Introduction to Microsoft Identity

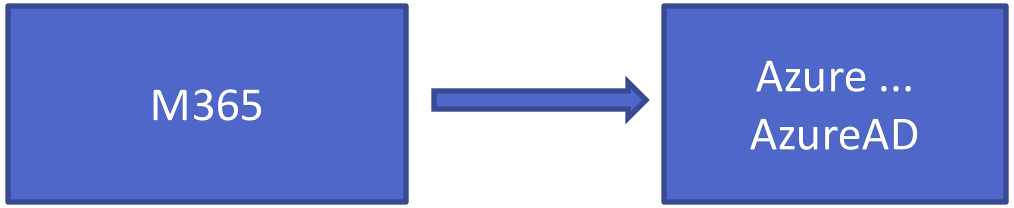
Contents

- Security Basics
- Claim Based Authentication
- Token Based Authentication
- Cloud Hosted Apps - Azure AD App Registrations
- ADAL

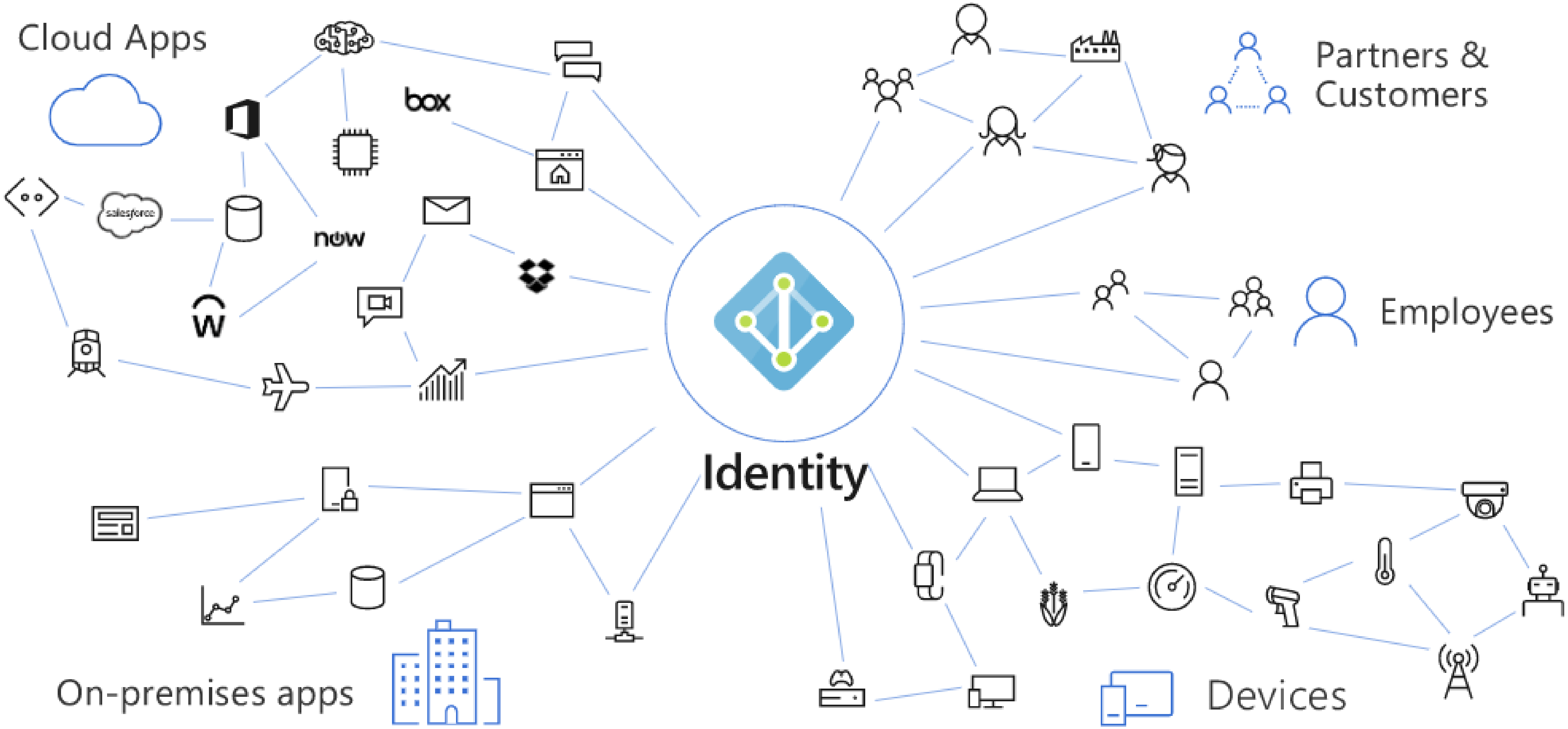
Introduction to Microsoft Identity

Overview of authentication and authorization

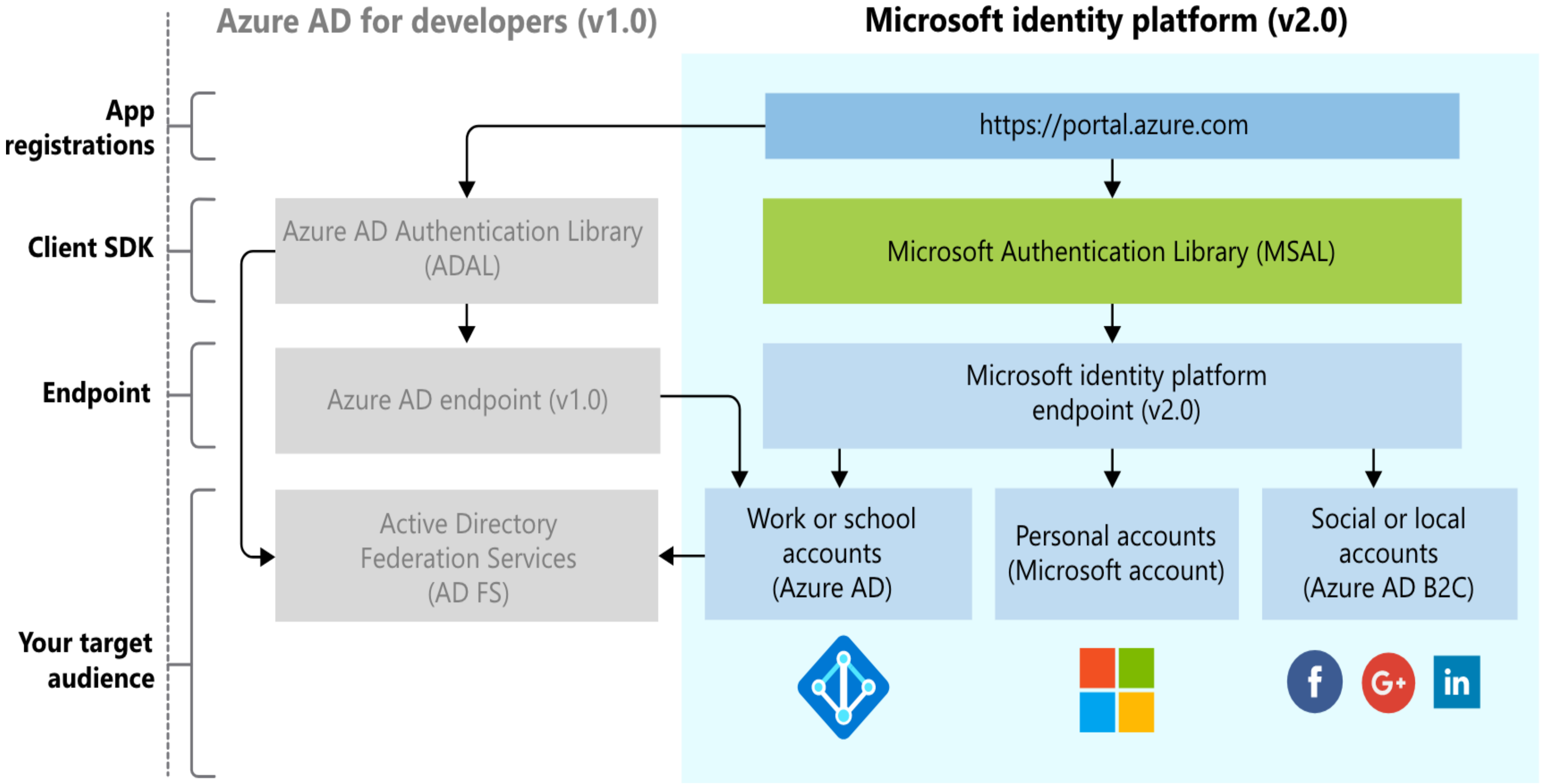
- **Authentication** is the process of proving you are who you say you are. Authentication is sometimes shortened to AuthN.
- **Authorization** is the act of granting an authenticated party permission to do something. It specifies what data you're allowed to access and what you can do with that data. Authorization is sometimes shortened to AuthZ.
- Delegating authentication and authorization enables scenarios such as:
  - Conditional Access policies that require a user to be in a specific location.
  - The use of multi-factor authentication.
  - Single sign-on (SSO.



Microsoft identity platform



Microsoft Identity Platform Architecture


Azure AD for developers (v1.0)
Microsoft identity platform (v2.0)
App registrations
Client SDK
Endpoint
Your target audience
https://portal.azure.com
Azure AD Authentication Library (ADAL)
Microsoft Authentication Library (MSAL)
Azure AD endpoint (v1.0)
Microsoft identity platform endpoint (v2.0)
Active Directory Federation Services (AD FS)
Work or school accounts (Azure AD)
Personal accounts (Microsoft account)
Social or local accounts (Azure AD B2C)
f
G+
in

oAuth Standards & Terms

Identity Provider

System that does Identity Management

- Traditionally Active Directory

In a more Cloud based Approach

- Social Logins
- Cloud based Logins
  - Azure AD
  - Firebase
  - …

OpenID Connect

A simple identity layer on top of the OAuth 2.0 protocol, which allows computing clients to verify the identity of an end-user based on the authentication performed by an authorization server

Enables Single Sign-on

Has become the leading standard for single sign-on and identity provision on the Internet by using:

- simple JSON-based identity tokens (JWT),
- delivered via the OAuth 2.0 protocol

JSON Web Tokens

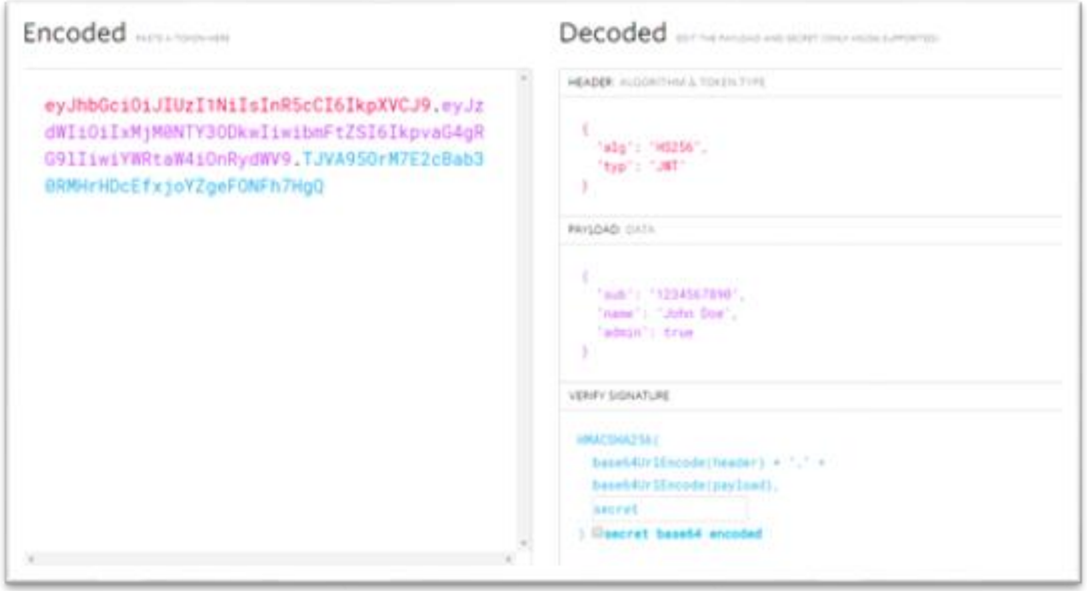
An open, industry standard RFC 7519 method for representing claims securely between two parties

Defines a format of how Auth information can be exchanged

Can be sent through a URL, POST parameter, or inside an HTTP header

Contains all the required information about the user

Doku @ https://jwt.io/



Encoded PASTE A TOKEN HERE
Decoded EDIT THE PAYLOAD AND SECRET (ONLY HS256 SUPPORTED)
HEADER: ALGORITHM & TOKEN TYPE
PAYLOAD: DATA
VERIFY SIGNATURE
secret base64 encoded

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJz
dWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gR
G9lIiwiYWRtaW4iOnRydWV9.TJVA95OrM7E2cBab3
0RMHrHDcEfxjoYZgeFONFh7HgQ
```

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret
) □secret base64 encoded
```

OAuth 2.0

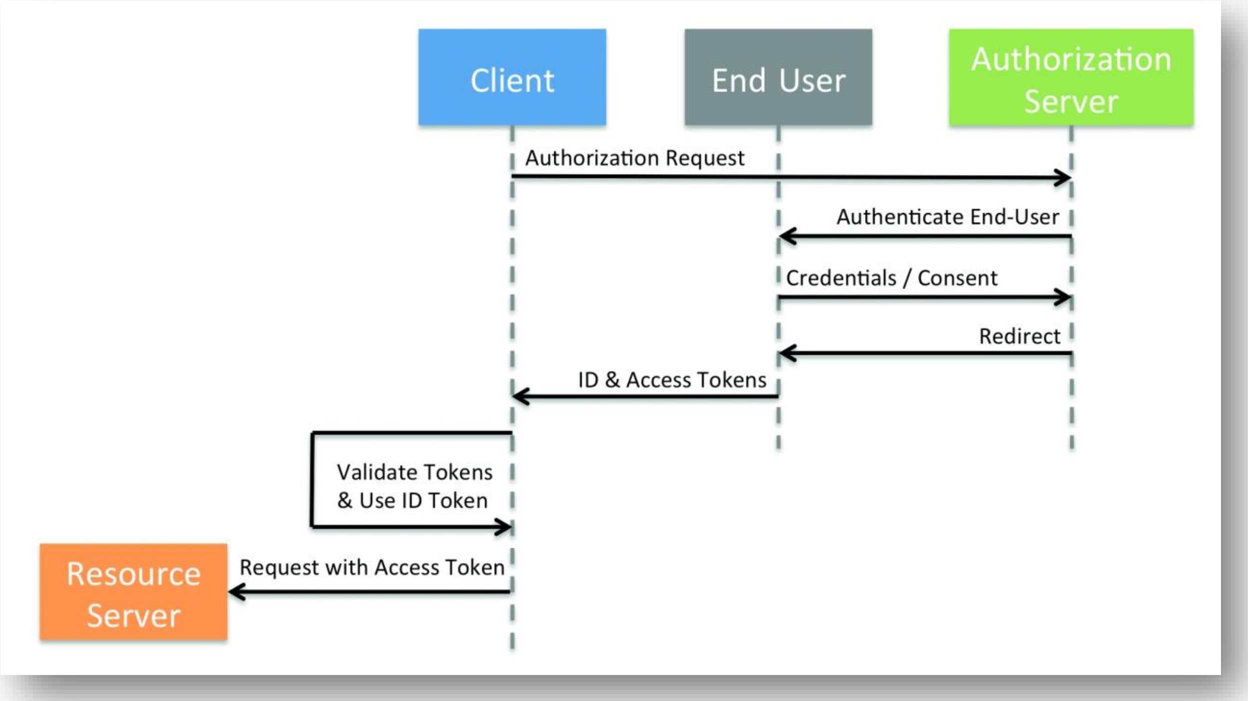
OAuth 2.0 is the industry-standard protocol for authorization

Defines Token Flows - How you get your tokens

- Authorization Code Grant
- Implicit Flow
- Client Credentials Flow
  - Certificate
- Daemon -> Service

Flow Depends on Use Case

- Web / Mobile



Token Types

ID Token / AuthToken

- JWT encoded Identity Information about the user

Access (Ressource) Token akn Bearer Tokens

- Used to access 3rd-party ressources without any further Identification
    - MS Graph, SharePoint, Azure Blob Storage, ...
    - Attached to the http-request in the header

Refresh Token

- The refresh token normally is sent together with the access token.
- The refresh token is used to get a new access token, when the old one expires

Grants & Scopes

Methods to get access tokens from the authorization server are called **grants**.

The same method used to request a token is also used by the resource server to validate a token.

The four basic grant types are:

- Authorization Code,
- Implicit,
- Resource Owner Credentials and
- Client Credentials

Scope

- In OAuth 2, the scope is a way to restrict access to specified areas

Cloud Hosted Applications

Cloud Hosted Applications

Implement your Add-In in what ever framework you like – I choose .NET Core & Angular

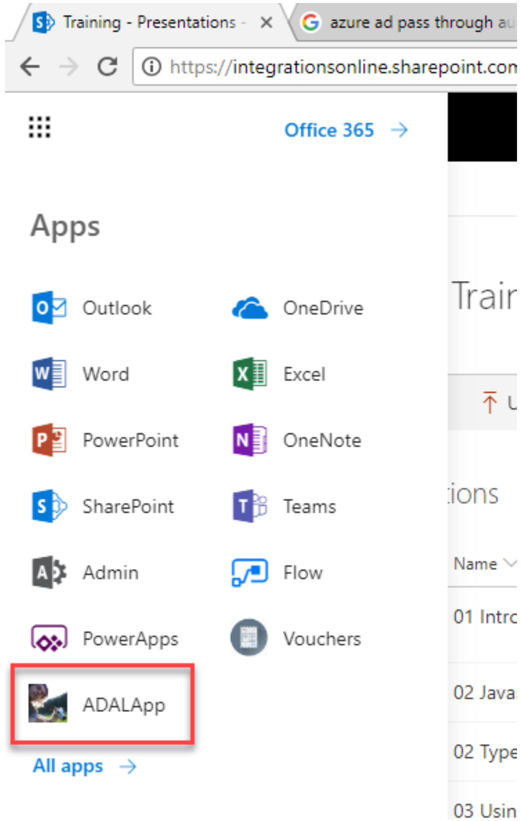
Similar to Provider Hosted Add-Ins

Easy Development Model of your choice: ie

- Angular & Net Core
- Node.js

Configure Enterprise Apps in Azure AD

Show up in App-Launcher & Hybrid App Launcher ☺

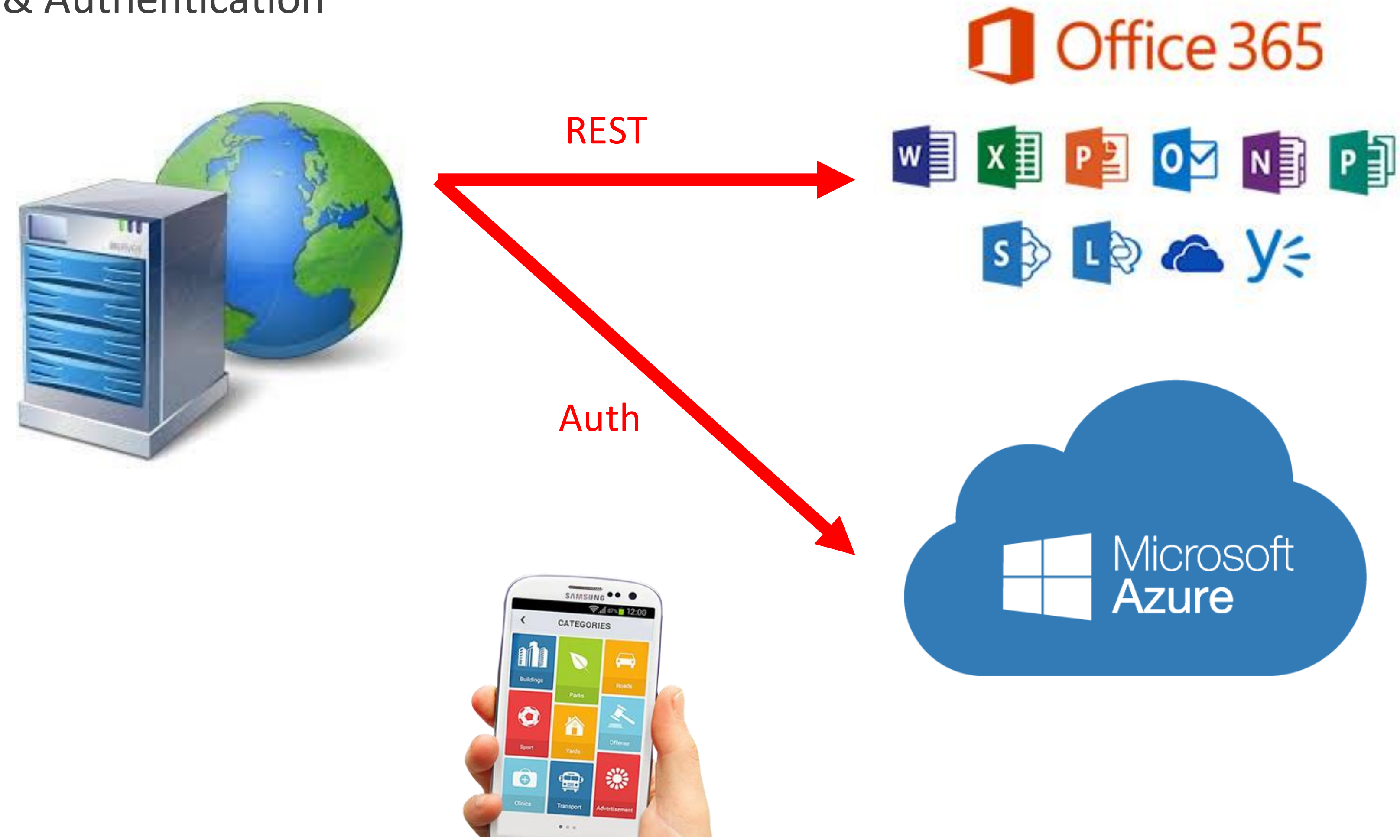


Cloud Hosted Applications Architecture

Uses Azure AD for Registration & Authentication

Hosted

- On Premises
- Azure / Cloud



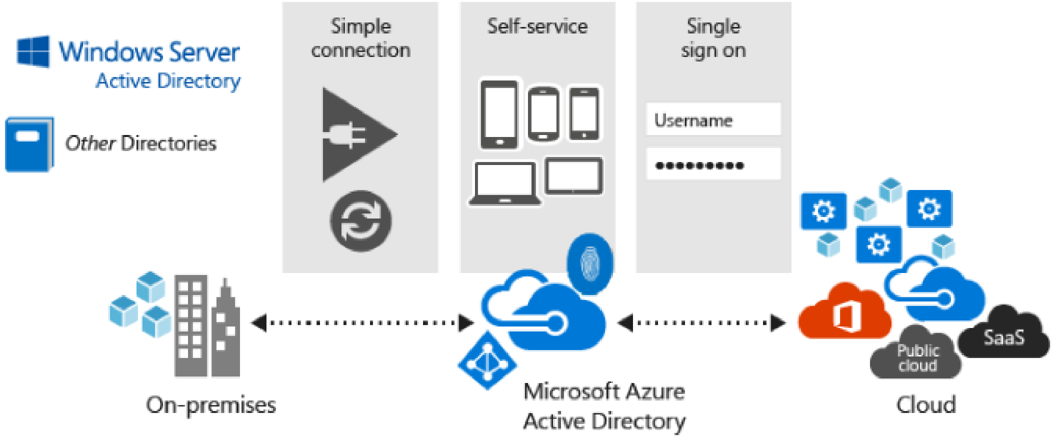
REST

Auth

Azure Active Directory

Is Microsoft's multi-tenant cloud based directory and identity management service.

Provides easy to use solution to give employees and business partners single sign-on (SSO) access to thousands of cloud SaaS Applications

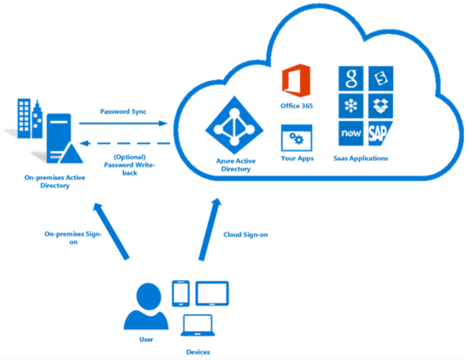


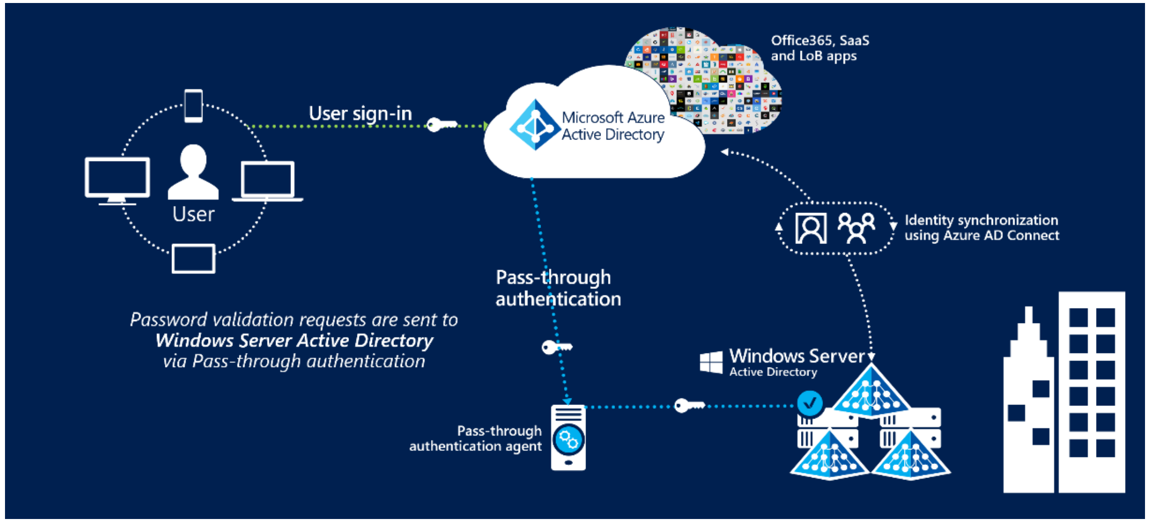
Connect on-premises Active Directory with Azure AD

Most companies have their accounts in their local AD

Three choices

- ADFS
- Synchronized identity
- Azure AD Pass Through Auth





Office365, SaaS and LoB apps
User sign-in
Microsoft Azure Active Directory
User
Identity synchronization using Azure AD Connect
Pass-through authentication
Windows Server Active Directory
Pass-through authentication agent

Password validation requests are sent to ***Windows Server Active Directory*** *via Pass-through authentication*

Token Base Auth Libraries

Active Directory Authentication Library – ADAL

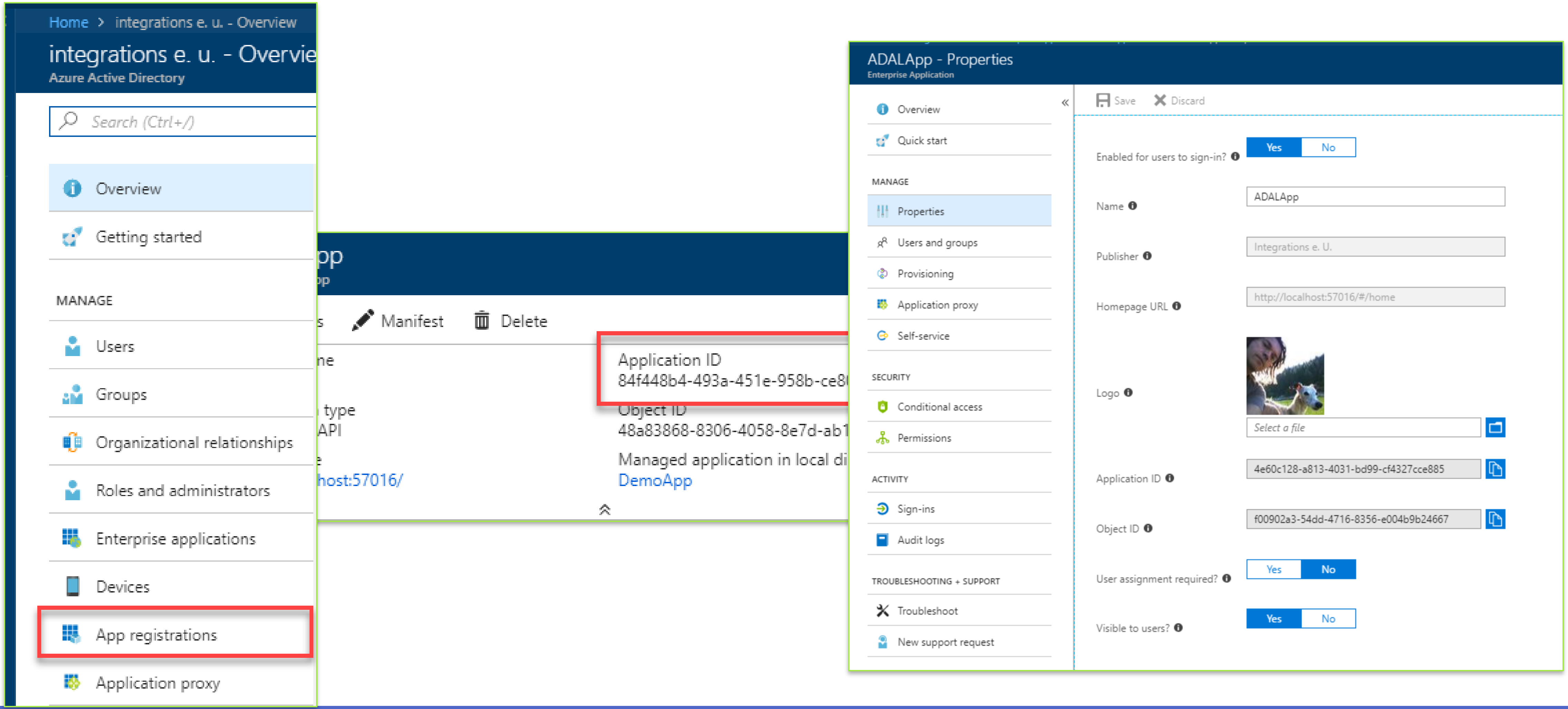
- Around for some year about
- Lot of samples for different Usecases (.NET, JS, Angular)
- Not 100% OpenID Connect compatible

Microsoft Authentication Library – MSAL

- Still in Preview
- 100% OpenID Connect compatible
- JS & Angular Wrapper available

Register Cloud Hosted Applications

Register Enterprise Apps



ne

Application ID
84f448b4-493a-451e-958b-ce8

type
API

Object ID
48a83868-8306-4058-8e7d-ab1

e
host:57016/

Managed application in local di
DemoApp

ADALApp - Properties
Enterprise Application

Enabled for users to sign-in? [✓ Yes] No

Name: ADALApp

Publisher: Integrations e. U.

Homepage URL: http://localhost:57016/#/home

Logo: Select a file

Application ID: 4e60c128-a813-4031-bd99-cf4327cce885

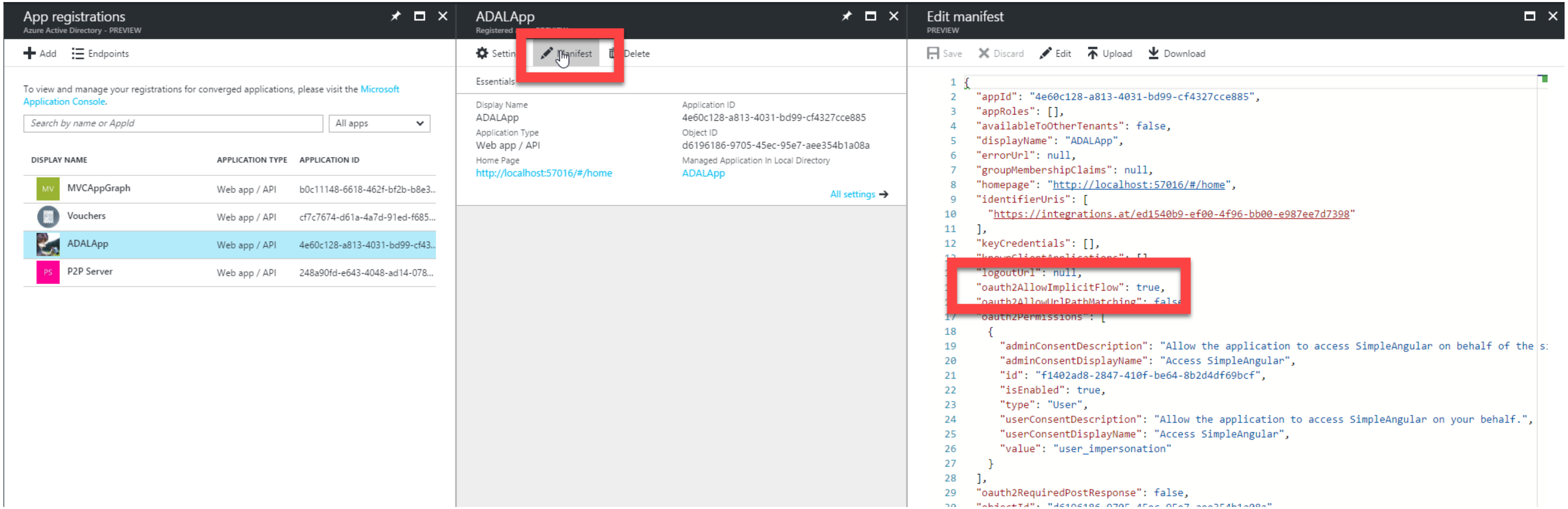
Object ID: f00902a3-54dd-4716-8356-e004b9b24667

User assignment required? Yes [✓ No]

Visible to users? [✓ Yes] No

oAuth Implicit Flow

Enable in AppMainifest



To view and manage your registrations for converged applications, please visit the Microsoft Application Console.

Search by name or AppId | All apps

DISPLAY NAME	APPLICATION TYPE	APPLICATION ID
MVCAppGraph	Web app / API	b0c11148-6618-462f-bf2b-b8e3...
Vouchers	Web app / API	cf7c7674-d61a-4a7d-91ed-f685...
ADALApp	Web app / API	4e60c128-a813-4031-bd99-cf43...
P2P Server	Web app / API	248a90fd-e643-4048-ad14-078...

Display Name
ADALApp

Application Type
Web app / API

Home Page
http://localhost:57016/#/home

Application ID
4e60c128-a813-4031-bd99-cf4327cce885

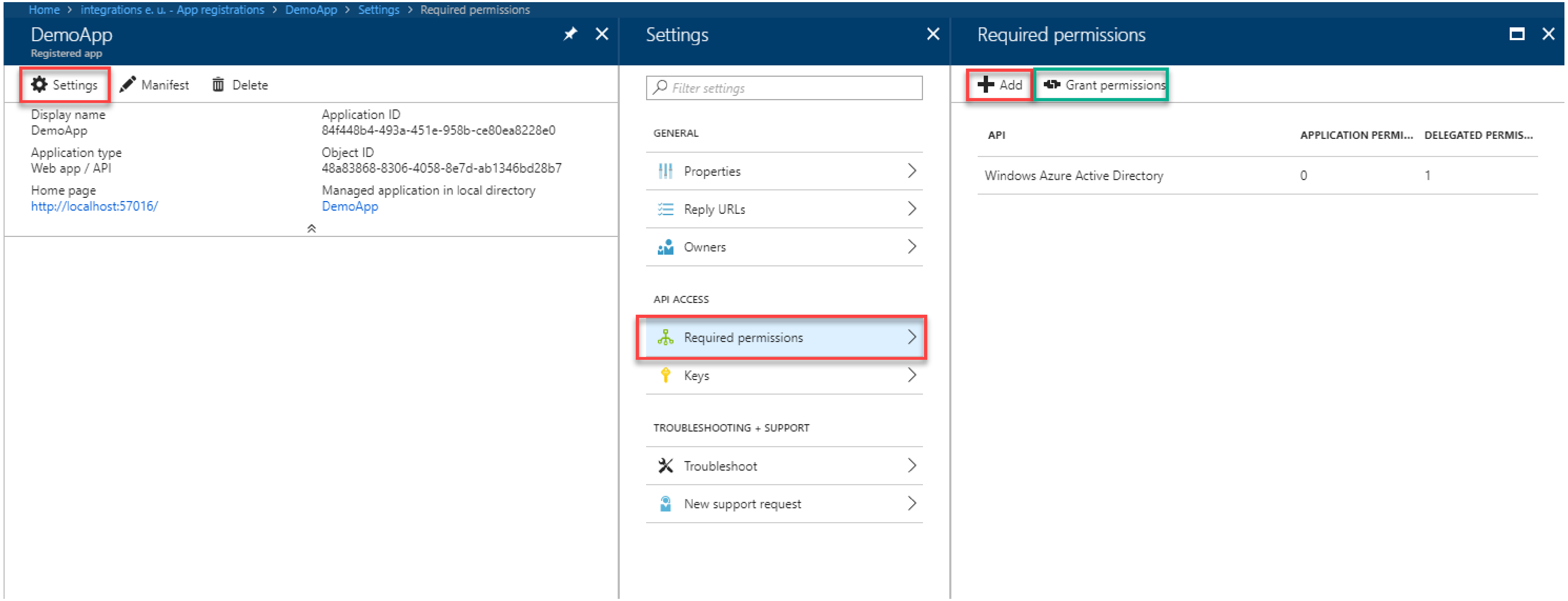
Object ID
d6196186-9705-45ec-95e7-aee354b1a08a

Managed Application In Local Directory
ADALApp

```
{
  "appId": "4e60c128-a813-4031-bd99-cf4327cce885",
  "appRoles": [],
  "availableToOtherTenants": false,
  "displayName": "ADALApp",
  "errorUrl": null,
  "groupMembershipClaims": null,
  "homepage": "http://localhost:57016/#/home",
  "identifierUris": [
    "https://integrations.at/ed1540b9-ef00-4f96-bb00-e987ee7d7398"
  ],
  "keyCredentials": [],
  "knownClientApplications": [],
  "logoutUrl": null,
  "oauth2AllowImplicitFlow": true,
  "oauth2AllowUrlPathMatching": false,
  "oauth2Permissions": [
    {
      "adminConsentDescription": "Allow the application to access SimpleAngular on behalf of the s:
      "adminConsentDisplayName": "Access SimpleAngular",
      "id": "f1402ad8-2847-410f-be64-8b2d4df69bcf",
      "isEnabled": true,
      "type": "User",
      "userConsentDescription": "Allow the application to access SimpleAngular on your behalf.",
      "userConsentDisplayName": "Access SimpleAngular",
      "value": "user_impersonation"
    }
  ],
  "oauth2RequiredPostResponse": false,
  "objectId": "d6196186-9705-45ec-95e7-aee354b1a08a"
```

Assign Permissions

Don‘t forget to „Grant Permissions“ after you have added Permissions



Display name
DemoApp

Application type
Web app / API

Home page
http://localhost:57016/

Application ID
84f448b4-493a-451e-958b-ce80ea8228e0

Object ID
48a83868-8306-4058-8e7d-ab1346bd28b7

Managed application in local directory
DemoApp

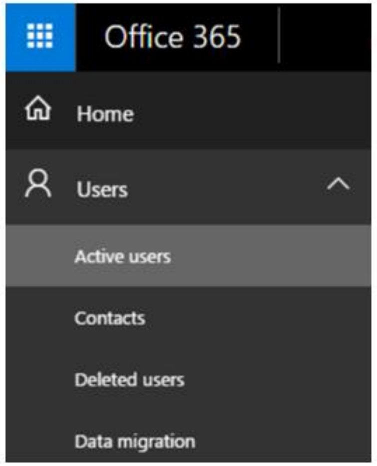
API	APPLICATION PERMI...	DELEGATED PERMIS...
Windows Azure Active Directory	0	1

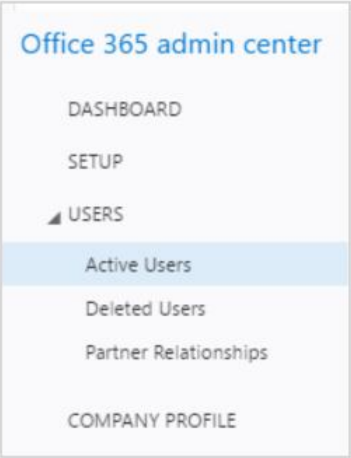
Active Directory Authentication Library

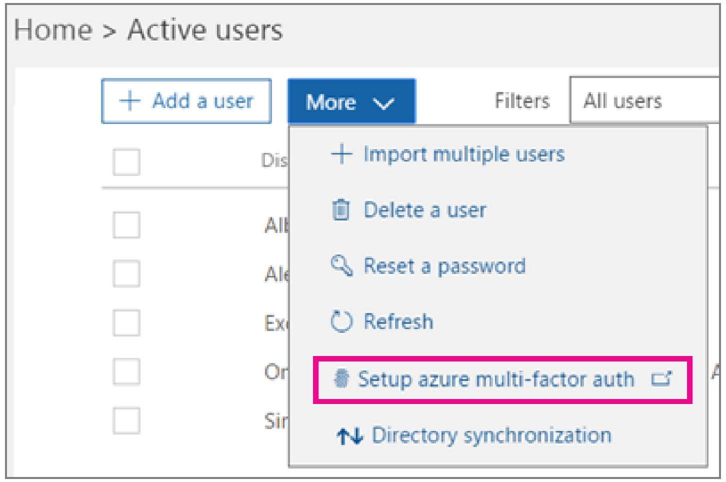
Designed to make secured resources in your directory available to client applications

Works with OAuth 2.0 to enable more authentication and authorization scenarios, like

- Multi-factor Authentication (MFA)
- Serveral forms of SAML Auth







Where to use it?

JavaScript Applications accessing Office 365 Ressources (or not)!!!

Can be used for Singel Page Applications (SPA)

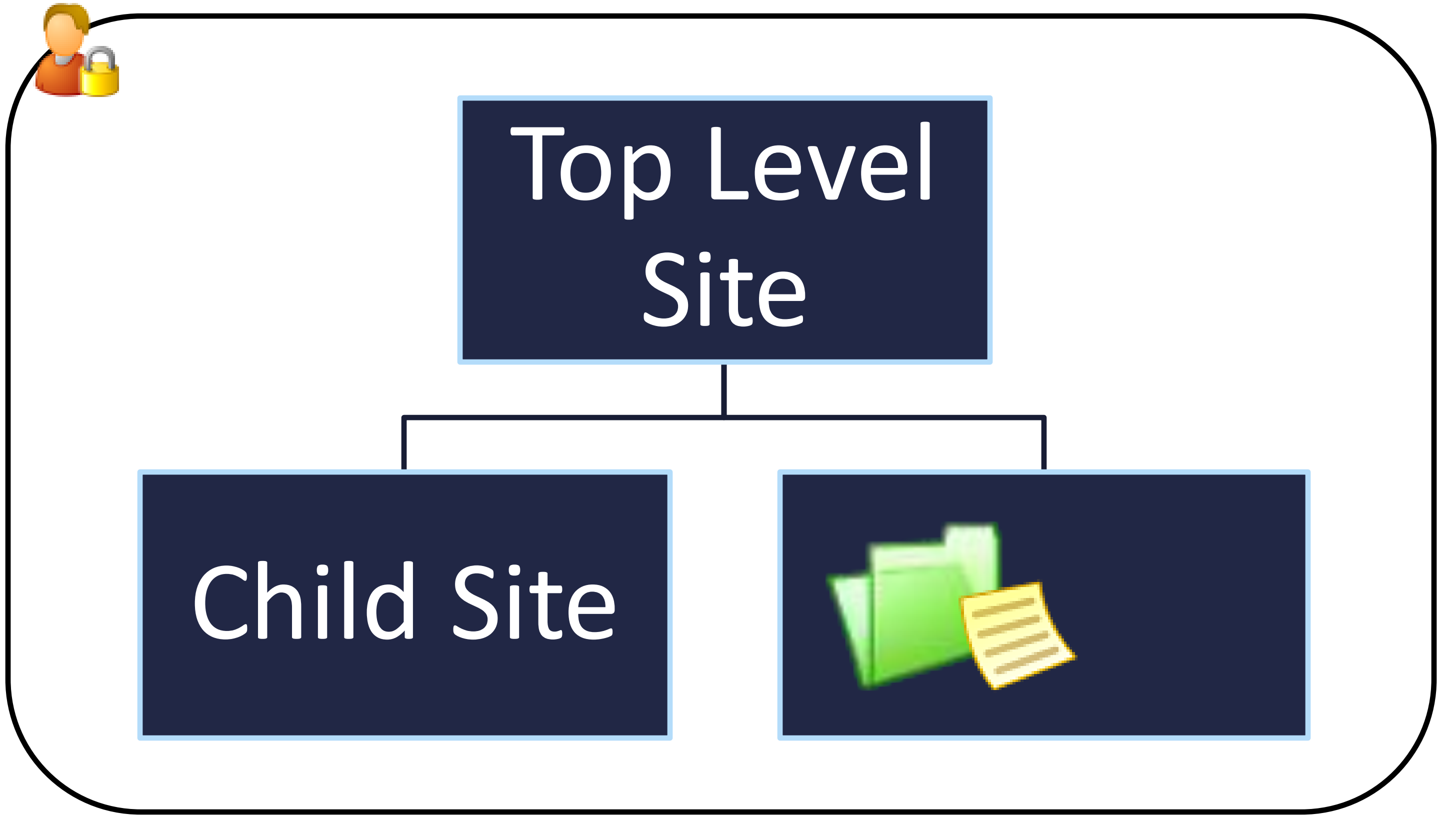
Azure Active Directory Authentication Library (ADAL) for JavaScript available

Use adal-angular.js for Angular JS integration

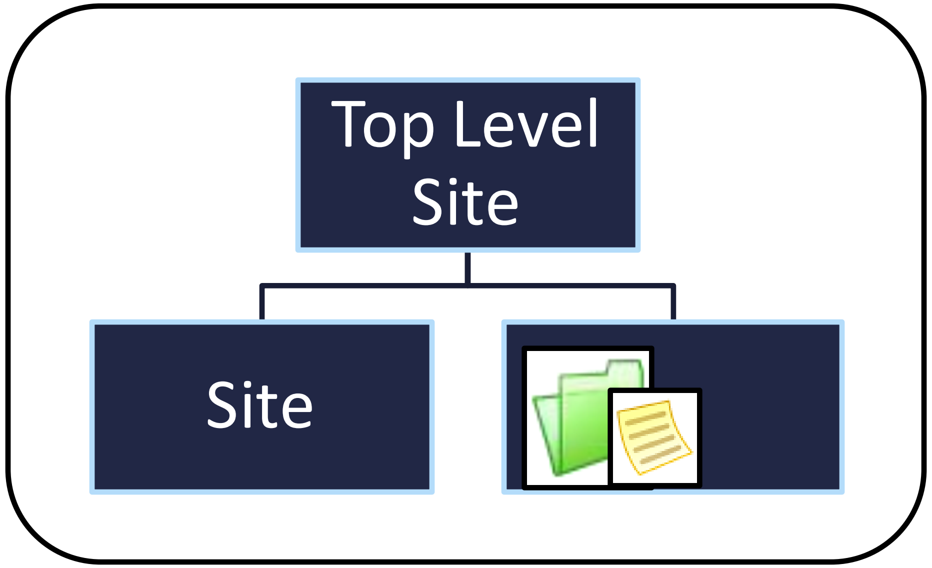
Use https://github.com/sureshchahal/angular2-adal for Angular

SharePoint Security

SharePoint Security Hierarchy



SharePoint Authorization



SharePoint Group

User / Group

Authentication Provider

Active Directory
Azure AD

CUSTOM:
ASP.NET Identity
Social Logins

Benefits of using SharePoint Groups

Flexibility - can add / remove other Groups / Users without touching permissions on Sites / Lists / Items

Can be used with Security Principals Authenticated by other Authentication Providers than Windows

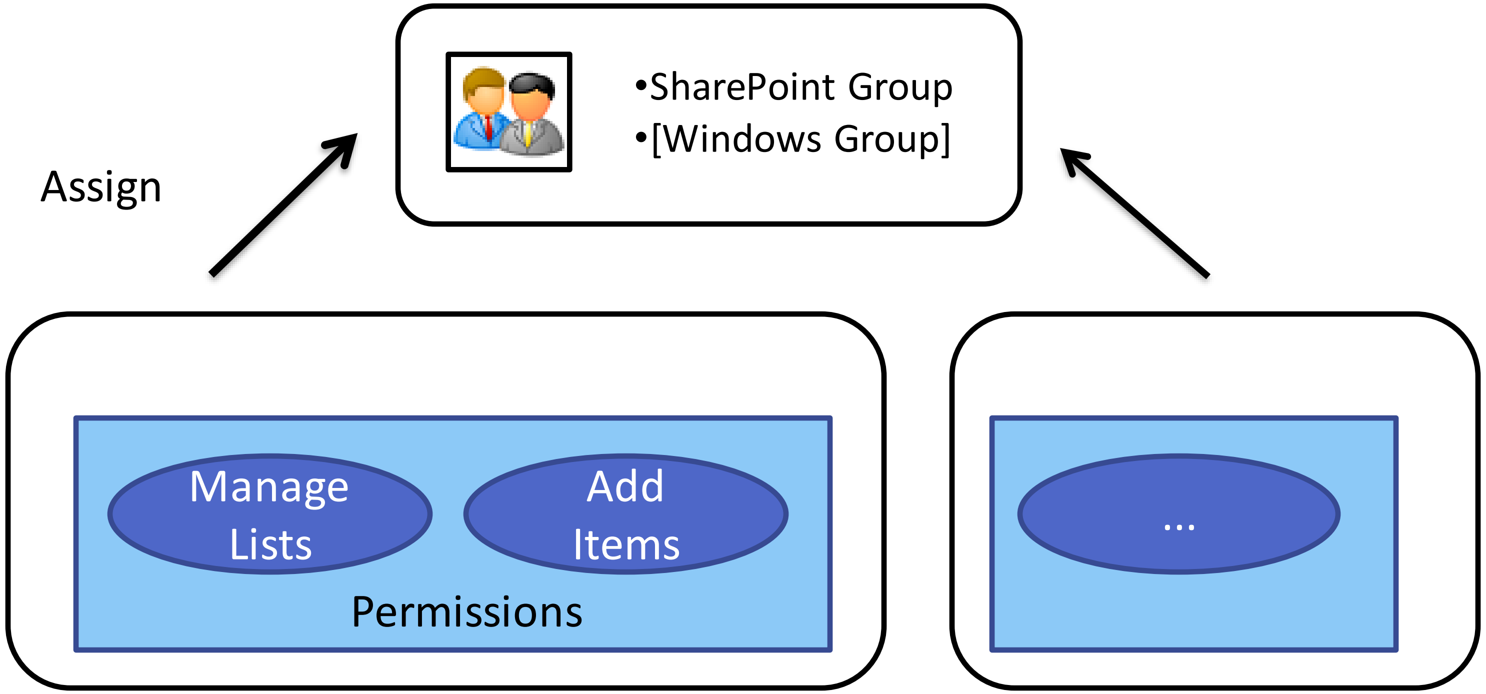
Securables

Securable Ressources in SharePoint

- Site (Site Collection) – Top Level Site
- Web (Site)
- List
- [Folder]
- ListItem

Permission Level

Permission Levels are Groupings of individual Permissions (SPBasePermission)



Managing Users & Group Memberships

Coding Security overview

4 Steps to code basic permsisions

Create Group

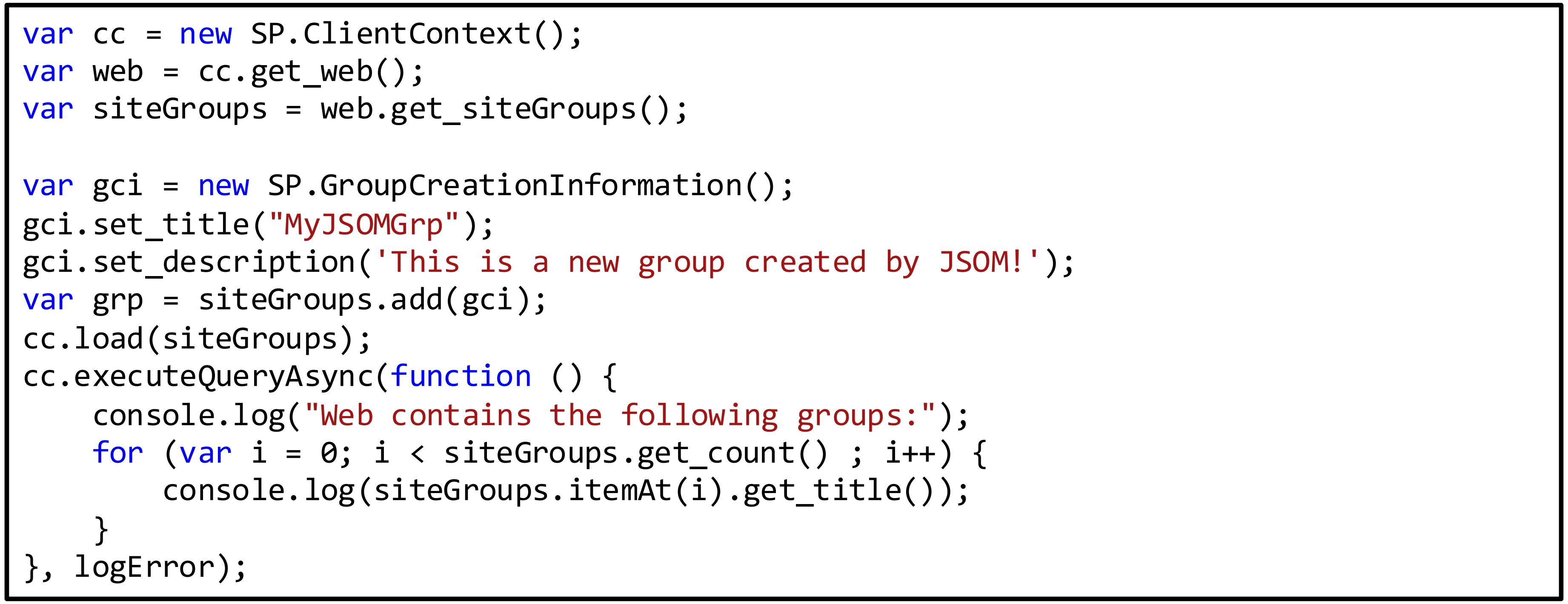
Add Users to Group

Create or Get Permission Level

Assign Permission Level to Group

Create Group

A group is created by passing the SP.GroupCreationInformation object to the .Add() method of the groups collection



```
var cc = new SP.ClientContext();
var web = cc.get_web();
var siteGroups = web.get_siteGroups();

var gci = new SP.GroupCreationInformation();
gci.set_title("MyJSOMGrp");
gci.set_description('This is a new group created by JSOM!');
var grp = siteGroups.add(gci);
cc.load(siteGroups);
cc.executeQueryAsync(function () {
    console.log("Web contains the following groups:");
    for (var i = 0; i < siteGroups.get_count() ; i++) {
        console.log(siteGroups.itemAt(i).get_title());
    }
}, logError);
```

Add User to Group

Users ar created using the SP.UserCreationInformation object

```
var cc = new SP.ClientContext(siteUrl);
var grps = cc.get_web().get_siteGroups();
var oGroup = grps.getById(7);
var uci = new SP.UserCreationInformation();
uci.set_email('alias@somewhere.com');
uci.set_loginName('DOMAIN\alias');
uci.set_title('John');
this.usr = oGroup.get_users().add(uci);

cc.load(usr);
cc.executeQueryAsync(function() {
    console.log("user created");
}, logError);
```

Implementing Permissions

Create Permission Level

Get the SP.BasePermissions

Add them to the SP.RoleDefinitionCreationInformation

Add that to the SP.RoleDefinitionCollection

```
var cc = new SP.ClientContext(siteUrl);
var web = cc.get_web();
var basePerm = new SP.BasePermissions();
basePerm.set(SP.PermissionKind.createAlerts);
basePerm.set(SP.PermissionKind.manageAlerts);
var roleCreationInfo = new SP.RoleDefinitionCreationInformation();
roleCreationInfo.set_basePermissions(basePerm);
roleCreationInfo.set_description('A new role with create and manage alerts permission');
roleCreationInfo.set_name('Create and Manage AlertsT');
var permissionLevel = web.get_roleDefinitions().add(roleCreationInfo);
cc.load(permissionLevel);
cc.executeQueryAsync(function() {
     console.log(permissionLevel.get_name() + ' role created.');
}, logError);
```

Assign Permission Level to Group / User

```
var cc = new SP.ClientContext(siteUrl);
var list = cc.get_web().get_lists().getByTitle('MyList');
var li = list.get_items().getById(1);
li.breakRoleInheritance(false);
var usr = cc.get_web().get_siteUsers().getByLoginName('DOMAIN\\alias');
var rdlbs = SP.RoleDefinitionBindingCollection.newObject(cc);
rdlbs.add(cc.get_web().get_roleDefinitions().getByType(SP.RoleType.reader));
li.get_roleAssignments().add(usr, rdlbs);
cc.load(usr);
cc.load(li);
cc.executeQueryAsync(function() {
    console.log('Role inheritance broken for item ' + li.get_item('Title') + ' and new role assignment for
     '+ usr.get_loginName());
}, logError);
```