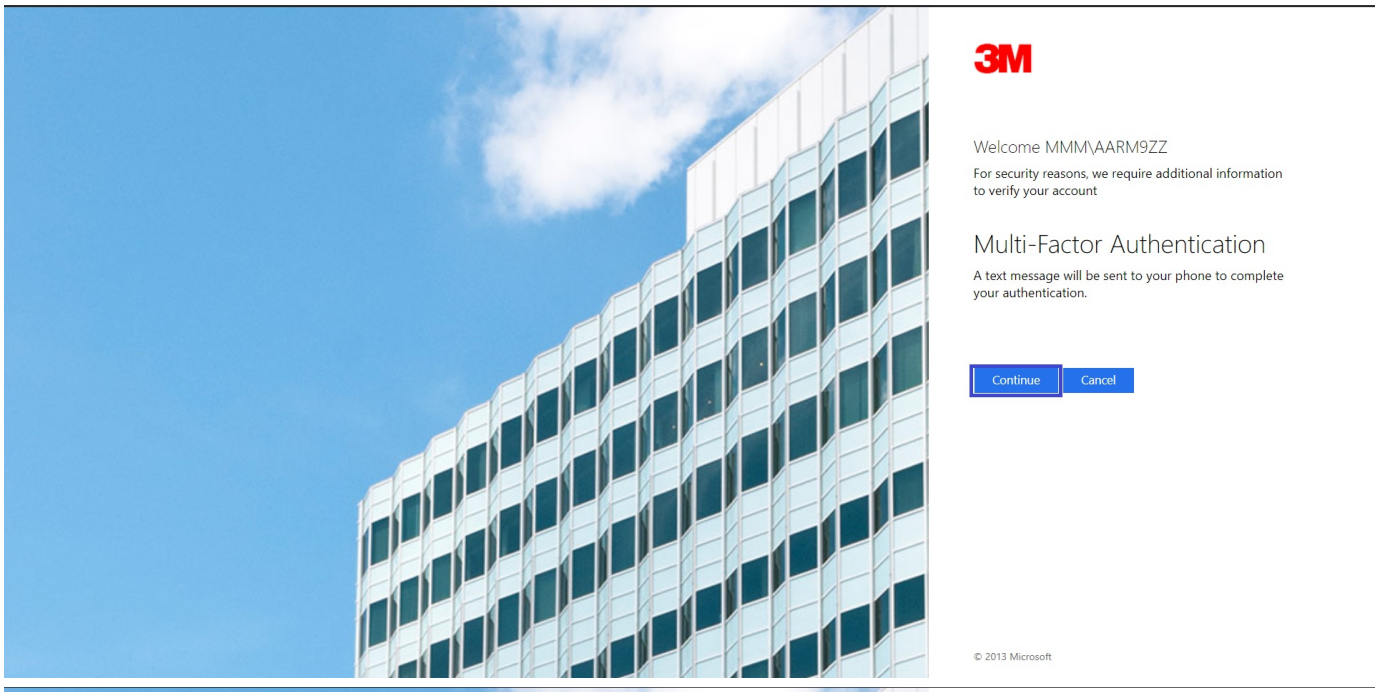
> **Note**
>
> If on click of the link does not open the Microsoft Authentication login page, copy the link from the email and try it in the web browser incognito tab.

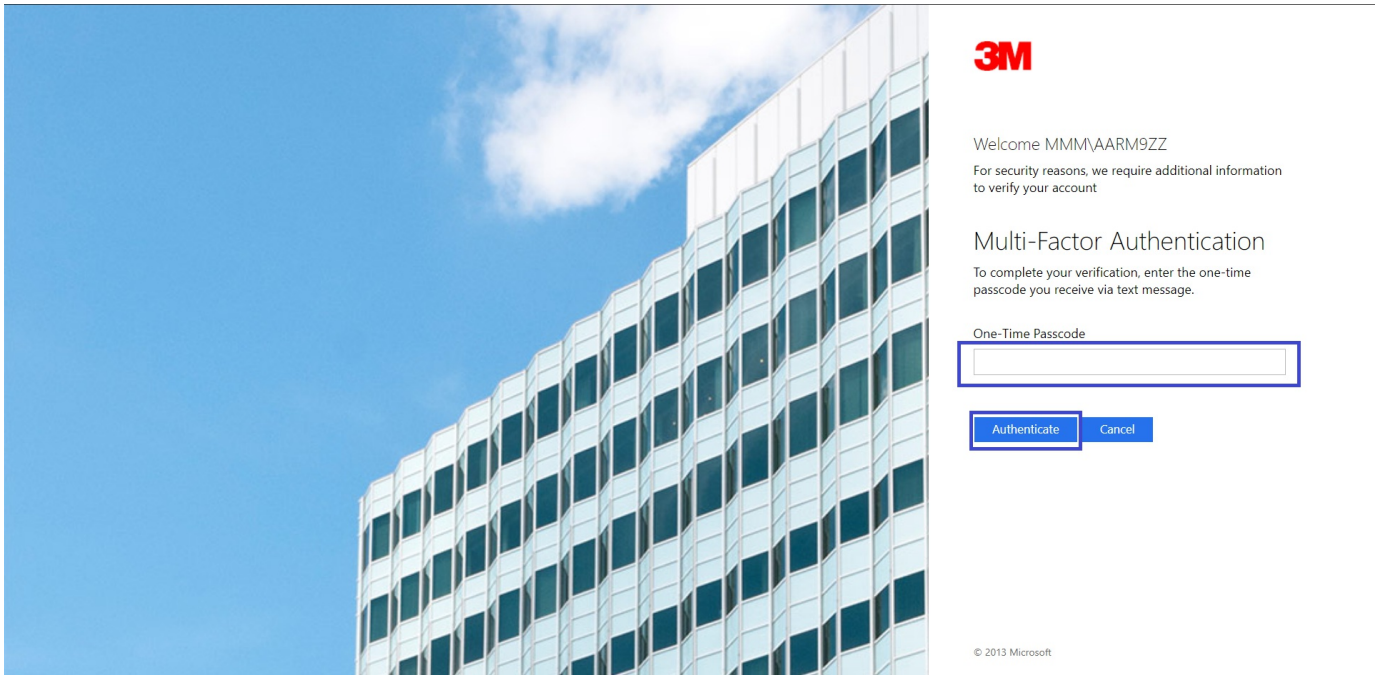
- Provide the 3M login details, click **Next**, on verifying the email address it redirects to the 3M Authentication page. Enter the login details and click **Sign in**.



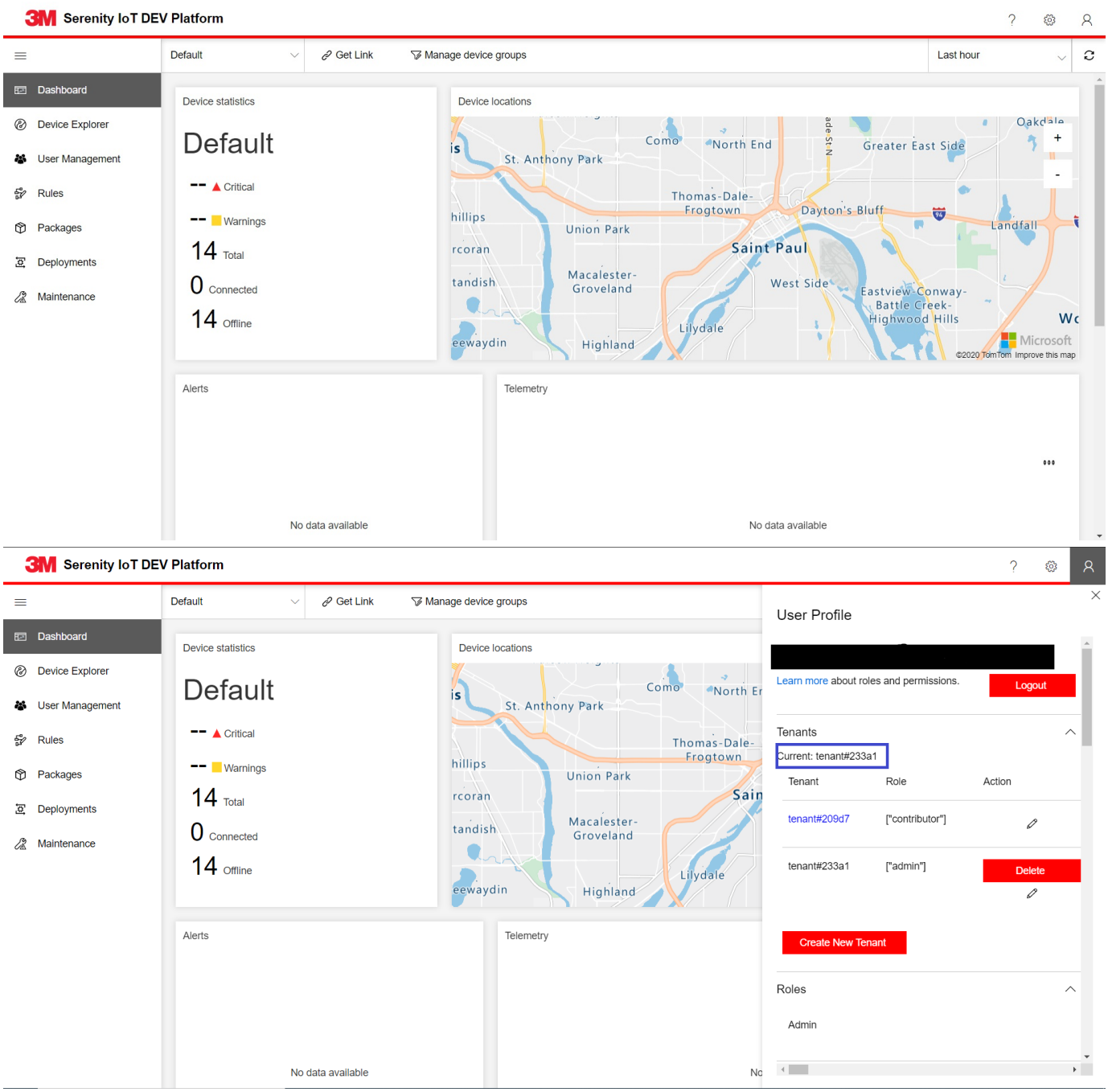


- If it asks for Multi-Factor Authentication, click **Continue**, a text message is sent to your registered mobile phone with an OTP. Enter OTP and click **Authenticate**.





- Once successfully authenticated, it will redirect to the 3M Remote Monitoring System. Click **User Profile** icon which is at the top right corner of the page. It opens **User Profile** panel and verify the tenant details for which you are invited to.



- With this 3M user is successfully registered to a tenant.

Add System Admin User

System admin is the user who has access across all the tenants. A System Admin can be added to a Serenity Instance using either of the following techniques:

1. Web Application
2. Pipeline(One-time)

Web Application

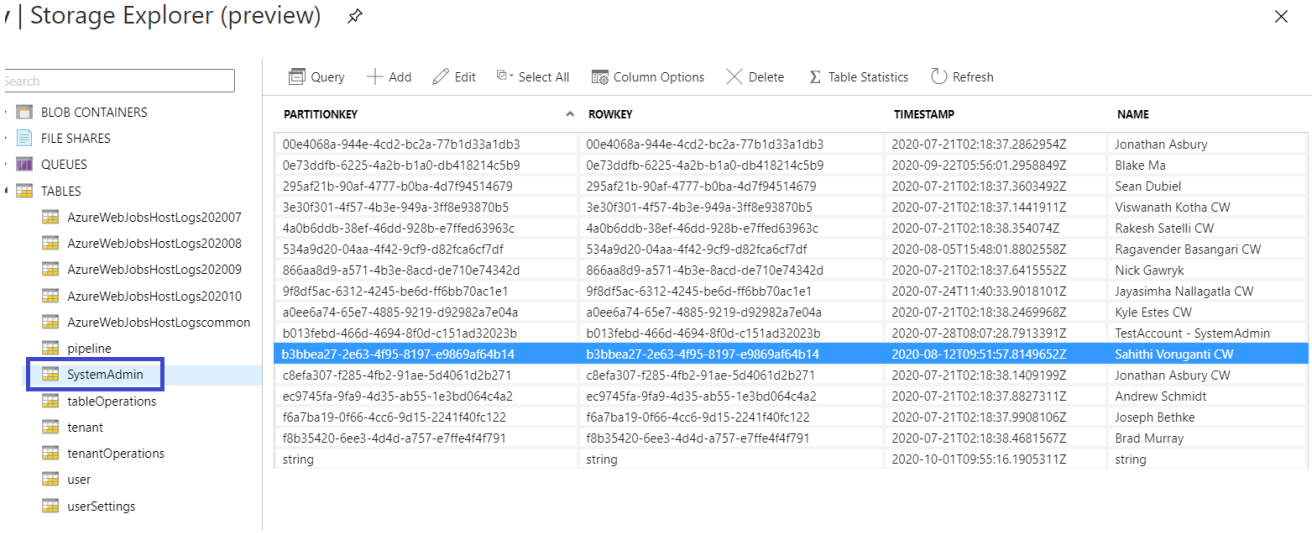
An existing user can be made as **SystemAdmin** to the System from Web Application. Refere the steps here for adding an existing user as SystemAdmin through Web Application.

Note: The user has to be a Non-SystemAdmin user to be added as a SystemAdmin to the system

Once a user is made SystemAdmin, the following happens:

1. A row is added for this user in **SystemAdmin** Azure Storage table with below details:
   - PartitionKey: UserID
   - RowKey: UserID
   - Timestamp: Datetime when the User is made SystemAdmin
   - Name: Name of the user

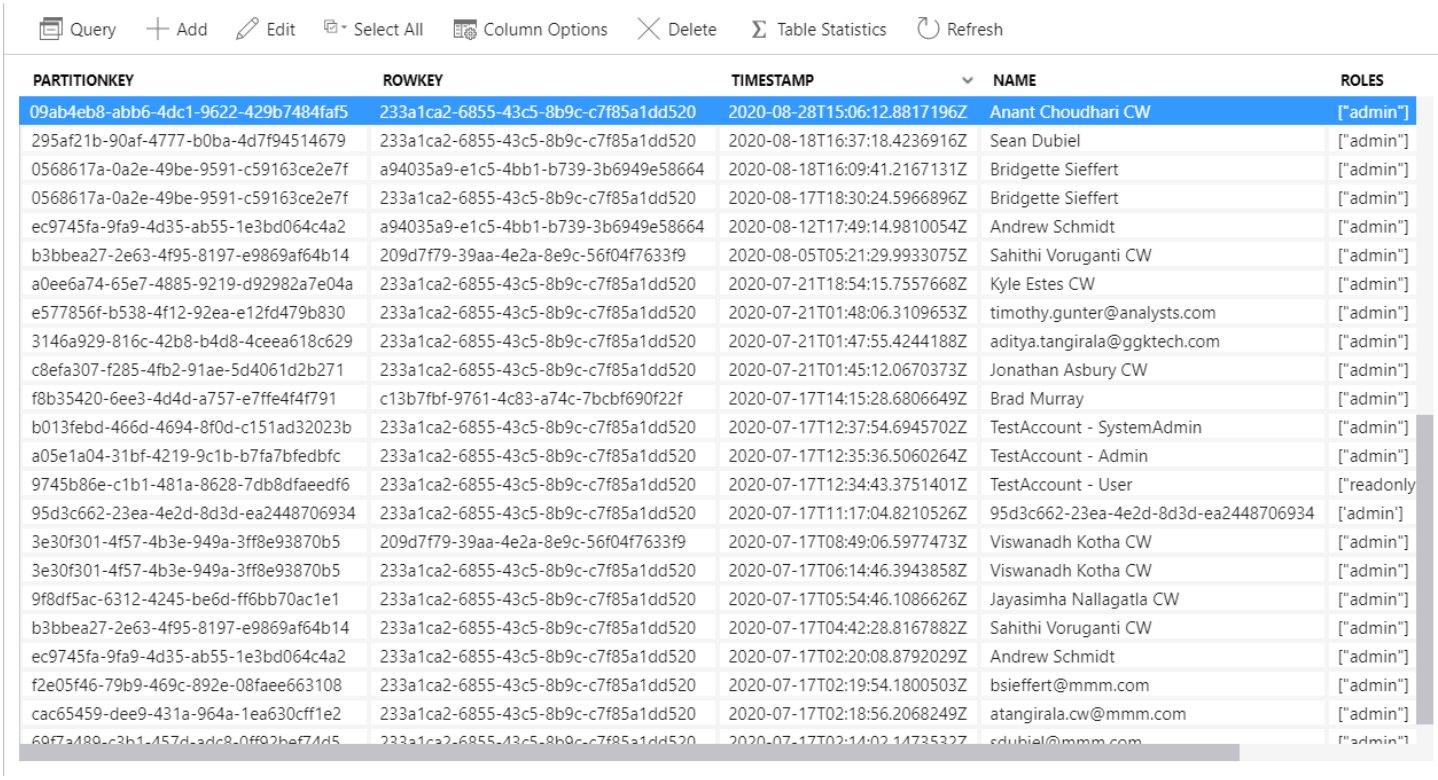
   Ex:



PARTITIONKEY	ROWKEY	TIMESTAMP	NAME
00e4068a-944e-4cd2-bc2a-77b1d33a1db3	00e4068a-944e-4cd2-bc2a-77b1d33a1db3	2020-07-21T02:18:37.2862954Z	Jonathan Asbury
0e73ddfb-6225-4a2b-b1a0-db418214c5b9	0e73ddfb-6225-4a2b-b1a0-db418214c5b9	2020-09-22T05:56:01.2958849Z	Blake Ma
295af21b-90af-4777-b0ba-4d7f94514679	295af21b-90af-4777-b0ba-4d7f94514679	2020-07-21T02:18:37.3603492Z	Sean Dubiel
3e30f301-4f57-4b3e-949a-3ff8e93870b5	3e30f301-4f57-4b3e-949a-3ff8e93870b5	2020-07-21T02:18:37.1441911Z	Viswanath Kotha CW
4a0b6ddb-38ef-46dd-928b-e7ffed63963c	4a0b6ddb-38ef-46dd-928b-e7ffed63963c	2020-07-21T02:18:38.354074Z	Rakesh Satelli CW
534a9d20-04aa-4f42-9cf9-d82fca6cf7df	534a9d20-04aa-4f42-9cf9-d82fca6cf7df	2020-08-05T15:48:01.8802558Z	Ragavender Basangari CW
866aa8d9-a571-4b3e-8acd-de710e74342d	866aa8d9-a571-4b3e-8acd-de710e74342d	2020-07-21T02:18:37.6415552Z	Nick Gawryk
9f8df5ac-6312-4245-be6d-ff6bb70ac1e1	9f8df5ac-6312-4245-be6d-ff6bb70ac1e1	2020-07-24T11:40:33.9018101Z	Jayasimha Nallagatla CW
a0ee6a74-65e7-4885-9219-d92982a7e04a	a0ee6a74-65e7-4885-9219-d92982a7e04a	2020-07-21T02:18:38.2469968Z	Kyle Estes CW
b013febd-466d-4694-8f0d-c151ad32023b	b013febd-466d-4694-8f0d-c151ad32023b	2020-07-28T08:07:28.7913391Z	TestAccount - SystemAdmin
b3bbea27-2e63-4f95-8197-e9869af64b14	b3bbea27-2e63-4f95-8197-e9869af64b14	2020-08-12T09:51:57.8149652Z	Sahithi Voruganti CW
c8efa307-f285-4fb2-91ae-5d4061d2b271	c8efa307-f285-4fb2-91ae-5d4061d2b271	2020-07-21T02:18:38.1409199Z	Jonathan Asbury CW
ec9745fa-9fa9-4d35-ab55-1e3bd064c4a2	ec9745fa-9fa9-4d35-ab55-1e3bd064c4a2	2020-07-21T02:18:37.8827311Z	Andrew Schmidt
f6a7ba19-0f66-4cc6-9d15-2241f40fc122	f6a7ba19-0f66-4cc6-9d15-2241f40fc122	2020-07-21T02:18:37.9908106Z	Joseph Bethke
f8b35420-6ee3-4d4d-a757-e7ffe4f4f791	f8b35420-6ee3-4d4d-a757-e7ffe4f4f791	2020-07-21T02:18:38.4681567Z	Brad Murray
string	string	2020-10-01T09:55:16.1905311Z	string

2. The selected user will be added as a user to the rest of the tenants with Admin role which means, a record is added in User table with below details:
   For each of the remaining available active tenants a record is inserted with
   - PartitionKey: UserID
   - RowKey: TenantID
   - Timestamp: Datetime when the User is made SystemAdmin
   - Name: Name of the user
   - Roles: **Admin**
   - Type: **Member**

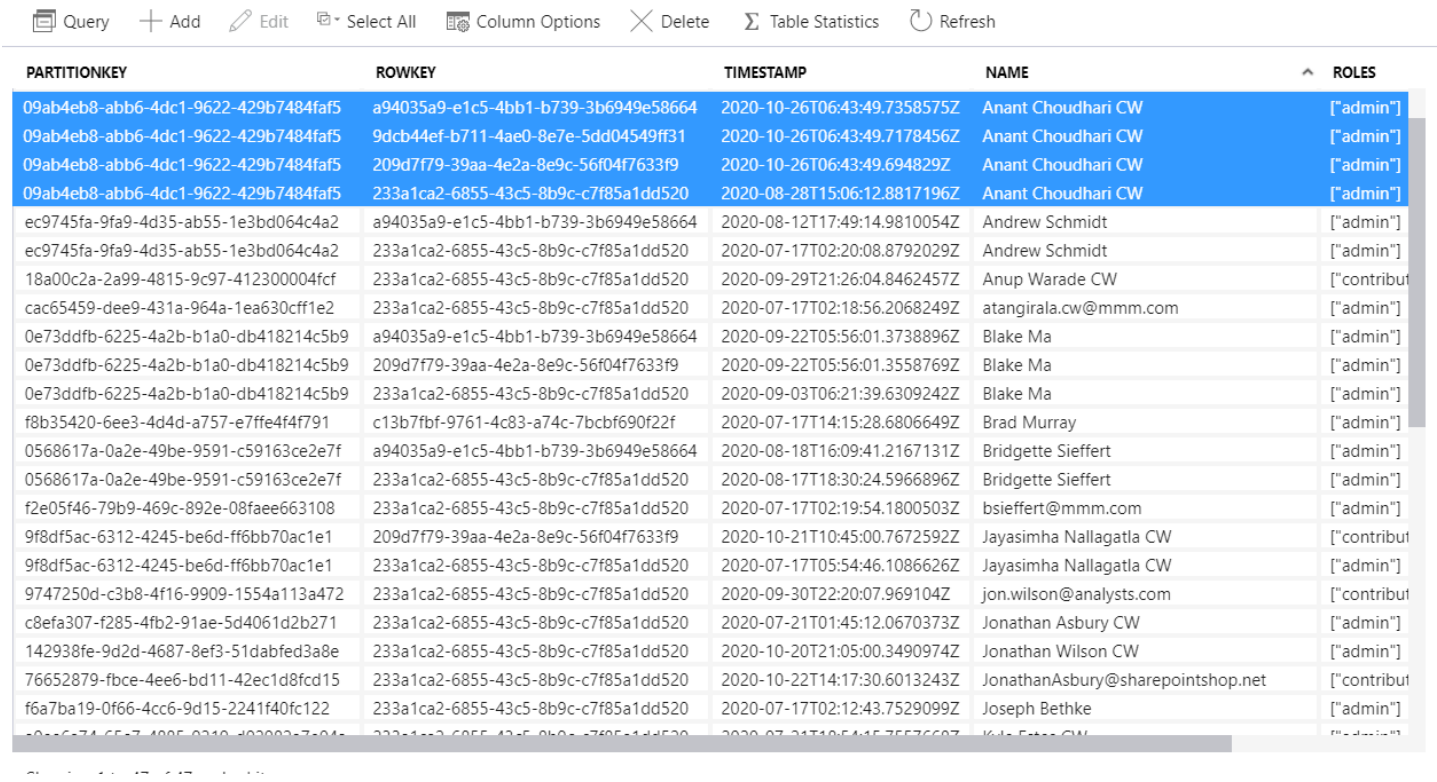
Example:
User table record before making the user as SystemAdmin:



Query | Add | Edit | Select All | Column Options | Delete | Table Statistics | Refresh

PARTITIONKEY	ROWKEY	TIMESTAMP	NAME	ROLES
09ab4eb8-abb6-4dc1-9622-429b7484faf5	233a1ca2-6855-43c5-8b9c-c7f85a1dd520	2020-08-28T15:06:12.8817196Z	Anant Choudhari CW	["admin"]
295af21b-90af-4777-b0ba-4d7f94514679	233a1ca2-6855-43c5-8b9c-c7f85a1dd520	2020-08-18T16:37:18.4236916Z	Sean Dubiel	["admin"]
0568617a-0a2e-49be-9591-c59163ce2e7f	a94035a9-e1c5-4bb1-b739-3b6949e58664	2020-08-18T16:09:41.2167131Z	Bridgette Sieffert	["admin"]
0568617a-0a2e-49be-9591-c59163ce2e7f	233a1ca2-6855-43c5-8b9c-c7f85a1dd520	2020-08-17T18:30:24.5966896Z	Bridgette Sieffert	["admin"]
ec9745fa-9fa9-4d35-ab55-1e3bd064c4a2	a94035a9-e1c5-4bb1-b739-3b6949e58664	2020-08-12T17:49:14.9810054Z	Andrew Schmidt	["admin"]
b3bbea27-2e63-4f95-8197-e9869af64b14	209d7f79-39aa-4e2a-8e9c-56f04f7633f9	2020-08-05T05:21:29.9933075Z	Sahithi Voruganti CW	["admin"]
a0ee6a74-65e7-4885-9219-d92982a7e04a	233a1ca2-6855-43c5-8b9c-c7f85a1dd520	2020-07-21T18:54:15.7557668Z	Kyle Estes CW	["admin"]
e577856f-b538-4f12-92ea-e12fd479b830	233a1ca2-6855-43c5-8b9c-c7f85a1dd520	2020-07-21T01:48:06.3109653Z	timothy.gunter@analysts.com	["admin"]
3146a929-816c-42b8-b4d8-4ceea618c629	233a1ca2-6855-43c5-8b9c-c7f85a1dd520	2020-07-21T01:47:55.4244188Z	aditya.tangirala@ggktech.com	["admin"]
c8efa307-f285-4fb2-91ae-5d4061d2b271	233a1ca2-6855-43c5-8b9c-c7f85a1dd520	2020-07-21T01:45:12.0670373Z	Jonathan Asbury CW	["admin"]
f8b35420-6ee3-4d4d-a757-e7ffe4f4f791	c13b7fbf-9761-4c83-a74c-7bcbf690f22f	2020-07-17T14:15:28.6806649Z	Brad Murray	["admin"]
b013febd-466d-4694-8f0d-c151ad32023b	233a1ca2-6855-43c5-8b9c-c7f85a1dd520	2020-07-17T12:37:54.6945702Z	TestAccount - SystemAdmin	["admin"]
a05e1a04-31bf-4219-9c1b-b7fa7bfedbfc	233a1ca2-6855-43c5-8b9c-c7f85a1dd520	2020-07-17T12:35:36.5060264Z	TestAccount - Admin	["admin"]
9745b86e-c1b1-481a-8628-7db8dfaeedf6	233a1ca2-6855-43c5-8b9c-c7f85a1dd520	2020-07-17T12:34:43.3751401Z	TestAccount - User	["readonly
95d3c662-23ea-4e2d-8d3d-ea2448706934	233a1ca2-6855-43c5-8b9c-c7f85a1dd520	2020-07-17T11:17:04.8210526Z	95d3c662-23ea-4e2d-8d3d-ea2448706934	['admin']
3e30f301-4f57-4b3e-949a-3ff8e93870b5	209d7f79-39aa-4e2a-8e9c-56f04f7633f9	2020-07-17T08:49:06.5977473Z	Viswanadh Kotha CW	["admin"]
3e30f301-4f57-4b3e-949a-3ff8e93870b5	233a1ca2-6855-43c5-8b9c-c7f85a1dd520	2020-07-17T06:14:46.3943858Z	Viswanadh Kotha CW	["admin"]
9f8df5ac-6312-4245-be6d-ff6bb70ac1e1	233a1ca2-6855-43c5-8b9c-c7f85a1dd520	2020-07-17T05:54:46.1086626Z	Jayasimha Nallagatla CW	["admin"]
b3bbea27-2e63-4f95-8197-e9869af64b14	233a1ca2-6855-43c5-8b9c-c7f85a1dd520	2020-07-17T04:42:28.8167882Z	Sahithi Voruganti CW	["admin"]
ec9745fa-9fa9-4d35-ab55-1e3bd064c4a2	233a1ca2-6855-43c5-8b9c-c7f85a1dd520	2020-07-17T02:20:08.8792029Z	Andrew Schmidt	["admin"]
f2e05f46-79b9-469c-892e-08faee663108	233a1ca2-6855-43c5-8b9c-c7f85a1dd520	2020-07-17T02:19:54.1800503Z	bsieffert@mmm.com	["admin"]
cac65459-dee9-431a-964a-1ea630cff1e2	233a1ca2-6855-43c5-8b9c-c7f85a1dd520	2020-07-17T02:18:56.2068249Z	atangirala.cw@mmm.com	["admin"]

User table record after making the user as SystemAdmin:



Query | Add | Edit | Select All | Column Options | Delete | Table Statistics | Refresh

PARTITIONKEY	ROWKEY	TIMESTAMP	NAME	ROLES
09ab4eb8-abb6-4dc1-9622-429b7484faf5	a94035a9-e1c5-4bb1-b739-3b6949e58664	2020-10-26T06:43:49.7358575Z	Anant Choudhari CW	["admin"]
09ab4eb8-abb6-4dc1-9622-429b7484faf5	9dcb44ef-b711-4ae0-8e7e-5dd04549ff31	2020-10-26T06:43:49.7178456Z	Anant Choudhari CW	["admin"]
09ab4eb8-abb6-4dc1-9622-429b7484faf5	209d7f79-39aa-4e2a-8e9c-56f04f7633f9	2020-10-26T06:43:49.694829Z	Anant Choudhari CW	["admin"]
09ab4eb8-abb6-4dc1-9622-429b7484faf5	233a1ca2-6855-43c5-8b9c-c7f85a1dd520	2020-08-28T15:06:12.8817196Z	Anant Choudhari CW	["admin"]
ec9745fa-9fa9-4d35-ab55-1e3bd064c4a2	a94035a9-e1c5-4bb1-b739-3b6949e58664	2020-08-12T17:49:14.9810054Z	Andrew Schmidt	["admin"]
ec9745fa-9fa9-4d35-ab55-1e3bd064c4a2	233a1ca2-6855-43c5-8b9c-c7f85a1dd520	2020-07-17T02:20:08.8792029Z	Andrew Schmidt	["admin"]
18a00c2a-2a99-4815-9c97-412300004fcf	233a1ca2-6855-43c5-8b9c-c7f85a1dd520	2020-09-29T21:26:04.8462457Z	Anup Warade CW	["contribut
cac65459-dee9-431a-964a-1ea630cff1e2	233a1ca2-6855-43c5-8b9c-c7f85a1dd520	2020-07-17T02:18:56.2068249Z	atangirala.cw@mmm.com	["admin"]
0e73ddfb-6225-4a2b-b1a0-db418214c5b9	a94035a9-e1c5-4bb1-b739-3b6949e58664	2020-09-22T05:56:01.3738896Z	Blake Ma	["admin"]
0e73ddfb-6225-4a2b-b1a0-db418214c5b9	209d7f79-39aa-4e2a-8e9c-56f04f7633f9	2020-09-22T05:56:01.3558769Z	Blake Ma	["admin"]
0e73ddfb-6225-4a2b-b1a0-db418214c5b9	233a1ca2-6855-43c5-8b9c-c7f85a1dd520	2020-09-03T06:21:39.6309242Z	Blake Ma	["admin"]
f8b35420-6ee3-4d4d-a757-e7ffe4f4f791	c13b7fbf-9761-4c83-a74c-7bcbf690f22f	2020-07-17T14:15:28.6806649Z	Brad Murray	["admin"]
0568617a-0a2e-49be-9591-c59163ce2e7f	a94035a9-e1c5-4bb1-b739-3b6949e58664	2020-08-18T16:09:41.2167131Z	Bridgette Sieffert	["admin"]
0568617a-0a2e-49be-9591-c59163ce2e7f	233a1ca2-6855-43c5-8b9c-c7f85a1dd520	2020-08-17T18:30:24.5966896Z	Bridgette Sieffert	["admin"]
f2e05f46-79b9-469c-892e-08faee663108	233a1ca2-6855-43c5-8b9c-c7f85a1dd520	2020-07-17T02:19:54.1800503Z	bsieffert@mmm.com	["admin"]
9f8df5ac-6312-4245-be6d-ff6bb70ac1e1	209d7f79-39aa-4e2a-8e9c-56f04f7633f9	2020-10-21T10:45:00.7672592Z	Jayasimha Nallagatla CW	["contribut
9f8df5ac-6312-4245-be6d-ff6bb70ac1e1	233a1ca2-6855-43c5-8b9c-c7f85a1dd520	2020-07-17T05:54:46.1086626Z	Jayasimha Nallagatla CW	["admin"]
9747250d-c3b8-4f16-9909-1554a113a472	233a1ca2-6855-43c5-8b9c-c7f85a1dd520	2020-09-30T22:20:07.969104Z	jon.wilson@analysts.com	["contribut
c8efa307-f285-4fb2-91ae-5d4061d2b271	233a1ca2-6855-43c5-8b9c-c7f85a1dd520	2020-07-21T01:45:12.0670373Z	Jonathan Asbury CW	["admin"]
142938fe-9d2d-4687-8ef3-51dabfed3a8e	233a1ca2-6855-43c5-8b9c-c7f85a1dd520	2020-10-20T21:05:00.3490974Z	Jonathan Wilson CW	["admin"]
76652879-fbce-4ee6-bd11-42ec1d8fcd15	233a1ca2-6855-43c5-8b9c-c7f85a1dd520	2020-10-22T14:17:30.6013243Z	JonathanAsbury@sharepointshop.net	["contribut
f6a7ba19-0f66-4cc6-9d15-2241f40fc122	233a1ca2-6855-43c5-8b9c-c7f85a1dd520	2020-07-17T02:12:43.7529099Z	Joseph Bethke	["admin"]

Pipeline

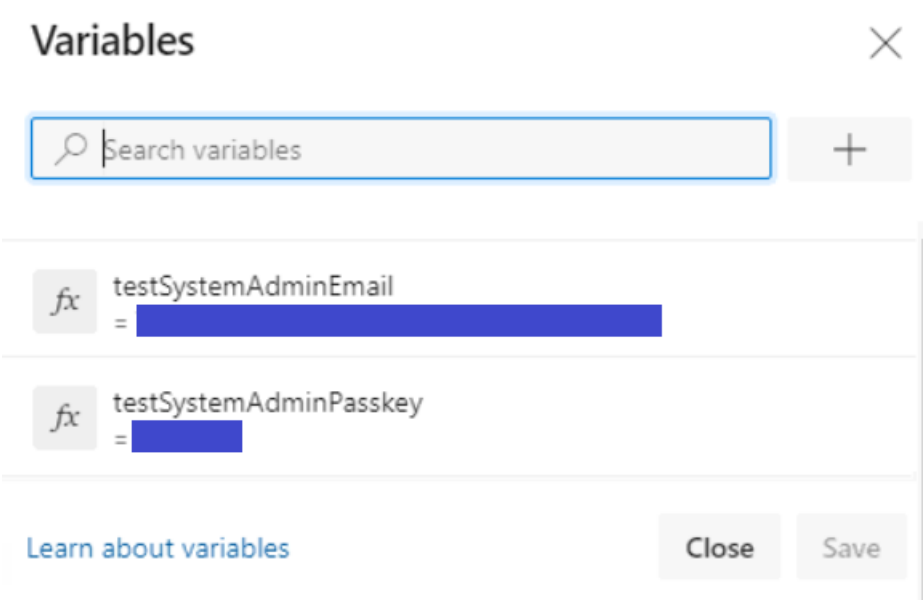
During the system deployment a Default tenant is added to the system and a user is configured as a SystemAdmin to this tenant. As this happens during the deployment, this is a one-time assignment. Below are the details on how a SystemAdmin can be added through pipelines

Configure credentials

The user credentials for SystemAdmin user are configured in the pipeline variables. Below are the details configured:

1. Email Address

2. Password



YAML file

Below are the details on how SystemAdmin is added through pipeline:

1. Define variables
   - **UserId:** Generate and assign a dedicated UserID in the variables section
   - **User table:** User storage table name
   - **SystemAdmin table:** SystemAdmin storage table name

```
108     - name: testAccountSysAdmin
109       value: [redacted]
110     - name: usersTableName
111       value: user
112     - name: sysAdminTableName
113       value: SystemAdmin
```

2. Add TestAccount details to App Configuration(Optional)
   SystemAdmin EmailID and password from pipeline variables can be added to App Configuration for future use. Below are the App Config keys currently used:
   - TestUserCredentials:SystemAdmin:UserEmail
   - TestUserCredentials:SystemAdmin:Password

```
az appconfig kv set --key TestUserCredentials:SystemAdmin:Password --value $(testSystemAdminPasskey) --name $(appConfigurationName)  --yes
az appconfig kv set --key TestUserCredentials:SystemAdmin:UserEmail --value $(testSystemAdminEmail) --name $(appConfigurationName)  --yes
```

3. Verify and add Test Account in the User Table
   The user is added in below tables:
   - User
   - SystemAdmin

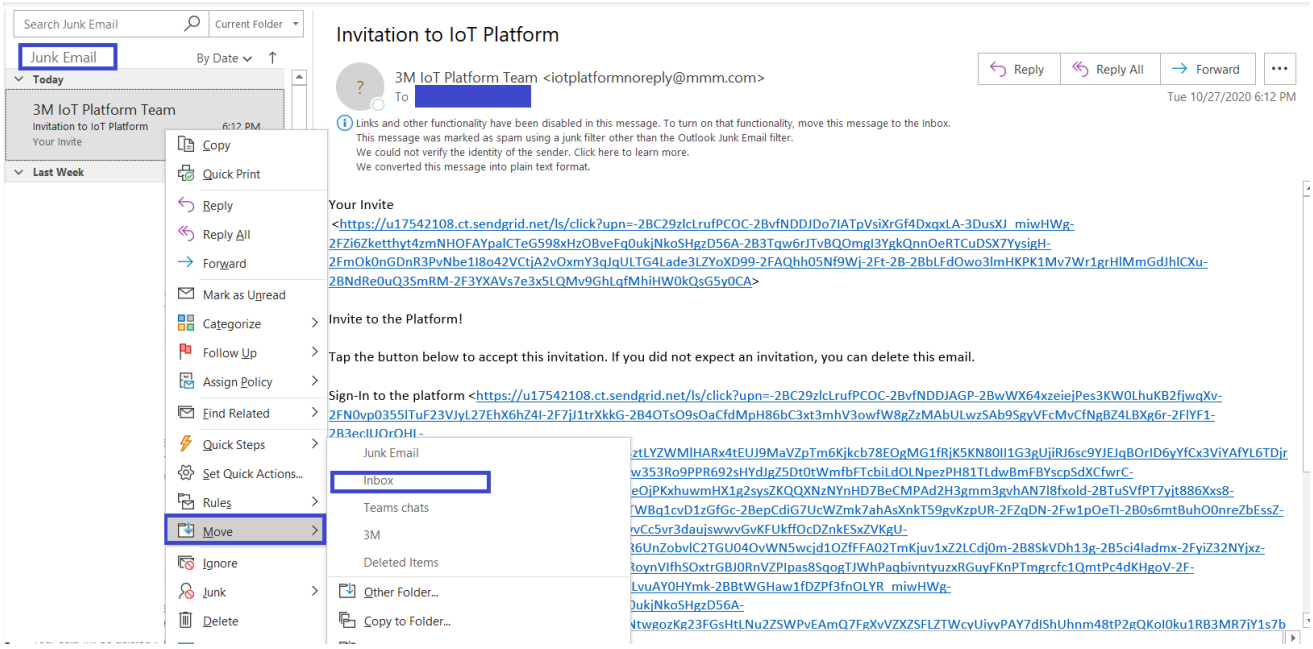
```
Install-Module AzTable -force
$table = Get-AzTableTable -resourceGroup $(resourceGroupName) -tableName $(usersTableName) -storageAccountName $(storageAccountName)
$saTable = Get-AzTableTable -resourceGroup $(resourceGroupName) -tableName $(sysAdminTableName) -storageAccountName $(storageAccountName)
$datetimevariable = Get-Date -Format "yyyy-MM-ddTHH:mm:ss:fffffffZ"
$testSystemAdmin = Get-AzTableRowByPartitionKeyRowKey -Table $table -PartitionKey $(testAccountSysAdmin) -RowKey $(serenityDevTenantId)
$saTestSystemAdmin = Get-AzTableRowByPartitionKey -Table $saTable -PartitionKey $(testAccountSysAdmin)

if(($testSystemAdmin -eq $Null) -or ($testSystemAdmin.Roles -ne '["admin"]')){
    Add-AzTableRow `
    -table $table `
    -partitionKey $(testAccountSysAdmin) `
    -rowKey $(serenityDevTenantId) -property @{"Timestamp"=$datetimevariable;"Roles"='["admin"]';"Name"="TestAccount - SystemAdmin";"Type"="Member"}
}
if($saTestSystemAdmin -eq $Null){
    Add-AzTableRow `
    -table $saTable `
    -partitionKey $(testAccountSysAdmin) `
    -rowKey $(testAccountSysAdmin) -property @{"Timestamp"=$datetimevariable;"Name"="TestAccount - SystemAdmin";"Type"="Member"}
}
```

External User Registration

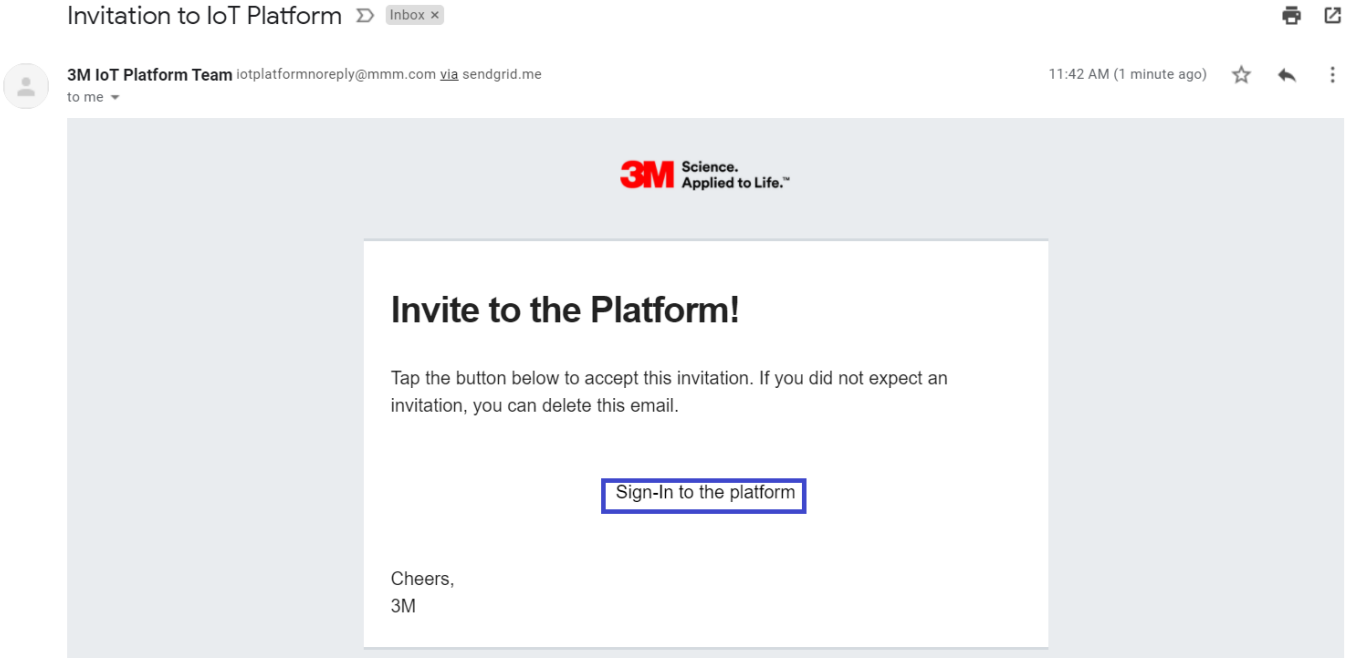
When a user is invited to a tenant, they will receive an email that must be used to complete the registration process so they can sign in. This document guides the invited user through the registration process so that they can access the intended resource, which might be the Serenity UI or a custom application built on top of Serenity.

User Registration

- Find the email invite
    - An email invite is sent to your mail address. Check for it in your inbox.
    - If the email invite is not found in your inbox, see if it ended up in the Junk folder
    - Once found, right-click on the email and select Move -> Inbox(any other folder of your choice). This allows enabling the links in the email.

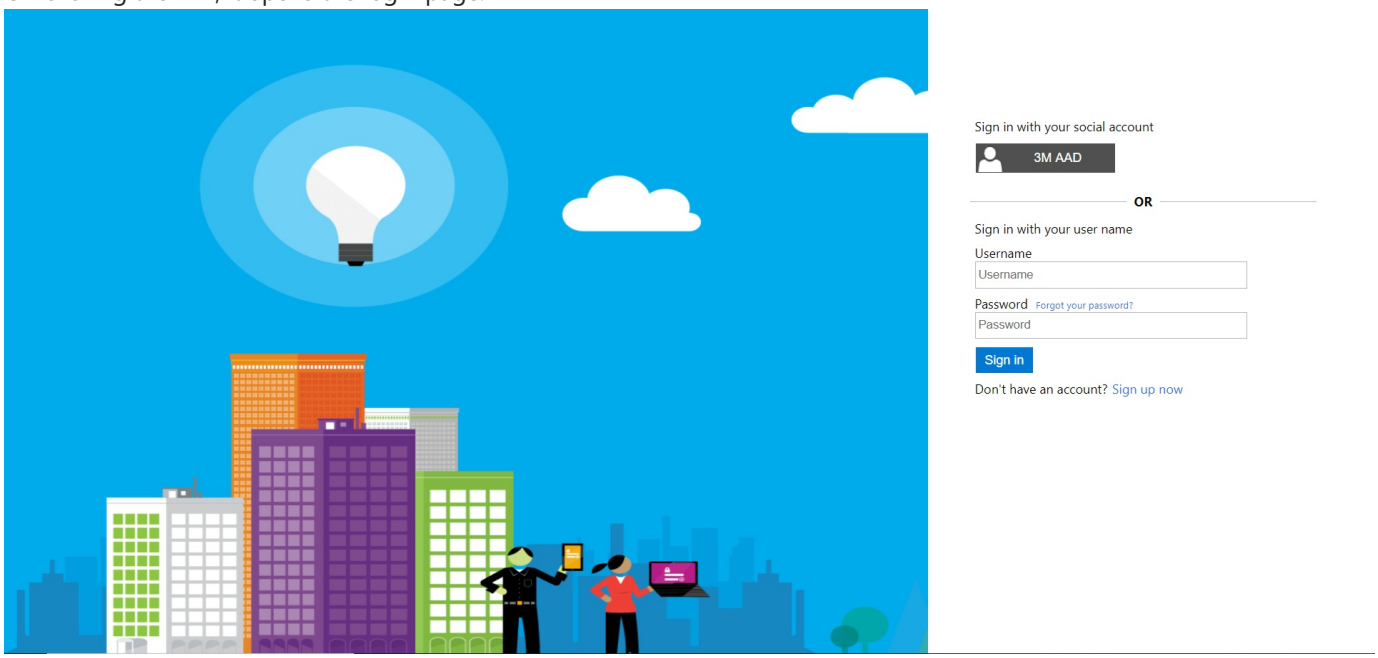


- Next, open the mail and click **Sign-In to the platform**.

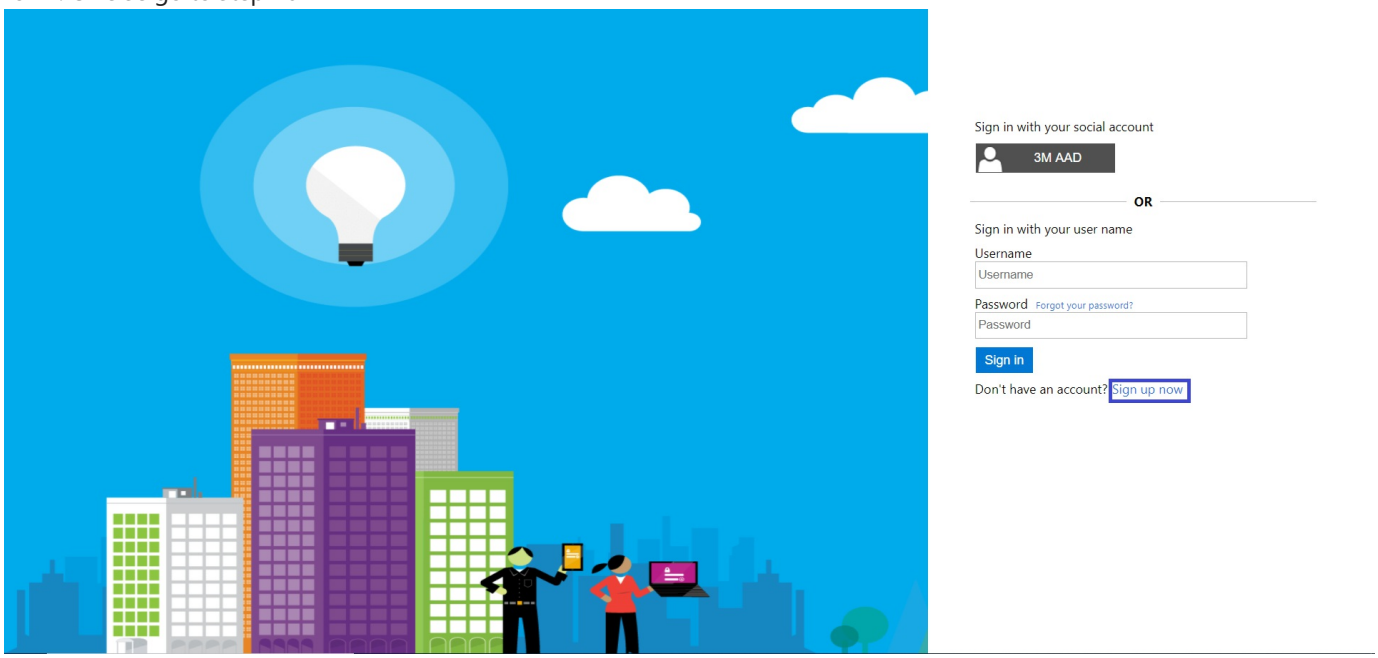


> **Note**
>
> If on click of the link does not open the Microsoft Authentication login page, copy the link from the email and try it in the web browser incognito tab.

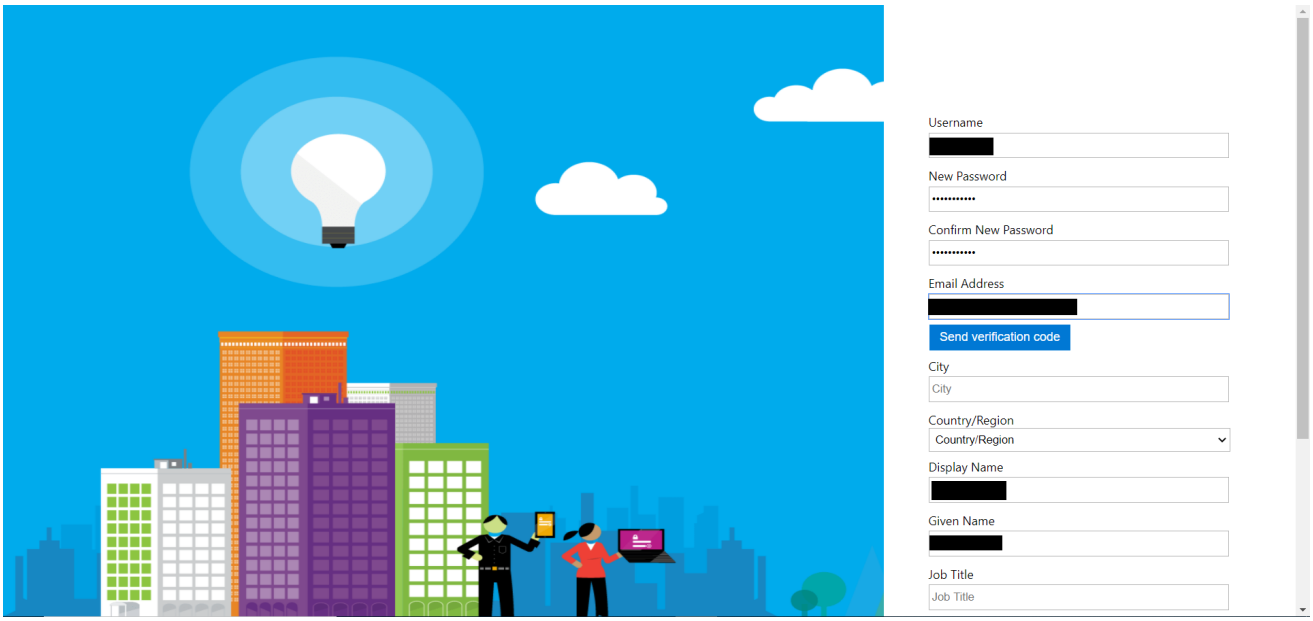
- On clicking the link, it opens the login page.



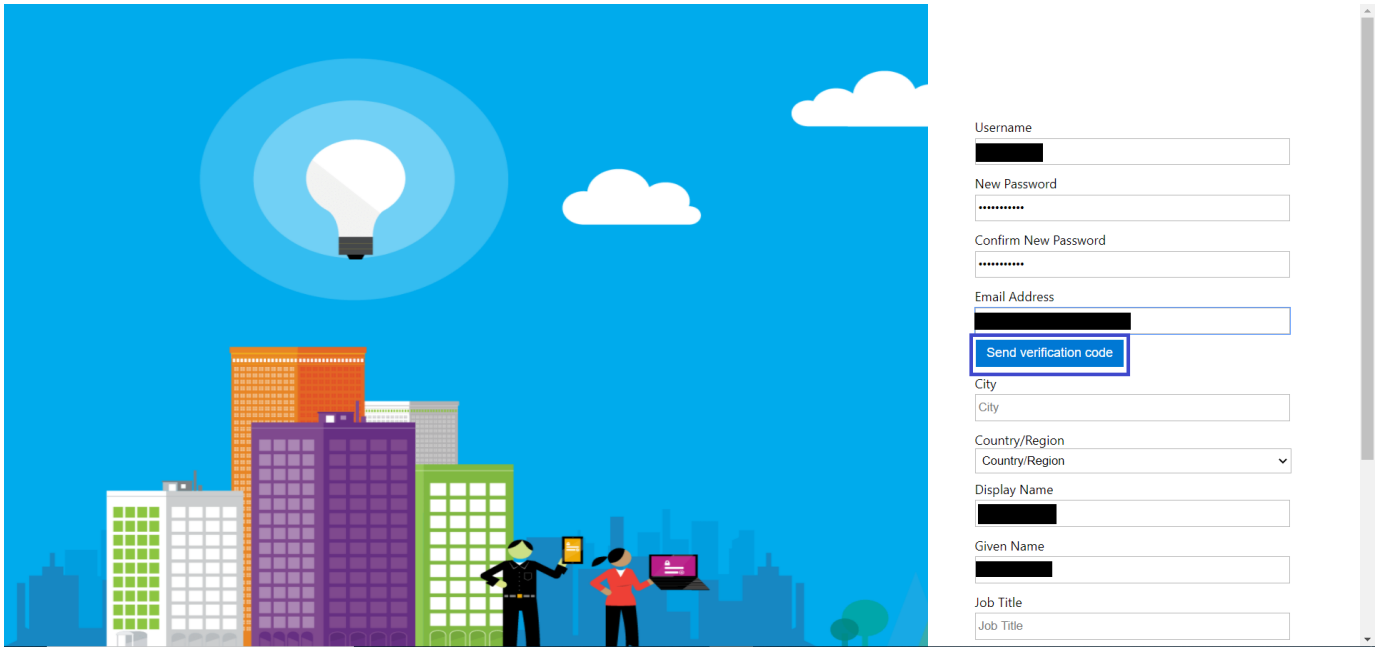
- If you don't have an account and not registered to the system previously, click on **Sign up now**, it opens the registration form. Or else go to step 10

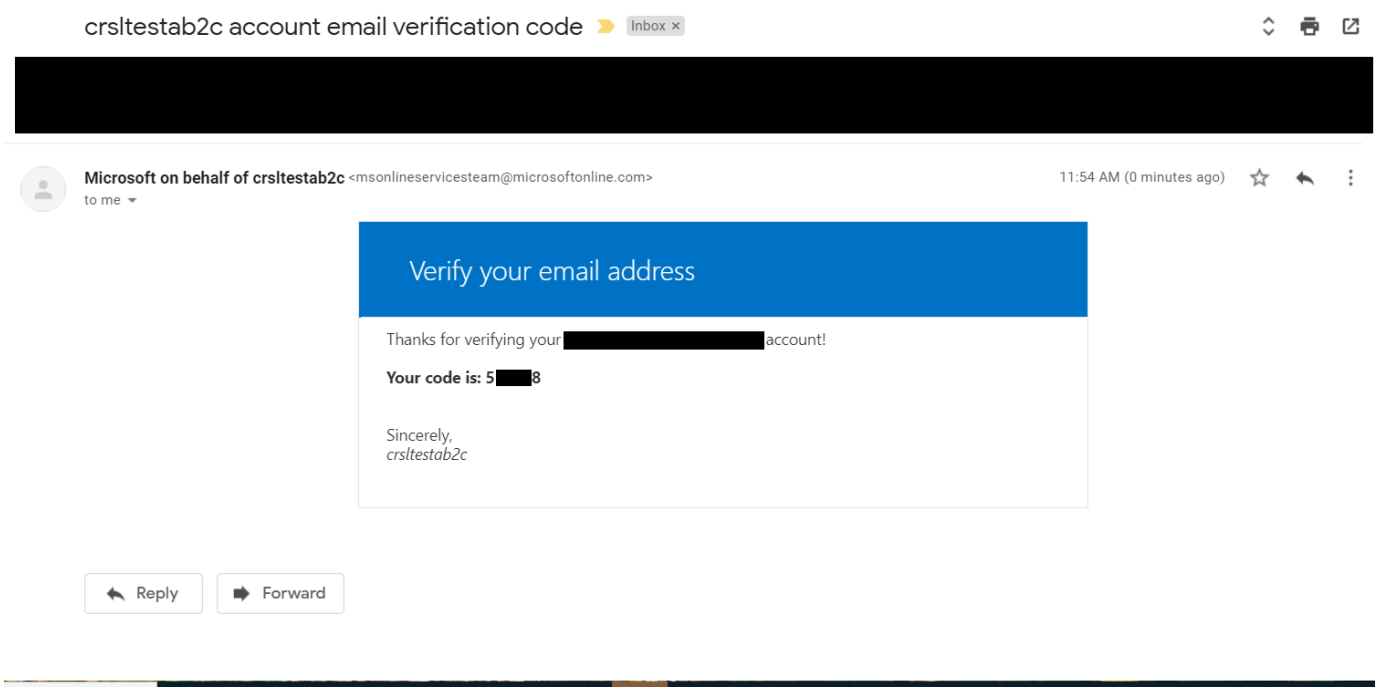


- Fill the below the required details in registration form
    - Username
    - New Password
    - Confirm New Password
    - Email Address
    - Display Name
    - Given Name

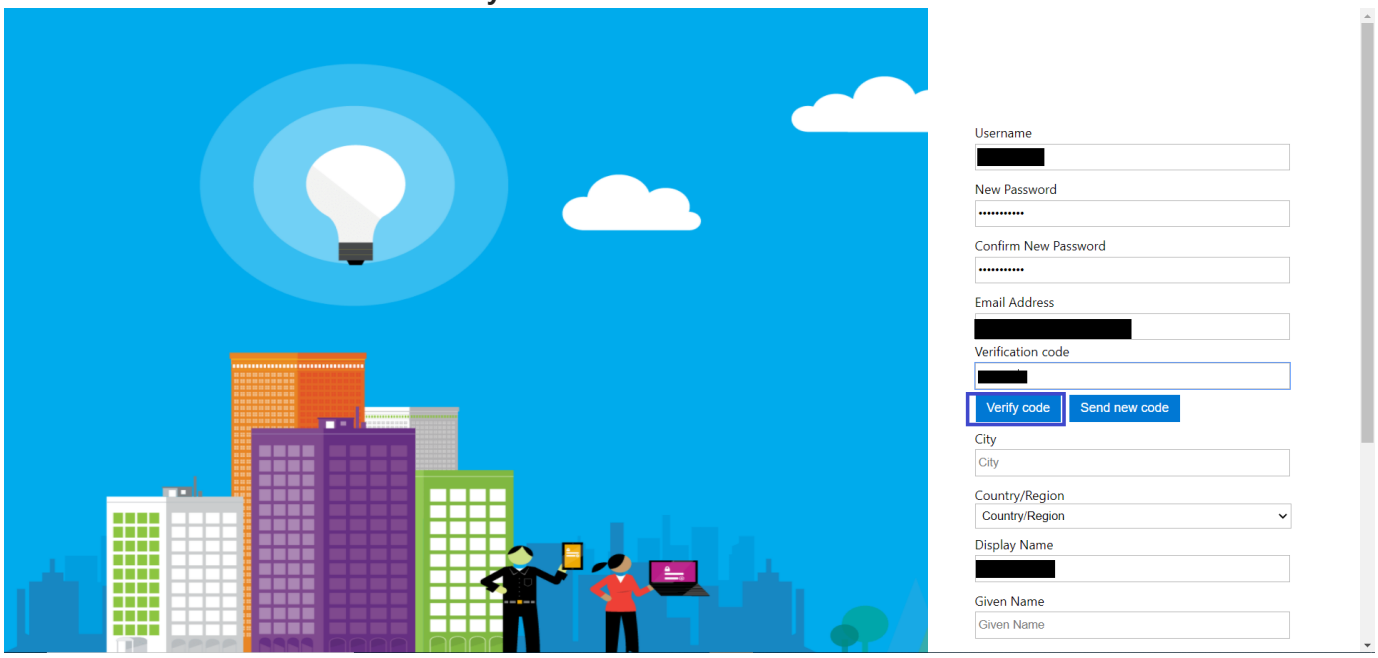


- After filling in the details varify the email by clicking **Send verification code**, an email will be sent to the mentioned email address with an OPT.

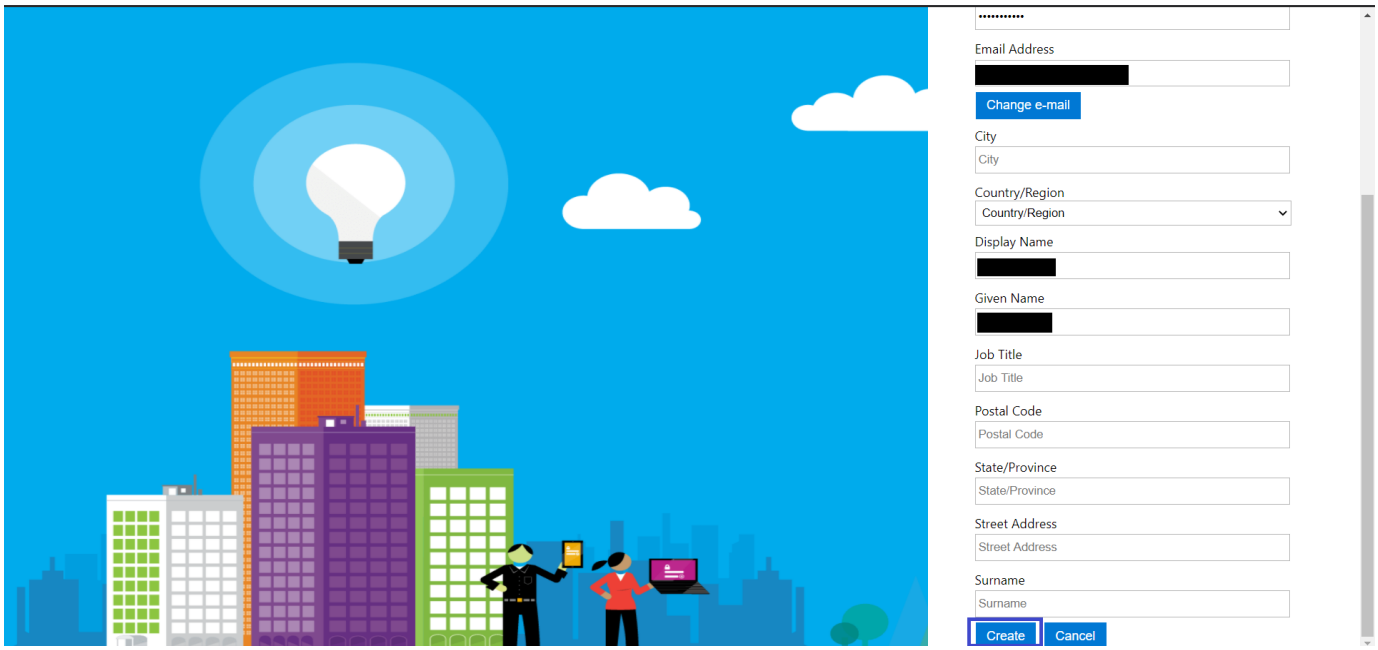




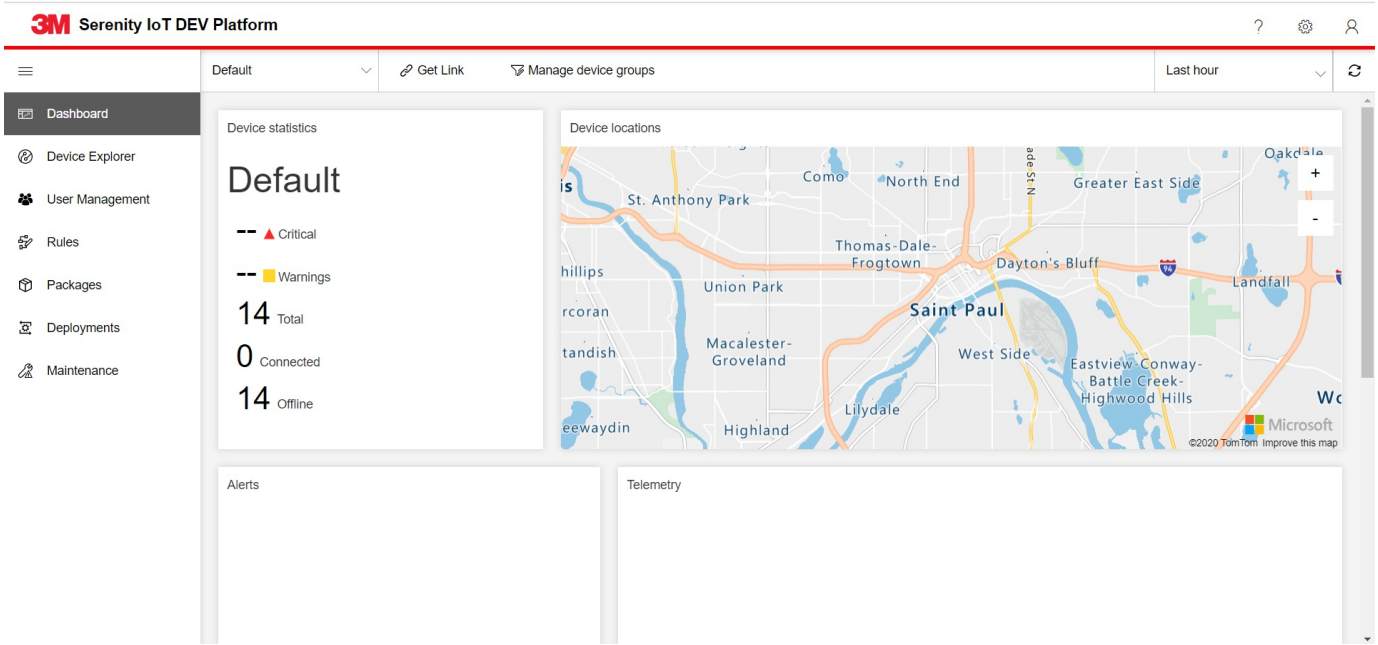
- Provide the verification code and click **Verify code**



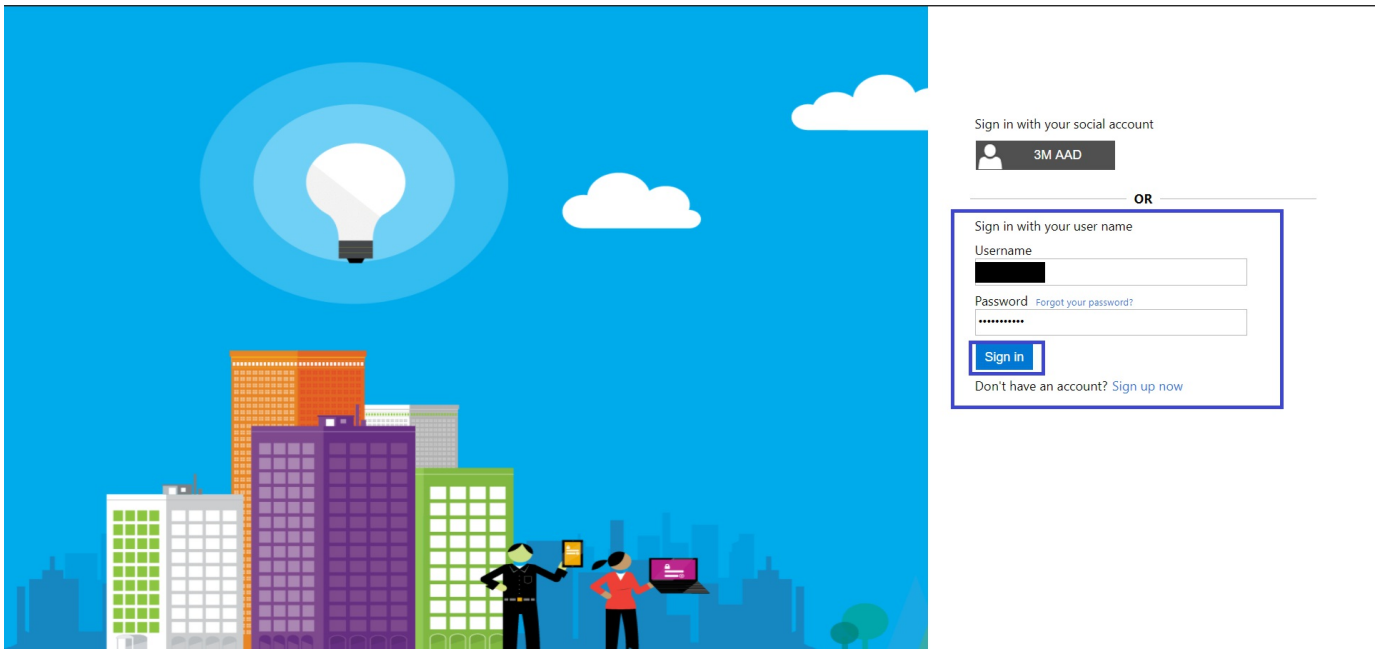
- Once the email is verified, click **Create** to register an account.



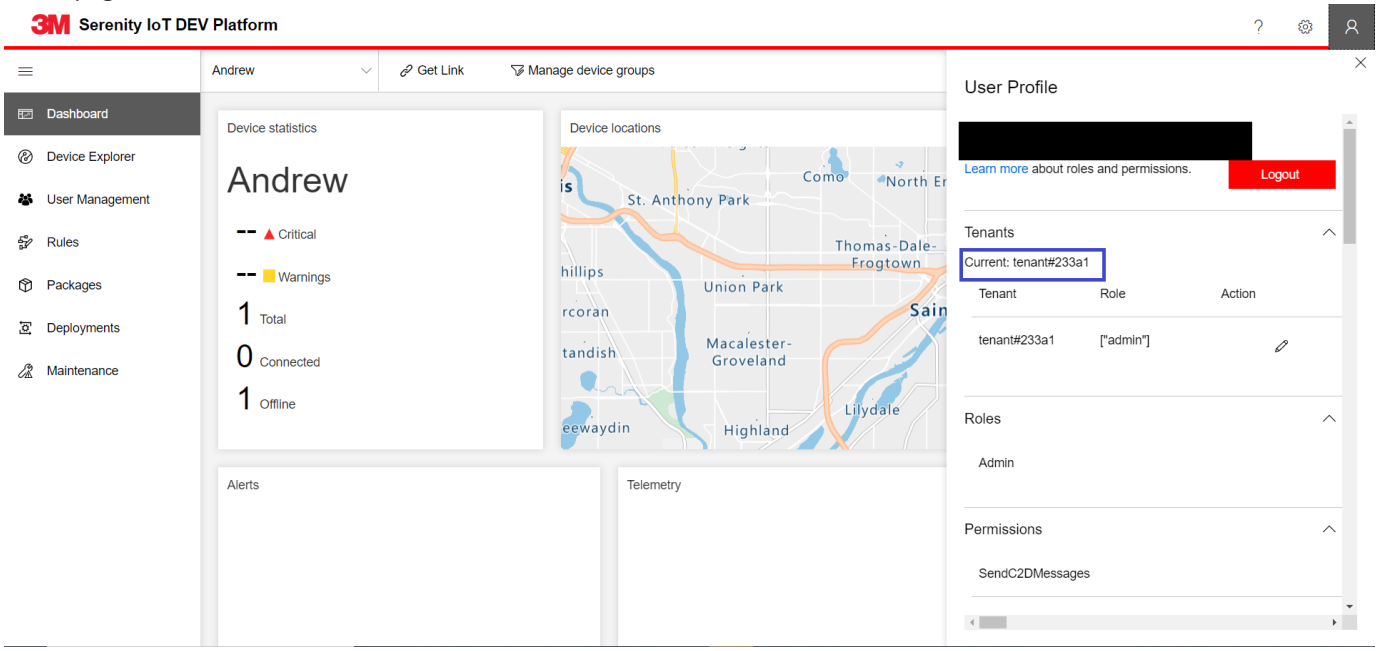
- After the user account is successfully registered, it will redirect to the 3M Remort Monitoring System.



- As you are already registered, use **Sign in with your user name** section on the login page to login into the system. Provide your login details and click **Sign in**.



- Once successful login into the system, verify the tenant details by click on **User Profile** icon which is at the top right corner of the page.



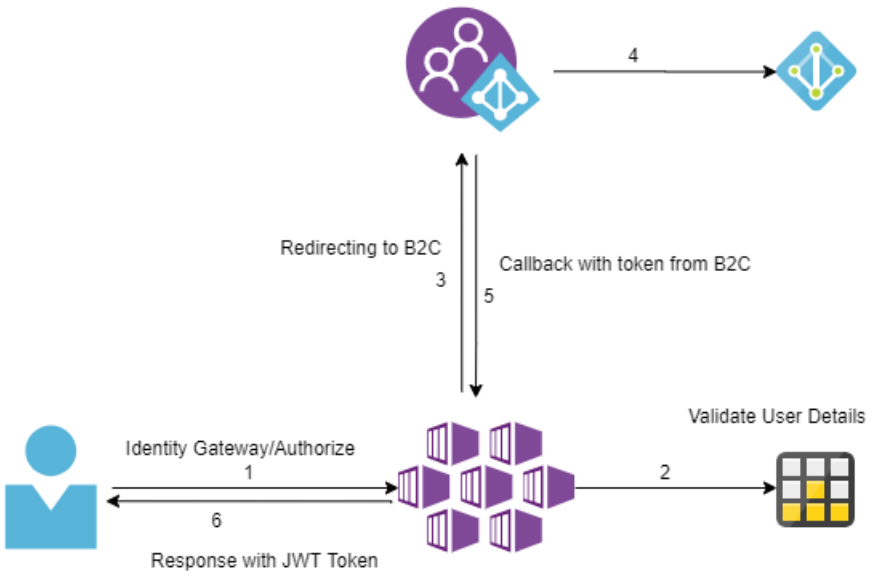
- With this 3M user is successfully registered to a tenant.

Authentication Architecture

This documents explains the flow of the authentication in Serenity platform.

The authentication and authorization microservice manages the users authorized to access the solution. User management can be done using any identity service provider that supports OpenId Connect.

Below is the flow diagram of authentication.



Source: Above diagram is created using the website "https://app.diagrams.net/". We have placed the xml version of the diagram (authentication-architecture.xml) in artifacts folder, and we can update the diagram by importing when we do any modifications to the authentication architecture.

Description

Currently Serenity Platform is using 3M Azure AD to authenticate the Users for the applications. Azure B2C uses external Identity providers to authenticate users.

Below are the steps as part of authentication in Serenity.

1. When user tries to access our application, a request is being sent to Identity gateway service to authenticate the request.
2. Identity gateway validates the user details from the system
3. Redirects the request to AD B2C
4. AD B2C validates request as Identity Provider to validate in AD
5. Returns callback url with the token from B2C.
6. Identity Gateway Service accepts the callback url and sends back the JWT Token to the user.

Serenity platform provides two ways of authentication.

1. Using 3M AAD
    - This Sign-in mechanism is used by the users who have 3M AD accounts.

2. Sign up and Sign-In
    - This mechanism can be used by all the users. For the first time users will need to use the sign up which generates a B2C Active Directory Account. After generating the user account we can use the sign in option to the system.

Authorization Architecture

This documents explains the flow of the authorization in Serenity platform.

The authentication and authorization microservice manages the users authorized to access the solution. User management can be done using any identity service provider that supports OpenId Connect.

What is a JWT?

A JWT (JSON Web Token) is a compact, URL-safe means of transferring information that contains JSON objects encoded and serialized for transmission.

How is this used?

We can get user information like name and roles values from the JWT token. In this implementation, we use Azure AAD tokens.

The claims are added to the AAD application manifest so that the token contains the desired information. You can learn more about optional token claims here.

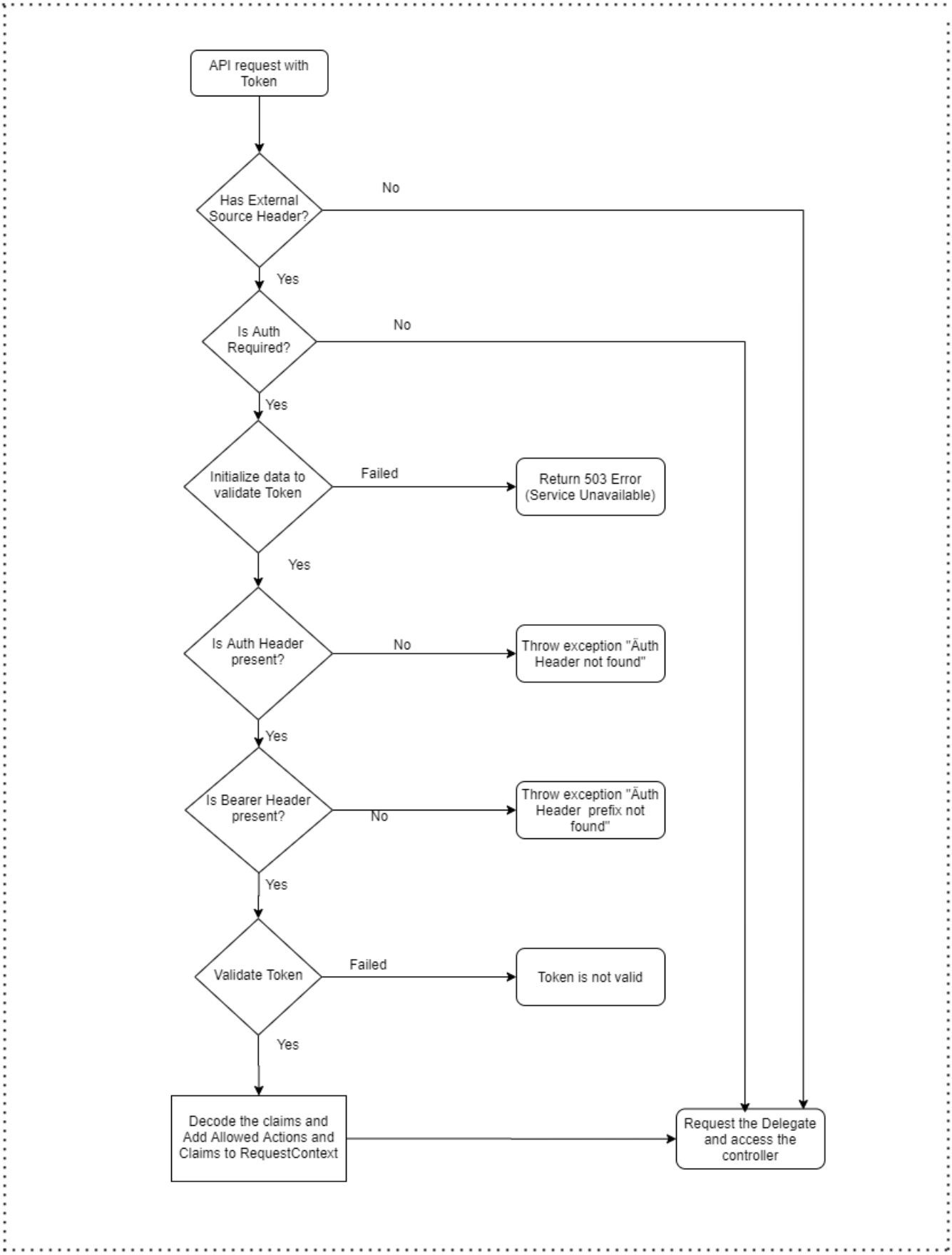
We can also get access token for resources by using application's credential so that Azure resource management task can be performed. The current user who has the role which includes 'AcquireToken' will be able to get this token.

Authorization Logic

In Serenity, we are handling authorization as part of AuthMiddleware for the every request in the api.

As part of authorization, we will be verifying whether the request has AuthHeader, Bearer header, auth token present and send the response validating the auth token either a success or an exception message appropriately which needs to be handled in the respective source.

Below is the flow diagram for the Authorization.


API request with Token
Has External Source Header?
No
Yes
Is Auth Required?
No
Yes
Initialize data to validate Token
Failed
Return 503 Error (Service Unavailable)
Yes
Is Auth Header present?
No
Throw exception "Äuth Header not found"
Yes
Is Bearer Header present?
No
Throw exception "Äuth Header prefix not found"
Yes
Validate Token
Failed
Token is not valid
Yes
Decode the claims and Add Allowed Actions and Claims to RequestContext
Request the Delegate and access the controller

Permissions Architecture

This document has the technical details of permissions that are being handled in the Serenity platform.

- For each and every environment, we are storing the roles and its associated permissions in **AppConfiguration**. One needs to have access to this in order to see the list of available roles and permissions.

Web Layer

A generic **Protected** component is built and being re-used all over the application.

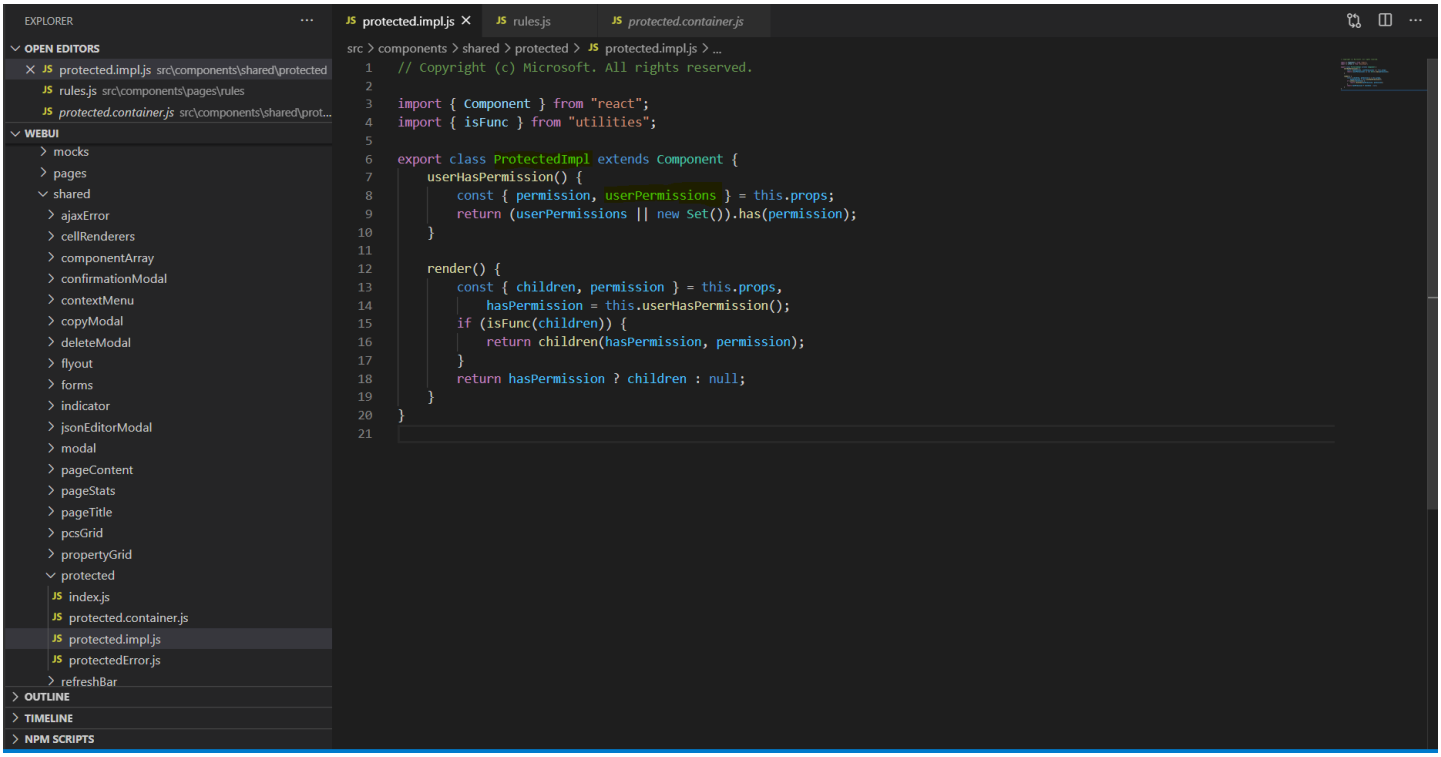
Here is the sample

```
<Protected permission={permissions.createPackages}>
    <Btn
        svg={svgs.plus}
        onClick={this.openNewPackageFlyout}
    >
        {t("packages.new")}
    </Btn>
</Protected>
```

Implementation Details

- As part of each deployment, we read the data from App Configuration and store them in a json file namely **Policies.js**.
- After deployment, when we load the application we read the data from policies.js and fetch the associated permissions based on the loggedIn user role.
- Then we store the data in the redux store and compare with the input permission through the **Protected** component.
- If the comparison is true then the associated component is loaded or else null.

Source: Above implementation code is present in **Protected** component(src\components\shared\protected\protected.impl.js)



API Layer

For all the API methods, we are adding authorize attribute by passing the respective permission. As part of this authorize filter attribute implementation we are validating the permission passed with the permissions associated with the loggedIn user role and respond based on the validation.

Below is the screenshot of the **implementation**


```
1 reference | Kyle Estes, 145 days ago | 1 author, 1 change
private bool IsValidAuthorization(HttpContext httpContext, string allowedAction)
{
    if (!httpContext.Request.GetAuthRequired() || !httpContext.Request.IsExternalRequest())
    {
        return true;
    }

    if (allowedAction == null || !allowedAction.Any())
    {
        return true;
    }

    var userAllowedActions = httpContext.Request.GetCurrentUserAllowedActions();
    if (userAllowedActions == null || !userAllowedActions.Any())
    {
        return false;
    }

    // validation succeeds if any required action occurs in the current user's allowed allowedAction
    return userAllowedActions.Select(a => a.ToLowerInvariant())
       .Contains(this.allowedAction.ToLowerInvariant());
}
```

Sample

```
[HttpPost]
[Authorize("CreatePackages")]
1 reference | Kyle Estes, 145 days ago | 1 author, 1 change
public async Task<PackageApiModel> PostAsync(string packageName, string packageType, string configType, string version, List<string> tags, IFormFile package)...
```

This document explains the current token format being used in Serenity Platform.

Currently as part of Serenity platform we are using **JWT Token**

JWT Token JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. JWTs can be signed using a secret (with the HMAC algorithm) or a public/private key pair using RSA or ECDSA.

Although JWTs can be encrypted to also provide secrecy between parties, we will focus on signed tokens. Signed tokens can verify the integrity of the claims contained within it, while encrypted tokens hide those claims from other parties. When tokens are signed using public/private key pairs, the signature also certifies that only the party holding the private key is the one that signed it.

When should you use JSON Web Tokens?

- **Authorization:** This is the most common scenario for using JWT. Once the user is logged in, each subsequent request will include the JWT, allowing the user to access routes, services, and resources that are permitted with that token.
- **Information Exchange:** JSON Web Tokens are a good way of securely transmitting information between parties.

JSON Web Token Structure

- JSON Web Tokens consist of three parts separated by dots (.), which are:
    - **Header**
        - The header typically consists of two parts: the type of the token, which is JWT, and the signing algorithm being used, such as HMAC SHA256 or RSA.
    - **Payload**
        - The second part of the token is the payload, which contains the claims. Claims are statements about an entity (typically, the user) and additional data.There are three types of claims: registered, public, and private claims.
            - **Registered claims:** These are a set of predefined claims which are not mandatory but recommended, to provide a set of useful, interoperable claims. Some of them are: iss (issuer), exp (expiration time), sub (subject), aud (audience), and others.
            - **Public claims:** These can be defined at will by those using JWTs. But to avoid collisions they should be defined in the IANA JSON Web Token Registry or be defined as a URI that contains a collision resistant namespace.
            - **Private claims:** These are the custom claims created to share information between parties that agree on using them and are neither registered or public claims.
                - **Signature**
                - To create the signature part you have to take the encoded header, the encoded payload, a secret, the algorithm specified in the header, and sign that.
- JWT looks like the following pattern. **xxxxx.yyyyy.zzzzz**

If you want to play with JWT and put these concepts into practice, you can use jwt.io Debugger(https://jwt.io/#debugger-io) to decode, verify, and generate JWTs.

Benefits of JWT Token

As JSON is less verbose than XML, when it is encoded its size is also smaller, making JWT more compact than SAML. This makes JWT a good choice to be passed in HTML and HTTP environments.

Security-wise, SWT can only be symmetrically signed by a shared secret using the HMAC algorithm. However, JWT and SAML tokens can use a public/private key pair in the form of a X.509 certificate for signing. Signing XML with XML Digital Signature without introducing obscure security holes is very difficult when compared to the simplicity of signing JSON.

JSON parsers are common in most programming languages because they map directly to objects. Conversely, XML doesn't have a natural document-to-object mapping. This makes it easier to work with JWT than SAML assertions.

Serenity Identity Management

Serenity IoT Platform leverages **Azure AD B2C** for user authentication.

- B2C responsible for handling user account sign-up, sign-in, profile edit and password reset functionalities outside the applications developed to meet any specific functionality. AAD B2C has its own login portal management which can be customized to a certain extent to change the look and feel as required by customers.
- It is an authentication service for publicly facing applications. In addition to provide authentication service for local email accounts, it also integrates with other third-party identity providers such as Google+, Facebook, Amazon, or LinkedIn to provide a one-stop shop for authentication. It acts as an alternative to the burden of having to manage authentication and account details yourself.
- Azure AD B2C provides identity as a service for your apps by supporting two industry standard protocols: OpenID Connect and OAuth 2.0.

Serenity Identity Providers

Currently Azure AD B2C is configured with:

- OpenID configuration with 3M AAD as Identity provider.
- Local Account with User Name as Identity provider.

Additional Identity Providers

Azure AD B2C supports following list of Identity providers

- Amazon
- Facebook
- GitHub (Preview)
- GitHub
- Google
- LinkedIn
- QQ (Preview)
- Twitter
- WeChat (Preview)
- Weibo (Preview)
- Local account
- Microsoft Account
- OpenID Connect

Roles and Permissions

This document provides information on Roles, Permissions, and mapping between them.

List of Permissions

PERMISSIONS NAME	PERMISSIONS KEY	DESCRIPTION
View solution	ReadAll	Allows read only content
Update alerts	UpdateAlarms	Allows to close or acknowledge alarms
Delete alerts	DeleteAlarms	Allows deleting alarms
Create devices	CreateDevices	Allows to add a device
Update devices	UpdateDevices	Allows updating devices details
Delete devices	DeleteDevices	Allows to delete devices
Create device groups	CreateDeviceGroups	Allows to create a device group
Update device groups	UpdateDeviceGroups	Allows updating a device group
Delete device groups	DeleteDeviceGroups	Allows deleting device group
Create rules	CreateRules	Allows creating a rule
Update rules	UpdateRules	Allows updating a rule
Delete rules	DeleteRules	Allows deleting a rule
Create jobs	CreateJobs	Allows to create a device jobs
Create packages	CreatePackages	Allows to create a package
Delete packages	DeletePackages	Allows deleting a package
Create deployments	CreateDeployments	Allows to create a deployment
Delete deployments	DeleteDeployments	Allows deleting deployments
Delete tenant	DeleteTenant	□□ No longer in use, only System Admin has the rights to delete tenant
Enable alerting	EnableAlerting	Allows enabling alerting
Disable alerting	DisableAlerting	Allows to disable alerting
Invite users	InviteUsers	Allows to invite a user to system

PERMISSIONS NAME	PERMISSIONS KEY	DESCRIPTION
Delete users	DeleteUsers	Allows deleting users from system
Update SIM management	UpdateSIMManagement	Allows managing device SIM cards
AcquireToken	AcquireToken	□□ TBD
TagPackages	TagPackages	Allows to tag package
SendC2DMessages	SendC2DMessages	Allows to Send cloud to device message
UserManage	UserManage	Allows managing user setting

List of Roles

Currently, the system offers only 3 roles per tenant, which are shown below.

1. Admin
2. Contributor
3. ReadOnly

> **Note:**
>
> System Admin is not related to above mentioned roles, it is special permission for a user who can create and delete tenants, and allowed to maintain other System Admins.
>
> By default, the System Admin who creates the tenant is automatically assigned the Admin role.

Permission and Role mappings

PERMISSIONS NAME	ADMIN	CONTRIBUTOR	READ ONLY
View solution	Yes	Yes	Yes
Update alerts	Yes	No	No
Delete alerts	Yes	No	No
Create devices	Yes	Yes	No
Update devices	Yes	Yes	No
Delete devices	Yes	Yes	No
Create device groups	Yes	Yes	No
Update device groups	Yes	Yes	No
Delete device groups	Yes	Yes	No
Create rules	Yes	No	No

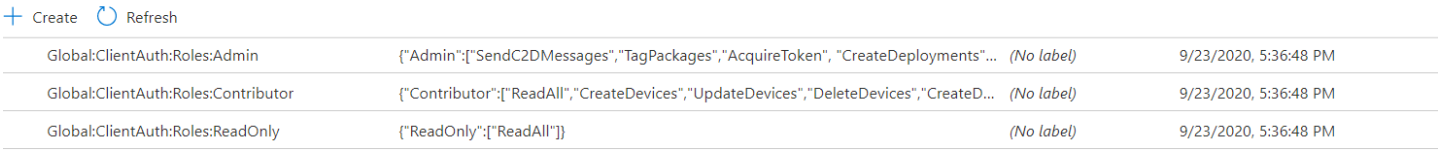
PERMISSIONS NAME	ADMIN	CONTRIBUTOR	READ ONLY
Update rules	Yes	No	No
Delete rules	Yes	No	No
Create jobs	Yes	No	No
Create packages	Yes	No	No
Delete packages	Yes	No	No
Create deployments	Yes	No	No
Delete deployments	Yes	No	No
Delete tenant	Yes	No	No
Disable alerting	Yes	No	No
Delete users	Yes	No	No
Enable alerting	Yes	No	No
Invite users	Yes	No	No
AcquireToken	Yes	No	No
TagPackages	Yes	No	No
SendC2DMessages	Yes	No	No
Update SIM management	Yes	No	No
UserManage	Yes	No	No

Custom Roles and Permissions

Roles and Permissions in Serenity are configured in Azure App Configuration. For each role, a key-value pair configured where,

- **Key** follows the format *Global:ClientAuth:Roles:{RoleName}*
- **Value** is a json with list of permissions grouped by role

| Configuration explorer



Example

For example, the configuration for Admin role is as shown below:

- **Key:** Global:ClientAuth:Roles:Admin
- **Value:**

```
{ "Admin": [ "SendC2DMessages", "TagPackages", "AcquireToken", "CreateDeployments", "CreateDeviceGroups",
"CreateDevices", "CreateJobs", "CreatePackages", "CreateRules", "DeleteAlarms", "DeleteDeployments",
"DeleteDeviceGroups", "DeleteDevices", "DeletePackages", "DeleteRules", "DeleteTenant", "DisableAlerting",
"DeleteUsers", "EnableAlerting", "InviteUsers", "ReadAll", "UpdateAlarms", "UpdateDeviceGroups",
"UpdateDevices", "UpdateRules", "UpdateSimManagement", "UserManage" ] }
```

Add New Permission

To add new permission to an existing role, add the new permission name to the json as mentioned in the example above.

Add New Role

To add new role, add new key-value pair into the App configuration as mentioned in the example above.

User Registration FootPrint

This document lists the data points and their storage while registering a User to a Serenity platform.

Below is the list of actions/operations performed while registering a new user to a Tenant:

1. Sending Invitation Email to User for a Tenant.
2. User Registration in AD B2C.
3. User Details Mapping in Storage.

Sending Invitation Email to User for a Tenant.

Sending Invitation to User is triggered by posting data to the following end point

`POST {identity-gateway-service}/v1/tenants/invite`

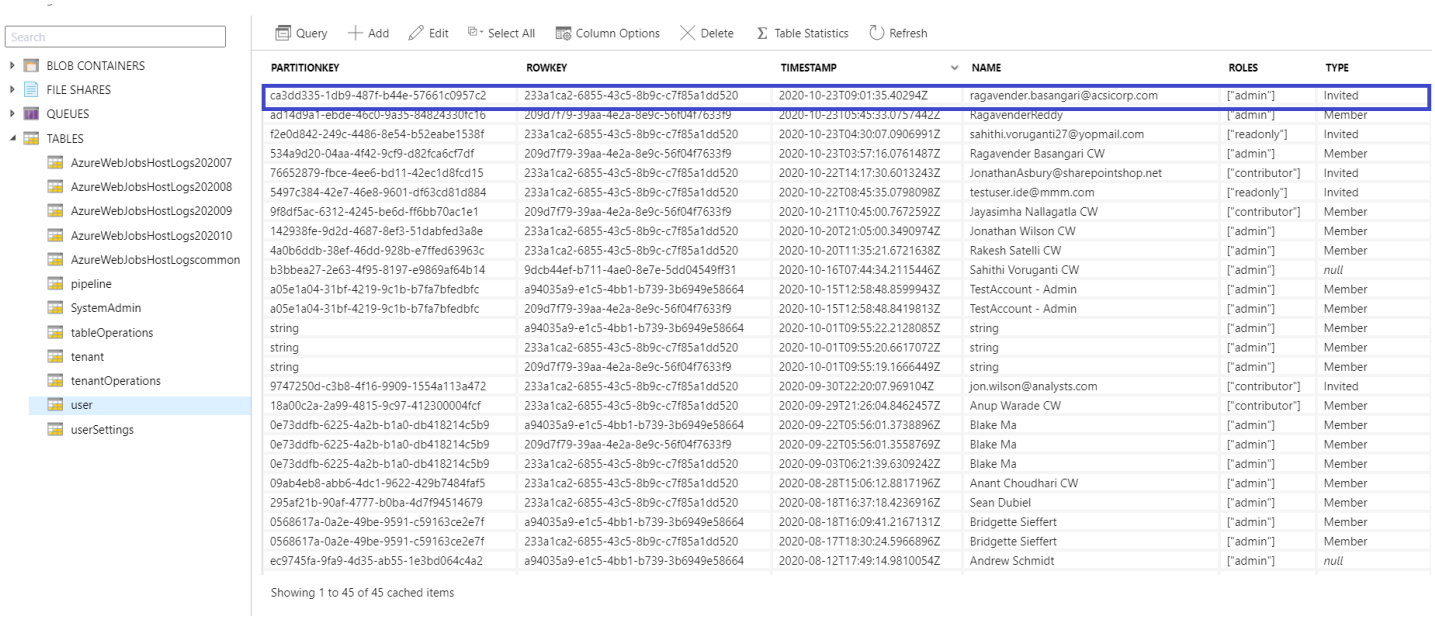
Payload

```
{
  "email_address": "testuser@gmail.com",
  "role": "admin",
  "email_subject" : "custom subject",
  "invite_uri" : "https://emdudahiotfepkg.azurewebsites.net",
  "from_email" : "testemail@email.com",
  "from_message" : "This is a custom message",
  "invite_user_meta_data : "User data info"
  "message_body" : "Welcome to the iot platform"
}
```

As part of this process, an UserTenant relationship is established using an arbitrary userid(a new GUID) that is generated, and inserted into User table in Table Storage.

Data that is inserted into User table is as follows

COLUMN	VALUE
UserId/PartitionKey	A new GUID.
Tenant/RowKey	TenantId from the token.
Name	Email address from request.
Roles	JSON Serialized string from request.
Type	Invited



If the request provides an data in **invite_user_meta_data**, then the data is inserted into the UserSettings table using the key **InviteUserMetaData**

Data that is inserted into UserSetting table is as follows

COLUMN	VALUE
UserId/PartitionKey	A new GUID.
SettingKey/RowKey	InviteUserMetaData
Value	Data from invite_user_meta_data property in request.

User Registration in AD B2C

Once User receives a Invite email, on click of the Sign-In to the Platform link following operations will be performed.

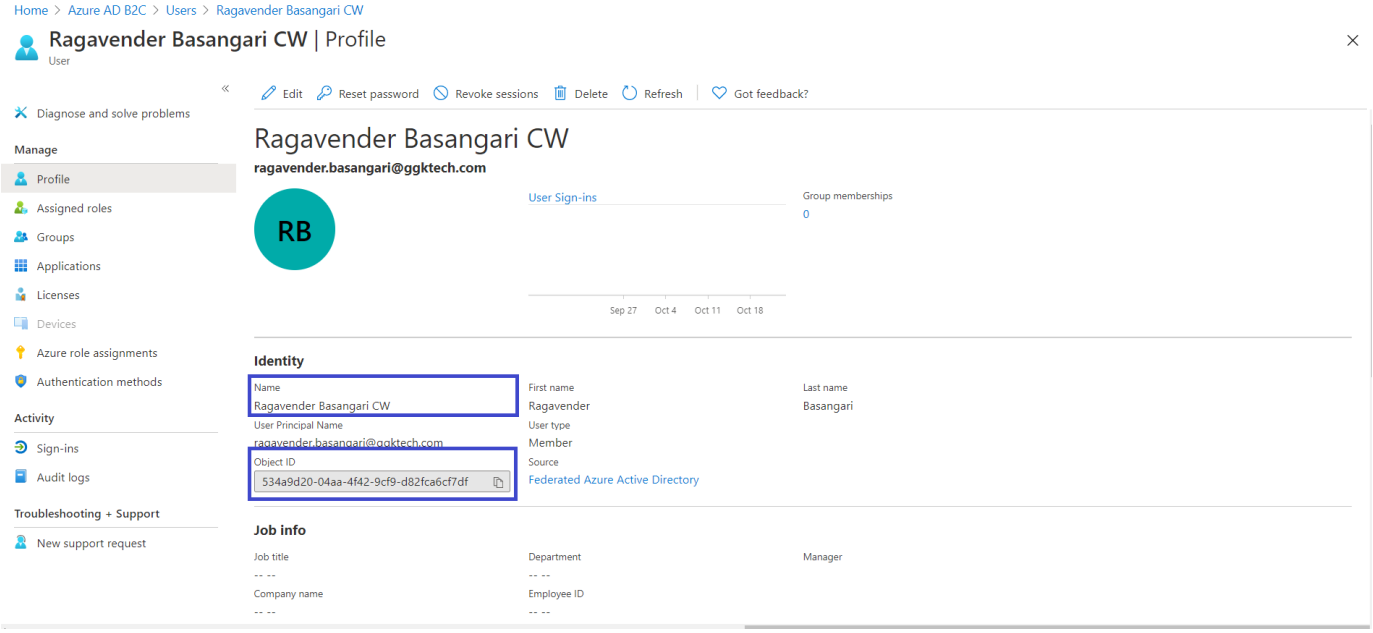
1. A call is triggered to `GET /connect/authorize` with the following query params

```
client_id: IoTPlatform
redirect_uri: https://crsliot-aks-dev.centralus.cloudapp.azure.com
state: acc7ec8aa2e145fc846c423da27486c3
nonce: ad6d71ef090d4f43b71a5133197e5d83
invite:
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJyb2xlIjoiYWRtaW4iLCJ0ZW5hbnQiOiIyMDlkN2Y3OS0zOWFhLTRlMmEtOGU5Yy01
NmYwNGY3NjMzZjkiLCJ1c2VySWQiOiIzMWRkNTBmNC1iYWVkLTQxNjUtOTUwZi1mNGM4NDQ4NGEyNjgiLCJleHAiOjE2MDM2NDIxMDYsI
mlzcyI6Imh0dHBzOi8vY3JzbGlvdC1ha3MtZGV2LmNlbnRyYWx1cy5jbG91ZGFwcC5henVyZS5jb20vYXV0aCIsImF1ZCI6IklkZW50aX
R5R2F0ZXdheSJ9.FahRD6gNnnT3XlHH9h69og0y7HbyZ-
gZrvfCnh1BROw2CcsbZ72BGNcINw3uTrn5_DDT6FH5plcA8klaSqViIke_LMY-
SIi_FXT4eBExti489wT0w5Dthft0u6jcutu_rORryvFisZMJAU3ApygALIyP0caY8KEdhXODHfJh0YIw6PdoeYEXP7YFZEJ9rz7wtd6TM
9VbzuZs3_Bm1PmNS-7T4mQcBl63P-
IF6mNXt5Xon9Fn0ScbH641v9KM6ad_HExB2W4muyCnyHGBmQaliLHcqtpIooC5KQPRwT5UQ1EJQCgwu1L1dwuNR-
vu5WceOux1VLm5dMFIbrsqR1bZrw
```

   As part of this request, User will be navigated to Azure AD B2C Login Page.

2. User Registration in Azure AD B2C

   As part of the User Registration, User needs to fill in the Signup form with the required information. Once the registration is complete, an Identity will be created for the user in B2C as a Member with Federated Identity or Active Directory.



3. Authorization Callback

   Once the User is registered and authenticated, a callback is triggered to `GET /connect/callback` with following query params

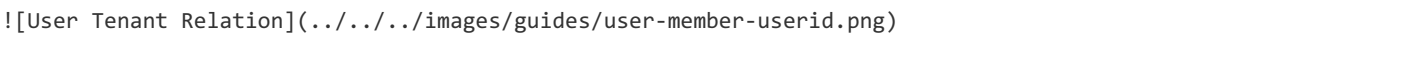
NAME	TYPE	EXAMPLE	DESCRIPTION
state	string	{'returnUrl':'https://aks-dev-spaces-odin.centralus.cloudapp.azure.com/', 'state':'{"test":"test"}', 'tenant': '151a2bbe-78d9-488b-8163-d0af87b88f46', 'nonce': 'n-0S6_WzA2Mj', 'client_id': 'IoTPlatform', invitation: null}	State Data that is being passed on from the Authorize action for post authentication data storage.
id_token	JWT Token	eyJhbGciOiJIUzI1NiIsInR5cCI....	Token that is being sent from B2C which contains the data about the User.
error	string	AADSTS16000	Error Code thrown by external provider
error_description	string	SelectUserAccount - This is an interrupt thrown...	Description of the error that was thrown by external auth provider

AuthState will provide us the JWT token that is minted during invitation process and it provides us the Tenant and arbitrary UserId that is created.

id_token that is sent as part of request from B2C, will provide UserId from B2C,
then a new User Tenant relationship is created in User table.

Data that is inserted into User table is as follows

Column	Value
UserId/PartitionKey	UserId from the **sub** claim in id_token.
Tenant/RowKey	TenantId from the invitation token from state.
Name	Name from **name** claim in id_token or Email from **emails** claims in id_token. Name will take precedence.
Roles	JSON Serialized string from roles claim invitation token from state.
Type	Member



UserSettings that are created as part of the Invitation process are transferred to UserId from B2C by updating in UserSettings table.

An UserSetting record is created with Arbitrary UserId created in Invitation process with setting key as **inviteId** for the user for reference.

Data that is inserted into User table is as follows

Column	Value
UserId/PartitionKey	UserId from the **sub** claim in id_token.
SettingKey/RowKey	InviteUserMetaData
Value	UserId from the **userId** claim in invitation token from state.

Finally, the placeholder UserTenant relationship that is created initially is deleted in User table.

User Registration Entities

User Registration is the process of inviting and providing access to an user for IoT Platform.

User Registration involves the following list of Entities

1. Azure B2C
2. User
3. UserSettings

Azure B2C

Azure B2C uses external Identity providers to authenticate users.

Currently for Platform is using 3M Azure AD to authenticate the Users for the applications.

Users who are added as Members to Azure AD B2C can authenticate using the Azure AD Account.

Users who are added as Guest, must have an B2B agreement with their organization AD to authenticate.

User

User table which will be residing in Table Storage of Storage Account holds the information about the user and the respective tenant for which user is assigned to.

Each environment has it's own storage account. If you have access to Azure Portal access to your subscription, you can search for it by name, which follows this convention:

> Storage Account follow the notation {ApplicationCode}**storageacct**{EnvironmentName}

For example, the storage account in CRLS's development (dev) environment is: **crsliotstorageacctdev**

Following table provides information about the data that is stored in the User table.

NAME	DESCRIPTION
Partition Key	Indicates UserId from Azure B2C
Row Key	Indicates TenantId
Time Stamp	Indicates Last updated Date and Time
Name	Indicates Name of the user
Roles	Indicates various roles User is assigned to.
Type	Indicates type of User

> **Reference:**

TYPE	DESCRIPTION
Member	User accepted the Invite and assigned to Tenant.
Invited	Invite sent to User and yet to sign in.
Client Credentials	

DataModel of User table is as follows

```
public class UserTenantModel : TableEntity
    {
        public string Roles { get; set; }

        public string Name { get; set; }

        public string Type { get; set; }

        public string UserId
        {
            get
            {
                return this.PartitionKey;
            }
        }

        public string TenantId
        {
            get
            {
                return this.RowKey;
            }
        }
    }
```

UserSettings

User Settings table will store the information specific to User and will work as an lookup data and configuration data for an User.

Following table provides information about the data that is stored in the User Settings table.

NAME	DESCRIPTION
Partition Key	UserId from Azure B2C
Row Key	Settings Key
Time Stamp	Last updated Date and Time
Value	Value for the Key.

DataModel of UserSettings table is as follows

```
public class UserSettingsModel : TableEntity
{
    public string Value { get; set; }

    public string UserId
    {
        get
        {
            return this.PartitionKey;
        }
    }

    public string SettingKey
    {
        get
        {
            return this.RowKey;
        }
    }
}
```

SendGrid

SendGrid is a cloud-based SMTP provider that allows you to send email without having to maintain email servers.

Document covers the following topics

1. How SendGrid is used by Serenity.
2. How SendGrid is configured.
3. Where is SendGrid Configuration is stored.
4. Guide to changing the SendGrid Account.
5. Guide to troubleshooting the SendGrid Account.

How SendGrid is used by Serenity

Send Grid is currently used in following two use cases of Serenity

- To send Invitation Mail as part of User Registration process.
- To send Alert Mails when Emails are enabled as part of Rules configuration.

How SendGrid is configured

Serenity Platform uses Free Plan of SendGrid

> Reference:
>
> Click here to see various plans of SendGrid.

As part of the Infrastructure setup for each environment, an SendGrid Account is created with Free Plan for each environment using

- **Email** specified as owner
- Randomly string as **Password**.

Password is added to KeyVault for reference using the secret name **Global--SendGridAcctPassword**

> Note:
>
> Currently SendGrid accounts are limited to 2 per Azure subscription. This limit exists for security and compliance reasons to prevent abuse.

Where is SendGrid Configuration is stored

Application will need the API Key if the SendGrid is used for sending mails.

Application Key is generated using API `https://api.sendgrid.com/v3/api_keys` of SendGrid.

Sample request

- Header

  ```
  Authorization:"Basic {base64 string from username and password}"
  ```

- Request Body

```
{ "name": "sendgrid-email-api-key", "scopes": [ "mail.send", "alerts.create", "alerts.read" ] }
```

- Response

```
{ "api_key": "SG.xxxxxxxx.yyyyyyyy", "api_key_id": "xxxxxxxx", "name": "My API Key", "scopes": [
"mail.send", "alerts.create", "alerts.read" ] }
```

API Key (api_key from response) is added to KeyVault with name **IdentityGatewayService--SendGridApiKey**.

> Note:
>
> SendGrid can support a 100 Application keys currently.

Guide to changing the SendGrid Account

Steps to change the SendGrid Account used in serenity platform are as follows

- Create a new SendGrid using a Email Address and Password as described here
- Generate API Key using SendGrid UserId and Password and update keyvault secret **IdentityGatewayService--SendGridApiKey** value as describe here

Guide to troubleshooting the SendGrid Account

TODO

User Registration APIs

This document describes how ot use register a User to a Serenity platform using the Serenity APIs.

User Registration APIs

The following APIs are use to register a new user to a tenant:

- **Tenants API**: POST /v1/tenants/invite
- **Connect API**: GET /connect/authorize

Prerequisites

In order to to invite a user, you must be an admin or have the UserManage permission. This is required to invite using through the Serenity UI or API. Specifically, the following is needed:

- **Role**: Admin or any equivalent role assigned with the above permission
- **Permission**: UserManage

How to use APIs to Register User

The registration steps are as follows:

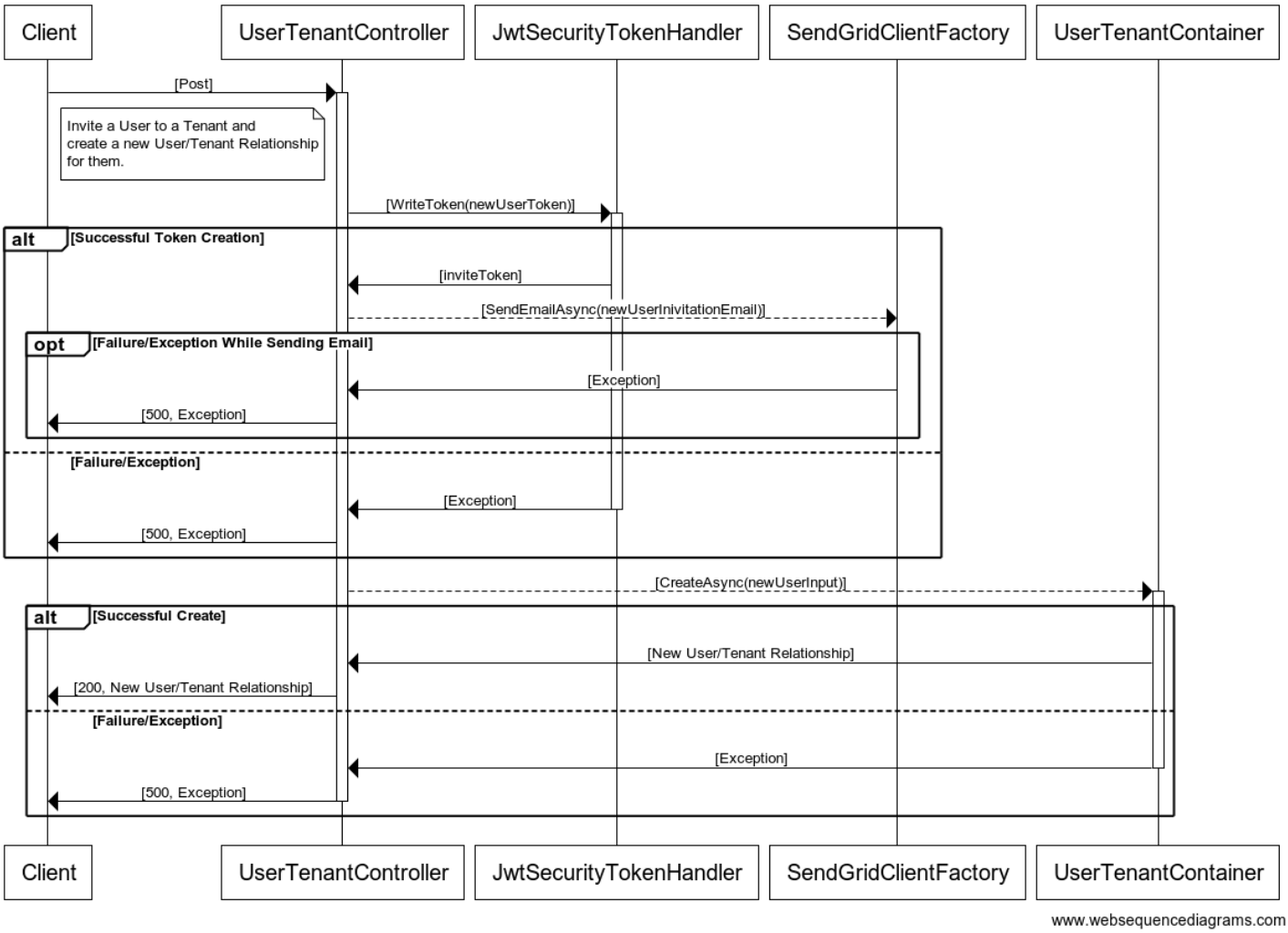
1. Invite User
2. Authorization
3. Callback

Invite User

In order to use the API to invite a new user to a tenant, you must have the following role and permission:

- **Role**: Admin or any equivalent role assigned with the above permission
- **Permission**: UserManage

```
POST /v1/tenants/invite
```



Request Body

```
{
  "email_address": "testuser@gmail.com",
  "role": "admin",
  "email_subject" : "custom subject",
  "invite_uri" : "https://emdudahiotfepkg.azurewebsites.net",
  "from_email" : "testemail@email.com",
  "from_message" : "This is a custom message",
  "invite_user_meta_data : "User data info"
  "message_body" : "Welcome to the iot platform"
}
```

- from_email,
- from_message,
- email_subject,
- invite_uri,
- invite_user_meta_data
- message_body

are optional and have these default values:

- DefaultFromEmail = "iotplatformnoreply@mmm.com";
- DefaultFromMessage = "3M IoT Platform Team";
- DefaultSubject = "Invitation to IoT Platform";
- DefaultInviteUri = Serenity front end.
- invite_user_meta_data = ""
- message_body = "Tap the button below to accept this invitation. If you did not expect an invitation, you can delete this

email."

Response

Returns 200 OK and object of the user invited in JSON. Example:

```
{
  "roles": "['admin']",
  "name": "testuser@gmail.com",
  "type": "Invited",
  "userId": "ec9745fa-9fa9-4d35-ab55-1e3bd064c4a2",
  "tenantId": "78984017-fa19-4aa6-8824-7bdd9d9c50c3",
  "roleList": [
    "admin"
  ],
  "partitionKey": "ec9745fa-9fa9-4d35-ab55-1e3bd064c4a2",
  "rowKey": "78984017-fa19-4aa6-8824-7bdd9d9c50c3",
  "timestamp": "0001-01-01T00:00:00+00:00",
  "eTag": "W/"datetime'2020-10-22T08%3A45%3A35.0798098Z'"
}
```

Authorization

Once the email invitation is sent to the user and user clicks on the invite, this method is called to redirect the user to Login and this also invokes the **callback** internally

`GET /connect/authorize`

You must pass the following items in the query string

NAME	TYPE	EXAMPLE	DESCRIPTION
redirect_uri	URI	https://aks-dev-spaces-odin.centralus.cloudapp.azure.com/	This is the URI that identity gateway should return to when it is done with the authentication.
state	string	{'test':'test'}	This is the string that will be returned to the frontend. This is for any WebUI state to be preserved.
client_id	string	IoTPlatform	This is the client id which gets included into the token as the audience
nonce	string	n-0S6_WzA2Mj	String value used to associate a Client session with an ID Token, and to mitigate replay attacks.
tenant	guid	151a2bbe-78d9-488b-8163-d0af87b88f46	Identifier for a tenant. User must have access to the tenant. If none is supplied then the LastUsedTenant setting will be the tenant they log into. If there is no LastUsedTenant Setting then a random one will be selected. If none is available an empty token will be minted
invite	JWT Token	eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI...	A JWT Token containing three claims "sub" (user_id of invitation record), "tenant" (tenant Guid to assign them to), "role" (role(s) to assign them to)

Response

Redirects to authentication externally

Callback

`GET /connect/callback`

You must pass the following items in multi-part form

NAME	TYPE	EXAMPLE	DESCRIPTION
state	string	{'returnUrl':'https://aks-dev-spaces-odin.centralus.cloudapp.azure.com/', 'state':'{"test":"test"}', 'tenant': '151a2bbe-78d9-488b-8163-d0af87b88f46', 'nonce': 'n-0S6_WzA2Mj', 'client_id': 'IoTPlatform', invitation: null}	This string is returned from the authentication. It originally comes from the authorize endpoint on this same controller.
id_token	JWT Token	eyJhbGciOiJIUzI1NiIsInR5cCI....	This is the client id which gets included into the token as the audience
error	string	AADSTS16000	Error Code thrown by external provider
error_description	string	SelectUserAccount - This is an interrupt thrown...	Description of the error that was thrown by external auth provider

The state from above has invitation property which indicates that the user has logged in for the first time if not null.

If the user logged in for the first time using the Invite, data will be added/updated in below storage tables:

- User
- UserSettings

Response

Redirects back to location specified in the auth state variable.