

# Introduction to Modern Software Development



# Student Introductions

- Basic Info
  - What's your name?
  - Where do you work? (optional)
  - How long have you been a developer?
- List skills with which you already feel comfortable
  - .NET programming with C# or VB.NET
  - SharePoint farm solution and add-in development
  - JavaScript and TypeScript
  - jQuery, React and Angular
  - Programming with HTTP Requests, REST and OData
  - Developing with ASP.NET MVC and Web API



# Agenda

- Understanding SharePoint Development Strategies
- Creating a SharePoint Development Environment
- Programming the Client-side Object Model (CSOM)
- Understanding Azure as a Development Platform
- Developing with TypeScript and Interfaces



# Evolution of the SharePoint Platform

- Farm Solutions
- ~~Sandboxed Solutions~~
- SharePoint Add-ins
- JavaScript Injection
- Remote Provisioning
- SharePoint Framework (SPFx)



# SharePoint App Add-in Model

- SharePoint 2013 introduced new development model
  - Originally introduced as "SharePoint App" model
  - Marketing folks renamed "SharePoint App" to "SharePoint Add-in"
- Add-in model designed to replace farm solutions
  - Add-ins designed to supported SPO and SharePoint on-premises
  - Add-in code not allowed to run on SharePoint host server
  - Add-in talks to SharePoint using REST and CSOM
  - Add-in authenticates and establishes add-in identity
  - Add-in has permissions independent of user
  - Add-ins deployed to catalogs using publishing scheme



# SharePoint APIs

- SharePoint REST API
  - Commonly used with client-side JavaScript code
  - Good fit when developing SharePoint-hosted add-ins
  - Accessible to any type of client on any platform
- Client-side Object Model (CSOM)
  - Commonly used with server-side C# code
  - Good fit when developing provider-hosted add-ins
  - Good fit when creating desktop clients (e.g. Console app)
  - Used to perform remote provisioning in SPO sites



# JavaScript Injection

- JavaScript injection based on central concept...
  1. upload custom JavaScript code to SharePoint Online
  2. execute code using identity and permissions of current user
- Approaches for using JavaScript injection
  - Script Editor Web Part
  - Adding JavaScript code behind SharePoint site pages
  - Full-blown Visual Studio project development
- Why create solution using JavaScript Injection?
  - Provides more flexibility than SharePoint add-in model
  - Poses fewer constraints than SharePoint add-in model





# Remote Provisioning

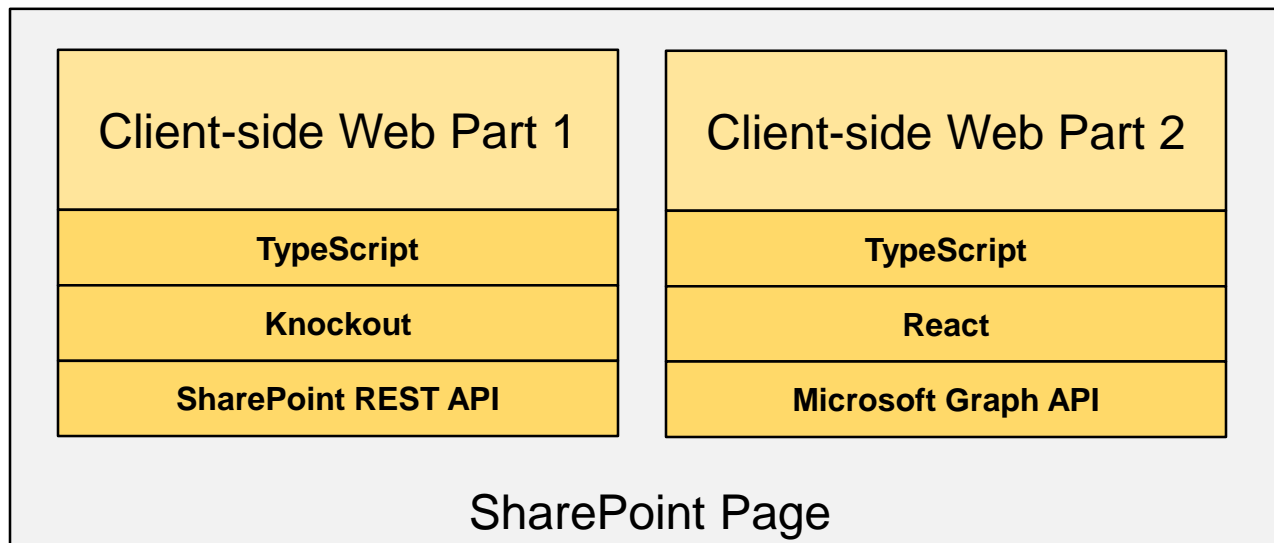
- Remote provisioning in SPO
  - Use CSOM to create SPO site elements
  - Recommended over SharePoint solutions & features
- What can you create with Remote Provisioning
  - New child sites, lists and document libraries
  - Site columns, content types and remote event receivers
  - New pages with custom JavaScript logic
  - User custom actions with custom JavaScript logic





# The SharePoint Framework (SPFx)

- Development model based on pages and web parts
  - Based on client-side development with JavaScript or TypeScript
  - Code runs with authenticated identity of current user
  - Easy access to SharePoint and Office 365 content and data
  - Developer tools designed to support cross-platform development
  - Great support for targeting mobile devices



# Agenda

- ✓ Understanding SharePoint Development Strategies
- Creating a SharePoint Development Environment
  - Programming the Client-side Object Model (CSOM)
  - Understanding Azure as a Development Platform
  - Developing with TypeScript and Interfaces



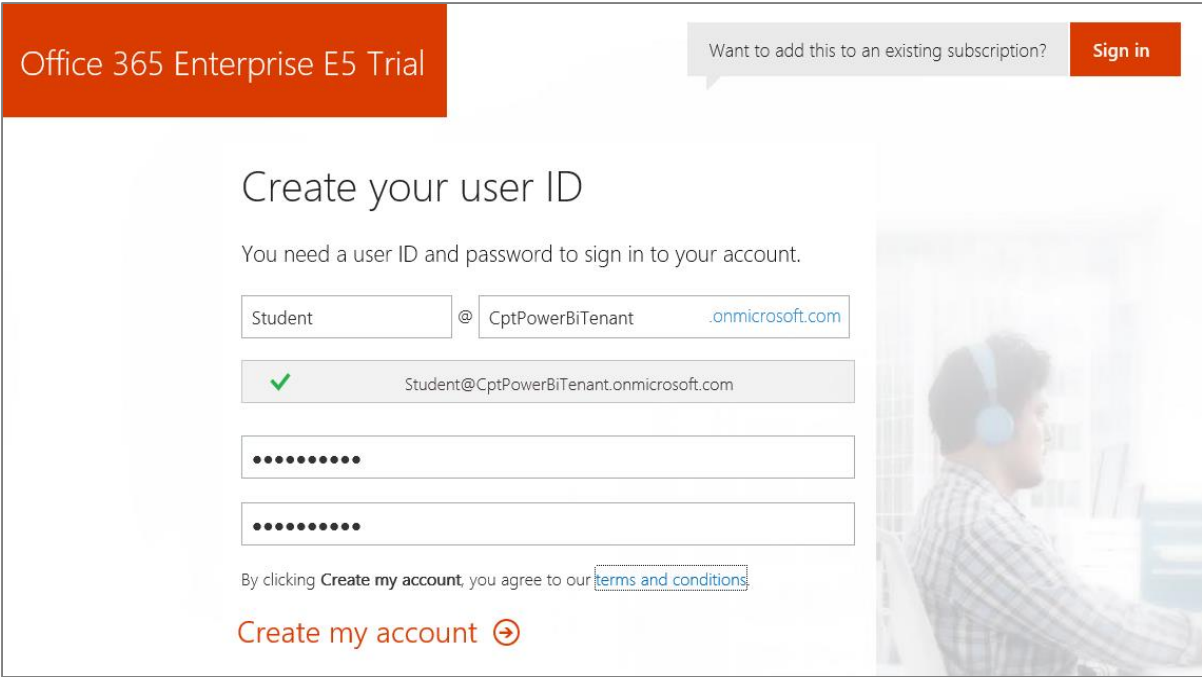
# Getting Started with Cloud Development

- Create an Office 365 trial tenant for development
  - In truth, you are really creating an Azure AD tenant
  - Create global tenant admin user account for developing
  - Create non-admin user accounts for testing
  - Create SharePoint Online sites for developing & testing
  - Obtain a Microsoft Azure subscription
- Getting around inside your Azure AD Tenant
  - Microsoft 365 administrative tools
  - Azure Portal
  - SharePoint admin center
  - PowerShell utilities



# Creating a SharePoint Trial Environment

- Sign up for an Office 365 Enterprise E5 trial account
  - Creates a new Office 365 tenant
  - Creates an account which is tenant administrator
  - You can create 25 user accounts for testing purposes
  - You can create and test Office 365 unified groups



The screenshot shows the 'Office 365 Enterprise E5 Trial' sign-up page. At the top left, there is an orange header with the text 'Office 365 Enterprise E5 Trial'. To the right of this header, there is a grey box with the text 'Want to add this to an existing subscription?' and an orange 'Sign in' button. The main content area has the heading 'Create your user ID' and a subtext 'You need a user ID and password to sign in to your account.' Below this, there are two input fields: the first contains 'Student' and the second contains 'CptPowerBiTenant.onmicrosoft.com'. To the right of these fields is an '@' symbol. Below the input fields, there is a green checkmark icon and the text 'Student@CptPowerBiTenant.onmicrosoft.com'. Below this, there are two password input fields, each containing a series of dots. At the bottom, there is a link 'terms and conditions' and a button 'Create my account' with a right-pointing arrow.

Office 365 Enterprise E5 Trial

Want to add this to an existing subscription? [Sign in](#)

## Create your user ID

You need a user ID and password to sign in to your account.

Student @ CptPowerBiTenant .onmicrosoft.com

✓ Student@CptPowerBiTenant.onmicrosoft.com

.....

.....

By clicking **Create my account**, you agree to our [terms and conditions](#)

**Create my account** ➔



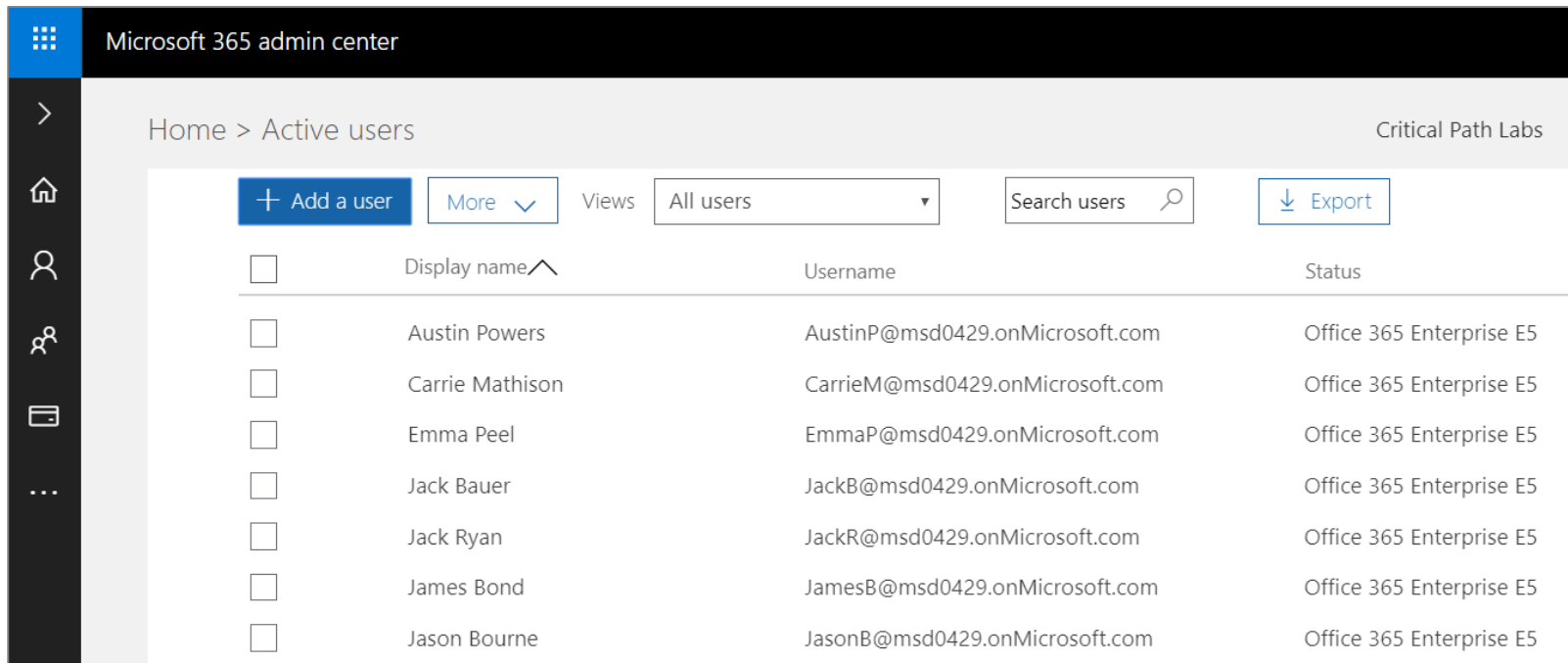
# Office 365 Tenancies in SharePoint Online

- Office 365 environment based on tenancies
  - New tenancy is created for each customer organization
  - Tenancy provides scope for creating users and groups
  - Tenancy provides scope for creating SharePoint sites
  - Tenancy provides scope for Azure AD applications
- Office 365 Developer should be tenant admin
  - Provides permissions you need to develop and test



# Microsoft 365 admin center

- Chores to accomplish in Microsoft 365 admin center
  - Accessible at <https://admin.microsoft.com/Adminportal>
  - Learn how to add secondary user accounts for testing
  - Learn how to view and manage groups



Microsoft 365 admin center

Home > Active users Critical Path Labs

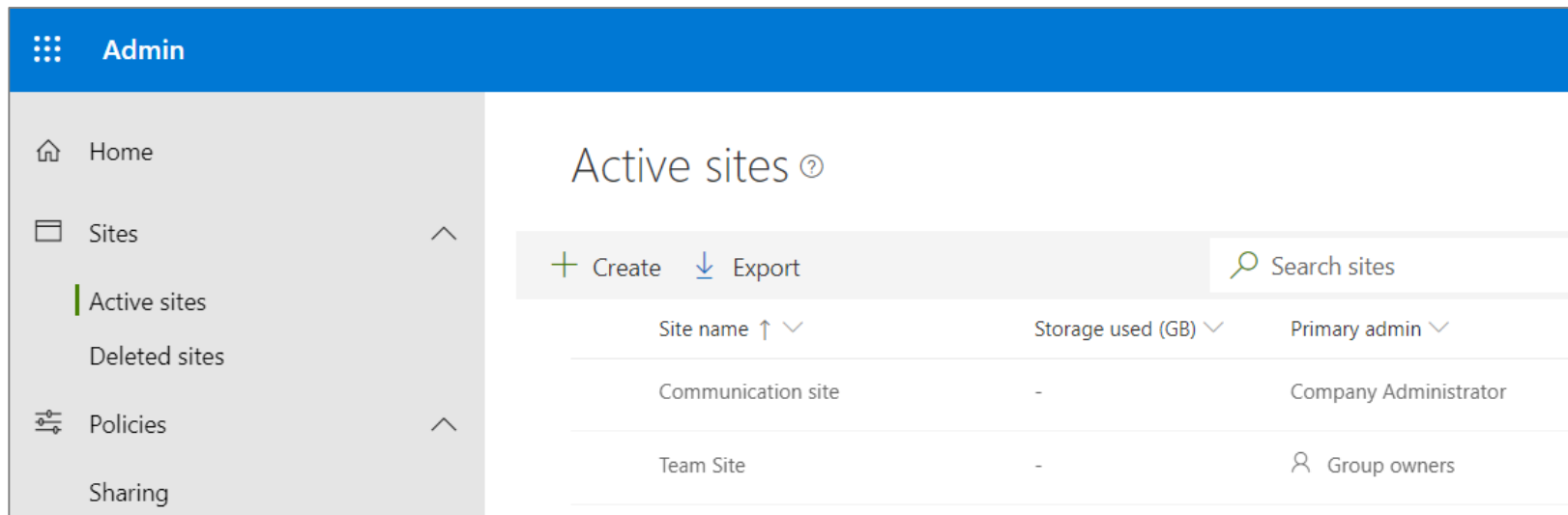
[+ Add a user](#) [More](#) Views All users  [Export](#)

<input type="checkbox"/>	Display name	Username	Status
<input type="checkbox"/>	Austin Powers	AustinP@msd0429.onMicrosoft.com	Office 365 Enterprise E5
<input type="checkbox"/>	Carrie Mathison	CarrieM@msd0429.onMicrosoft.com	Office 365 Enterprise E5
<input type="checkbox"/>	Emma Peel	EmmaP@msd0429.onMicrosoft.com	Office 365 Enterprise E5
<input type="checkbox"/>	Jack Bauer	JackB@msd0429.onMicrosoft.com	Office 365 Enterprise E5
<input type="checkbox"/>	Jack Ryan	JackR@msd0429.onMicrosoft.com	Office 365 Enterprise E5
<input type="checkbox"/>	James Bond	JamesB@msd0429.onMicrosoft.com	Office 365 Enterprise E5
<input type="checkbox"/>	Jason Bourne	JasonB@msd0429.onMicrosoft.com	Office 365 Enterprise E5



# The New SharePoint admin center

- Provides tenant-level administrative features
  - Accessible at [https://\[TENANT\\_NAME\]-admin.sharepoint.com](https://[TENANT_NAME]-admin.sharepoint.com)



The screenshot displays the SharePoint Admin Center interface. On the left is a navigation pane with the following items: Home, Sites (expanded), Active sites (selected), Deleted sites, Policies, and Sharing. The main content area is titled 'Active sites' and includes a '+ Create' button, a '↓ Export' button, and a 'Search sites' search bar. Below these are two rows of site information:

Site name ↑ ↓	Storage used (GB) ↓	Primary admin ↓
Communication site	-	Company Administrator
Team Site	-	Group owners





# SharePoint Online Management Shell

- Connect to admin site using **Connect-SPOService**
- Call SPO cmdlets to query and update SharePoint assets

```
SharePoint Online Management Shell
PS C:\> Connect-SPOService -Url https://msd0429-admin.sharepoint.com
PS C:\> Get-SPOSite

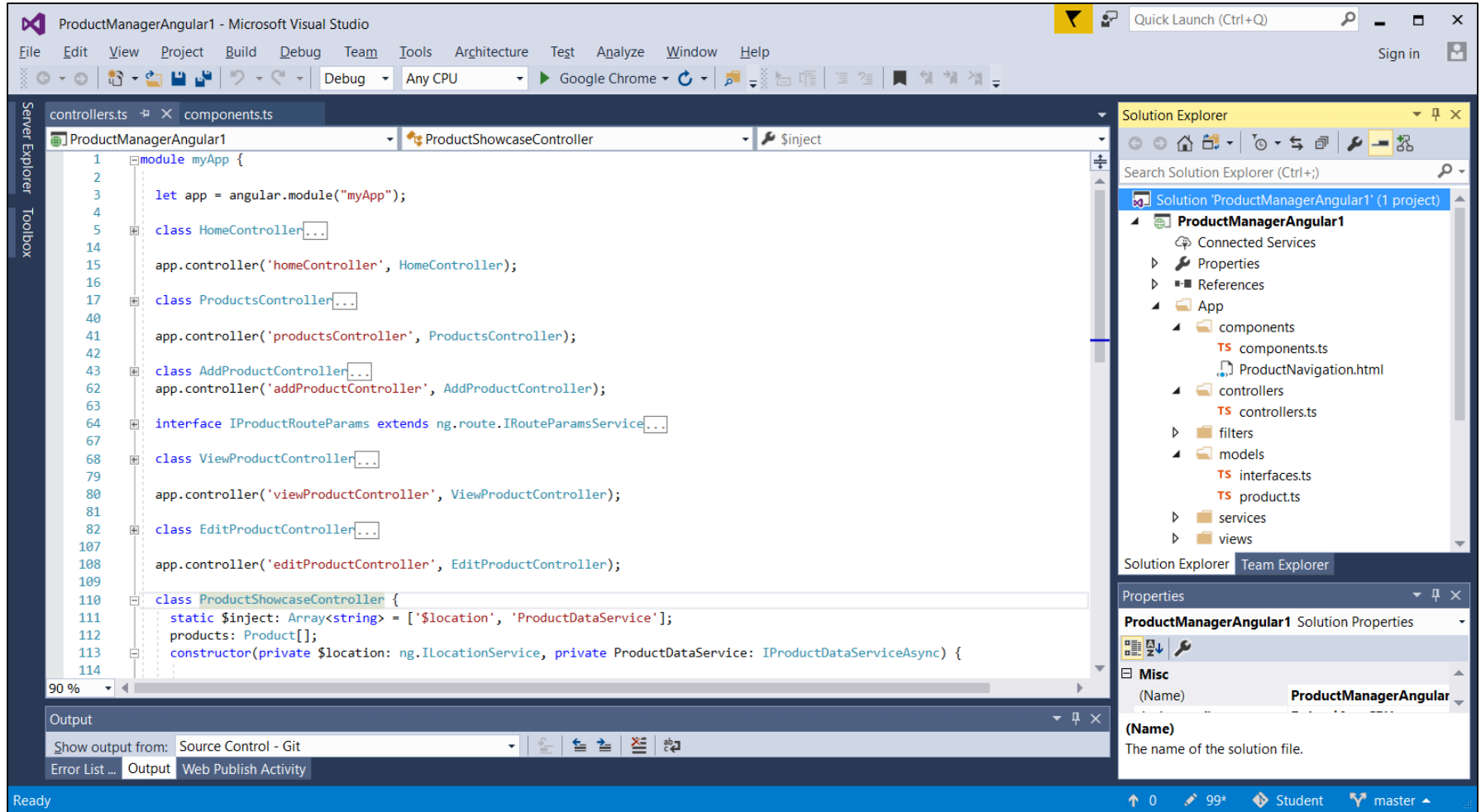
Url                                     Owner Storage Quota
---                                     -
https://msd0429.sharepoint.com/        26214400
https://msd0429.sharepoint.com/portals/Community 26214400
https://msd0429-my.sharepoint.com/     26214400
https://msd0429.sharepoint.com/sites/TeamSite 26214400
https://msd0429.sharepoint.com/search  26214400
https://msd0429.sharepoint.com/portals/hub 26214400
```

- Call **New-SPOSite** to create a new SharePoint site

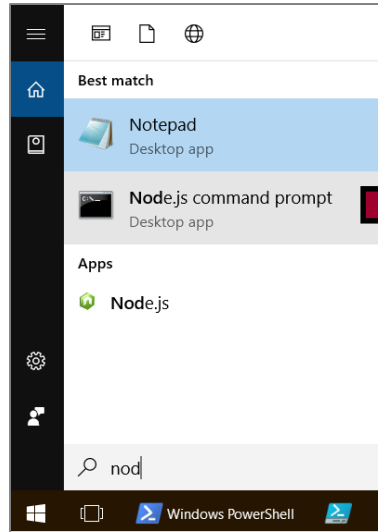
```
SharePoint Online Management Shell
PS C:\> $url = "https://msd0429.sharepoint.com/sites/teamsite2"
PS C:\> $owner = "tedp@msd0429.onmicrosoft.com"
PS C:\> $quota = 0
PS C:\> $siteTitle = "Team Site 2"
PS C:\> $template = "STS#3"
PS C:\> New-SPOSite -Url $url -Owner $owner -Title $siteTitle -StorageQuota $quota -Template $template
```



# Developing with Visual Studio 2017

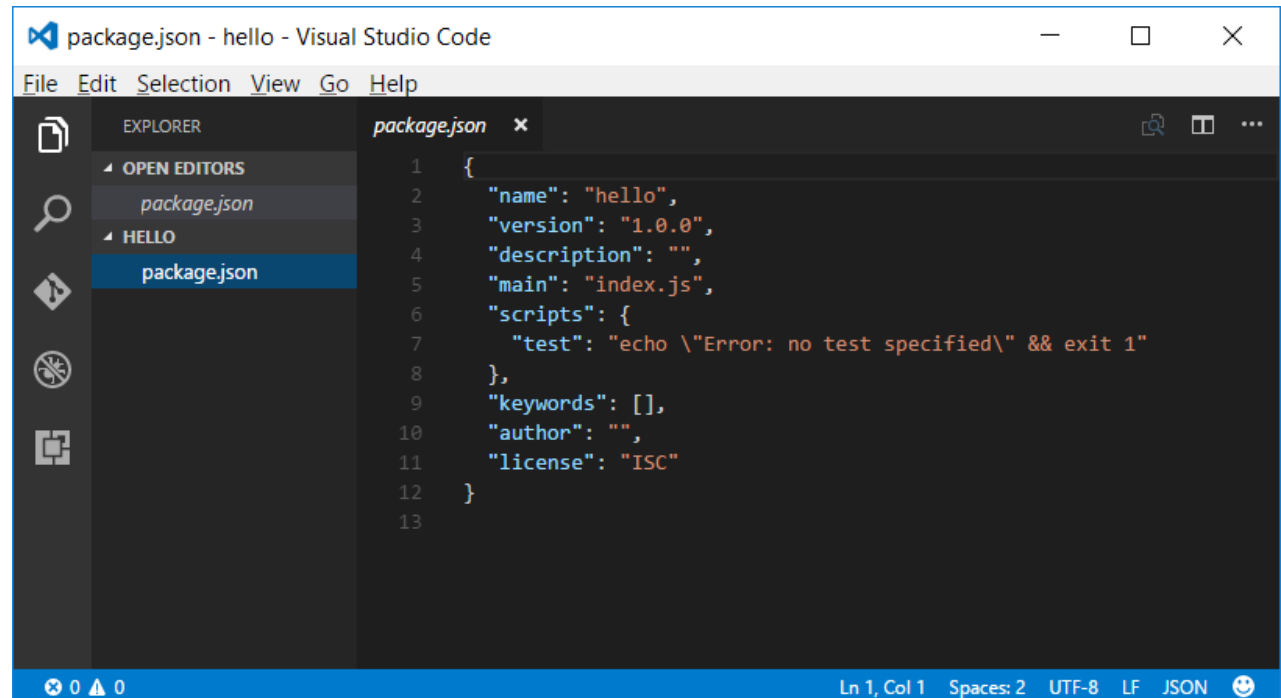


# Developing with NPM & Visual Studio Code



```
Node.js command prompt

C:\Users\Student>mkdir hello
C:\Users\Student>cd hello
C:\Users\Student\hello>npm init -y
C:\Users\Student\hello>code .
```



# Agenda

- ✓ Understanding SharePoint Development Strategies
- ✓ Creating a SharePoint Development Environment
- Programming the Client-side Object Model (CSOM)
  - Understanding Azure as a Development Platform
  - Developing with TypeScript and Interfaces



# Why Client Object Model (CSOM)?

- Advantages of CSOM over the REST API
  - Strongly-typed programming
  - Format Digest managed automatically
  - Higher productivity when writing C# or VB
  - Provides ability to batch requests to web server
  - CSOM provides functionality beyond REST APIs
- CSOM more preferable on server-side C#
  - CSOM isn't best fit for JavaScript apps



# Supported CSOM Functionality

- What can you do with CSOM?
  - Work within a specific site collection
  - Read and modify site properties
  - Create site columns and content types
  - Create lists, items, views and list types
  - Register remote event handlers
  - Create folder and upload and download files
  - Add web part and web part pages
  - Create new site collections



# CSOM in SharePoint Online

- CSOM Assemblies for SharePoint Foundation
  - Version 15 intended for SharePoint 2013 On-premises
  - Version 16.0 intended for SharePoint 2016 On-premises
  - Version 16.1 (or greater) intended for SharePoint Online

- Microsoft.SharePoint.Client
- Microsoft.SharePoint.Client.Runtime

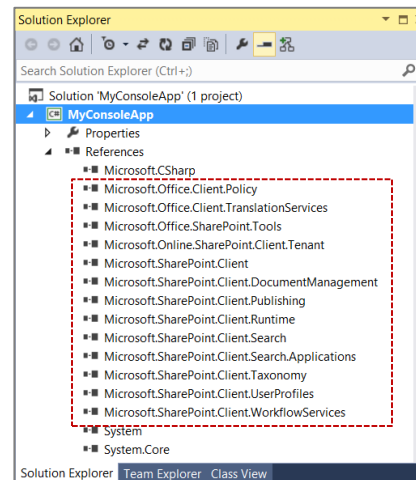
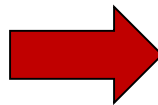
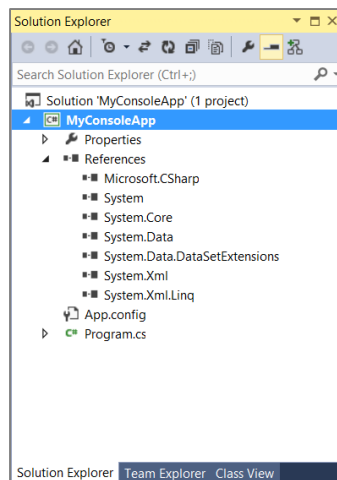
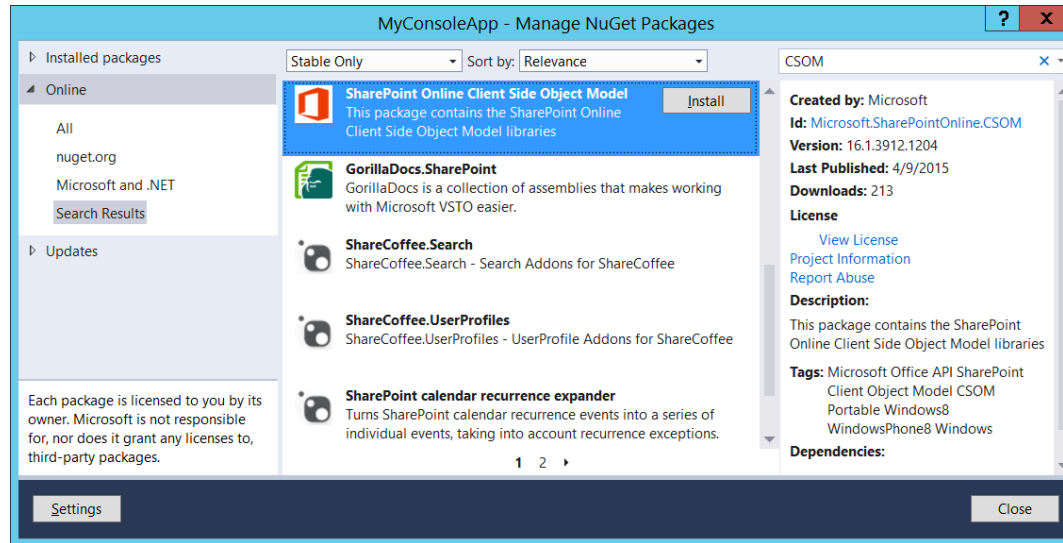
- CSOM Assemblies for SharePoint Server

- Microsoft.SharePoint.Client.DocumentManagement
- Microsoft.SharePoint.Client.Publishing
- Microsoft.SharePoint.Client.Search
- Microsoft.SharePoint.Client.Taxonomy
- Microsoft.SharePoint.Client.UserProfiles
- Microsoft.SharePoint.Client.WorkflowServices



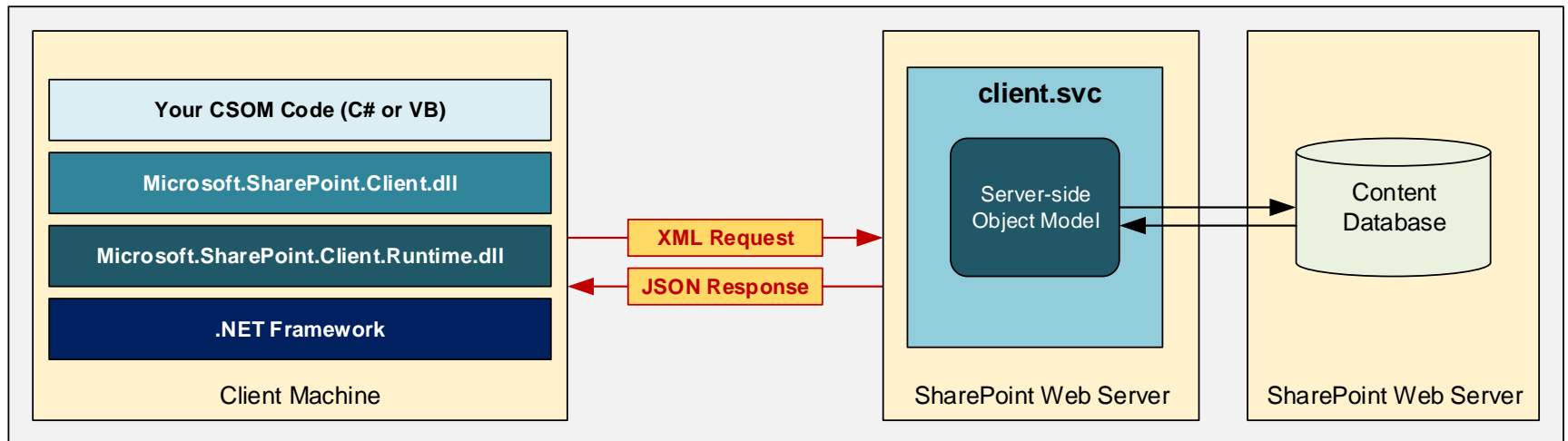


# SPO CSOM NuGet Package



# CSOM Architecture

- CSOM Objects act as client-side proxies
  - CSOM uses Windows Communication Foundation (WCF)
  - CSOM Runtime layer handles WCF calls behind scenes
  - Request body contains XML document of instructions
  - Response returned in JavaScript Object Nation (JSON)



# ClientContext

- CSOM coding starts with ClientContext
  - Provides connection to SharePoint site
  - Provides access to site and site collection
  - Provides authentication behavior
  - Provides ExecuteQuery method to call server

```
string siteUrl = "http://intranet.wingtip.com";  
ClientContext clientContext = new ClientContext(siteUrl);
```



# Hello CSOM

```
using System;
using Microsoft.SharePoint.Client;

namespace HelloCSOM {
    class Program {
        static void Main() {

            ClientContext clientContext = new ClientContext("http://intranet.wingtip.com");

            Site siteCollection = clientContext.Site;
            Web site = clientContext.Web;

            clientContext.Load(siteCollection);
            clientContext.Load(site);

            clientContext.ExecuteQuery();

            Console.WriteLine("The site collection URL is " + siteCollection.Url);
            Console.WriteLine("The site title is " + site.Url);
        }
    }
}
```



# Inspecting CSOM Calls with Fiddler

- ExecuteQuery triggers call to SharePoint web server
  - CSOM calls made behind the scenes using WCF
  - CSOM calls target `/_vti_bin/client.svc/ProcessQuery`
  - Can be helpful to inspect CSOM calls using Fiddler Web Debugger

```
using System;
using Microsoft.SharePoint.Client;

namespace HelloCSOM {
    class Program {
        static void Main() {

            ClientContext clientContext = new ClientContext("http://intranet.wingtip.com");

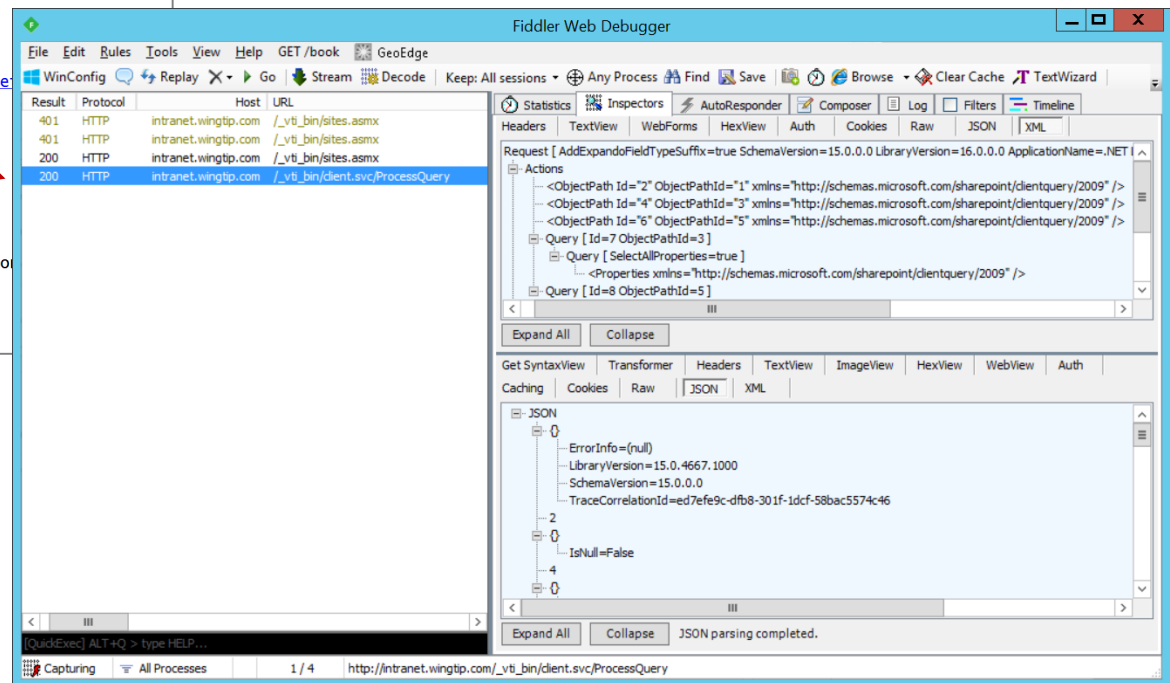
            Site siteCollection = clientContext.Site;
            Web site = clientContext.Web;

            clientContext.Load(siteCollection);
            clientContext.Load(site);

            clientContext.ExecuteQuery();

            Console.WriteLine("The site collection URL is " + siteCollection.Url);
            Console.WriteLine("The site title is " + site.Title);

        }
    }
}
```



# User Authentication (On-premises)

```
string siteUrl = "http://intranet.wingtip.com";
ClientContext clientContext = new ClientContext(siteUrl);

// set up authentication credentials
string userName = @"WINGTIP\Administrator";
string userPassword = "Password1";
clientContext.Credentials = new NetworkCredential(userName, userPassword);

// get title of the target site
Web site = clientContext.Web;
clientContext.Load(site);

// call across network
clientContext.ExecuteQuery();

// display title
Console.WriteLine(site.Title);
```



# User Authentication (SPO)

```
string siteUrl = "https://SharepointConfessions.sharepoint.com";
ClientContext clientContext = new ClientContext(siteUrl);

string userName = "tedp@sharepointconfessions.onmicrosoft.com";
string userPassword = "PinkieDoo@42";
// convert password to SecureString format
SecureString secureUserPassword = new SecureString();
foreach (char c in userPassword.ToCharArray()) {
    secureUserPassword.AppendChar(c);
}

// create SharePointOnlineCredentials object to authenticate
clientContext.Credentials =
    new SharePointOnlineCredentials(userName, secureUserPassword);

// get title of the target site
Web site = clientContext.Web;
clientContext.Load(site);

// call across network
clientContext.ExecuteQuery();

// display title
Console.WriteLine(site.Title);
```





# Coding with Lambda Expressions

- C# supports the use of lambda expressions
  - Syntax Introduced as part of LINQ with .NET 3.5
  - Can (and should) be used with CSOM
- Lambda expression is anonymous function
  - It defines a parameter list and a function body

```
clientContext.Load(site, s => s.Title );
```

Input Parameter(s)

Lambda Operator

Statement Block



# Using Lambda Expressions

- Loading an object populates all scalar property values
  - Can result in inefficient use of network bandwidth

```
web site = clientContext.Web;  
clientContext.Load(site);  
clientContext.ExecuteQuery();
```



```
{  
  "_ObjectType": "SP.Web",  
  "AllowRssFeeds": true,  
  "AppInstanceId": "Guid(00000000-0000-0000-0000-000000000000)",  
  "Configuration": 0,  
  "Created": "Date(2013/5/31, 3, 53, 32, 0)",  
  "CustomMasterUrl": "/_catalogs/masterpage/seattle.master",  
  "Description": "",  
  "DocumentLibraryCalloutOfficeWebAppPreviewersDisabled": false,  
  "EnableMinimalDownload": true,  
  "Id": "Guid(8e70e4a1-7528-4822-ac08-45a443d31bbd)",  
  "Language": 1033,  
  "LastItemModifiedDate": "Date(1379086272000)",  
  "MasterUrl": "/_catalogs/masterpage/seattle.master",  
  "QuickLaunchEnabled": true,  
  "RecycleBinEnabled": true,  
  "ServerRelativeUrl": "",  
  "SyndicationEnabled": true,  
  "Title": "Wingtip Team Site",  
  "TreeViewEnabled": false,  
  "UIVersion": 15,  
  "UIVersionConfigurationEnabled": false,  
  "Url": "http://wingtipserver",  
  "WebTemplate": "STS"  
}
```

- Lambda expressions can be used to optimize
  - You can indicate which properties you want populated

```
web site = clientContext.Web;  
clientContext.Load(site, s => s.Title);  
clientContext.ExecuteQuery();
```



```
{  
  "_ObjectType": "SP.Web",  
  "Title": "Wingtip Team Site"  
}
```



# Using Where() and Include()

- Where lets you pass filter criteria to server

```
// instead of this
clientContext.Load(site.Lists);

// use this instead
clientContext.Load(site.Lists, lists => lists.Where(list => !list.Hidden));
```

- Include lets you pick fields on item in a collection

```
// indicate which list properties you want to populate for each list
clientContext.Load(site.Lists,
    lists => lists.Include(list => list.Title, list => list.DefaultViewUrl));
```

- Syntax is powerful but tricky to read and write

```
ListCollection Lists = clientContext.Web.Lists;
clientContext.Load(Lists, lists => lists.Where(list => !list.Hidden)
    .Include(list => list.Title,
        list => list.DefaultViewUrl));

clientContext.ExecuteQuery();
```



# Creating a List

```
Web site = clientContext.Web;
clientContext.Load(site);

// create and initialize ListCreationInformation object
ListCreationInformation listInformation = new ListCreationInformation();
listInformation.Title = "Announcements";
listInformation.Url = "Lists/Announcements";
listInformation.QuickLaunchOption = QuickLaunchOptions.On;
listInformation.TemplateType = (int)ListTemplateType.Announcements;

// Add ListCreationInformation to lists collection and return list object
List list = site.Lists.Add(listInformation);

// modify additional list properties and update
list.OnQuickLaunch = true;
list.EnableAttachments = false;
list.Update();

// send command to server to create list
clientContext.ExecuteQuery();
```



# Creating List Items

```
ListItemCreationInformation lici = new ListItemCreationInformation();

var item1 = list.AddItem(lici);
item1["Title"] = "SharePoint introduces new app model";
item1["Body"] = "<div>Developers wonder what happened to solutions.</div>";
item1["Expires"] = DateTime.Today.AddYears(10);
item1.Update();

var item2 = list.AddItem(lici);
item2["Title"] = "All SharePoint developers must now learn JavaScript";
item2["Body"] = "<div>Some developers are more excited then others.</div>";
item2["Expires"] = DateTime.Today.AddYears(1);
item2.Update();

var item3 = list.AddItem(lici);
item3["Title"] = "CSOM programming is super fun";
item3["Body"] = "<div>Just ask my mom.</div>";
item3["Expires"] = DateTime.Today.AddDays(7);
item3.Update();

clientContext.ExecuteQuery();
```



# Creating Site Columns - Part 1

```
static Field CreateSiteColumn(string fieldName, string fieldDisplayName, string fieldType) {  
    Console.WriteLine("Creating " + fieldName + " site column...");  
  
    // delete existing field if it exists  
    try {  
        Field fld = site.Fields.GetByInternalNameOrTitle(fieldName);  
        fld.DeleteObject();  
        clientContext.ExecuteQuery();  
    }  
    catch { }  
  
    string fieldXML = @"<Field Name='" + fieldName + "' " +  
        "DisplayName='" + fieldDisplayName + "' " +  
        "Type='" + fieldType + "' " +  
        "Group='wingtip' > " +  
        "</Field>";  
  
    Field field = site.Fields.AddFieldAsXml(fieldXML, true, AddFieldOptions.DefaultValue);  
    clientContext.Load(field);  
    clientContext.ExecuteQuery();  
    return field;  
}
```



# Creating Site Columns - Part 2

```
fieldProductCode = CreateSiteColumn("ProductCode", "Product Code", "Text");
fieldProductCode.EnforceUniqueValues = true;
fieldProductCode.Indexed = true;
fieldProductCode.Required = true;
fieldProductCode.Update();
clientContext.ExecuteQuery();
clientContext.Load(fieldProductCode);
clientContext.ExecuteQuery();

fieldProductDescription =
    clientContext.CastTo<FieldMultiLineText>(CreateSiteColumn("ProductDescription", "Product Description", "Note"));
fieldProductDescription.NumberOfLines = 4;
fieldProductDescription.RichText = false;
fieldProductDescription.Update();
clientContext.ExecuteQuery();

fieldProductListPrice =
    clientContext.CastTo<FieldCurrency>(CreateSiteColumn("ProductListPrice", "List Price", "Currency"));
fieldProductListPrice.MinimumValue = 0;
fieldProductListPrice.Update();
clientContext.ExecuteQuery();

fieldProductCategory =
    clientContext.CastTo<TaxonomyField>(CreateSiteColumn("ProductCategory", "Product Category", "TaxonomyFieldType"));
fieldProductCategory.SspId = localTermStoreId;
fieldProductCategory.TermSetId = termSetId;
fieldProductCategory.AllowMultipleValues = false;
fieldProductCategory.Update();
clientContext.ExecuteQuery();

fieldProductColor =
    clientContext.CastTo<FieldMultiChoice>(CreateSiteColumn("ProductColor", "Product Color", "MultiChoice"));
string[] choicesProductColor = { "White", "Black", "Grey", "Blue", "Red", "Green", "Yellow" };
fieldProductColor.Choices = choicesProductColor;
fieldProductColor.Update();
clientContext.ExecuteQuery();
```





# Creating Content Types - Part 1

```
static ContentType CreateContentType(string contentTypeName, string baseContentType) {
    DeleteContentType(contentTypeName);

    ContentTypeCreationInformation contentTypeCreateInfo = new ContentTypeCreationInformation();
    contentTypeCreateInfo.Name = contentTypeName;
    contentTypeCreateInfo.ParentContentType = site.ContentTypes.GetById(baseContentType); ;
    contentTypeCreateInfo.Group = "wingtip";
    ContentType ctype = site.ContentTypes.Add(contentTypeCreateInfo);
    clientContext.ExecuteQuery();
    return ctype;
}

static void DeleteContentType(string contentTypeName) {
    try {
        foreach (var ct in site.ContentTypes) {
            if (ct.Name.Equals(contentTypeName)) {
                ct.DeleteObject();
                Console.WriteLine("Deleting existing " + ct.Name + " content type...");
                clientContext.ExecuteQuery();
                break;
            }
        }
    }
    catch { }
}
```



# Creating Content Types - Part 2

```
ctypeProduct = CreateContentType("Product", "0x01");

// add site columns
FieldLinkCreationInformation fieldLinkProductCode = new FieldLinkCreationInformation();
fieldLinkProductCode.Field = fieldProductCode;
ctypeProduct.FieldLinks.Add(fieldLinkProductCode);
ctypeProduct.Update(true);

FieldLinkCreationInformation fieldLinkProductDescription = new FieldLinkCreationInformation();
fieldLinkProductDescription.Field = fieldProductDescription;
ctypeProduct.FieldLinks.Add(fieldLinkProductDescription);
ctypeProduct.Update(true);

FieldLinkCreationInformation fieldLinkProductListPrice = new FieldLinkCreationInformation();
fieldLinkProductListPrice.Field = fieldProductListPrice;
ctypeProduct.FieldLinks.Add(fieldLinkProductListPrice);
ctypeProduct.Update(true);

FieldLinkCreationInformation fieldLinkProductCategory = new FieldLinkCreationInformation();
fieldLinkProductCategory.Field = fieldProductCategory;
ctypeProduct.FieldLinks.Add(fieldLinkProductCategory);
ctypeProduct.Update(true);

FieldLinkCreationInformation fieldLinkProductColor = new FieldLinkCreationInformation();
fieldLinkProductColor.Field = fieldProductColor;
ctypeProduct.FieldLinks.Add(fieldLinkProductColor);
ctypeProduct.Update(true);

clientContext.ExecuteQuery();
```



# Creating List with Content Type

```
ListCreationInformation listInformationProducts = new ListCreationInformation();
listInformationProducts.Title = "Products";
listInformationProducts.Url = "Lists/Products";
listInformationProducts.QuickLaunchOption = QuickLaunchOptions.On;
listInformationProducts.TemplateType = (int)ListTemplateType.GenericList;
listProducts = site.Lists.Add(listInformationProducts);
listProducts.OnQuickLaunch = true;
listProducts.Update();

clientContext.Load(listProducts);
clientContext.Load(listProducts.ContentTypes);
clientContext.ExecuteQuery();

// configure list to use custom content type
listProducts.ContentTypesEnabled = true;
listProducts.ContentTypes.AddExistingContentType(ctypeProduct);
ContentType existing = listProducts.ContentTypes[0]; ;
existing.DeleteObject();
listProducts.Update();
clientContext.ExecuteQuery();

// add custom site columns to default view of list
View viewProducts = listProducts.DefaultView;
viewProducts.ViewFields.Add("ProductCode");
viewProducts.ViewFields.Add("ProductListPrice");
viewProducts.ViewFields.Add("ProductCategory");
viewProducts.ViewFields.Add("ProductColor");
viewProducts.Update();

clientContext.ExecuteQuery();
```



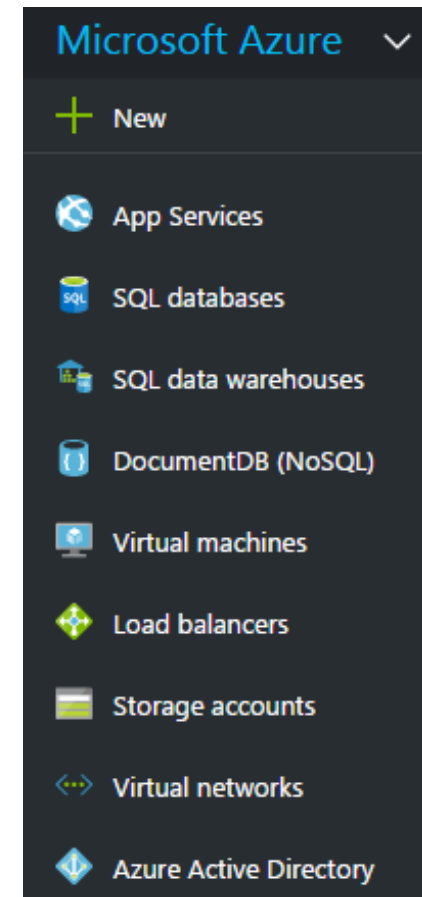
# Agenda

- ✓ Understanding SharePoint Development Strategies
- ✓ Creating a SharePoint Development Environment
- ✓ Programming the Client-side Object Model (CSOM)
- Understanding Azure as a Development Platform
  - Developing with TypeScript and Interfaces



# Azure Services Overview

- Azure provides PaaS, DaaS and IaaS Services
  - App Service Plans and Web Apps
  - SQL databases
  - Virtual machines
  - Storage accounts
  - Virtual networks
  - Load balancers
  - Cloud Services
  - Azure Active Directory
  - Azure Functions



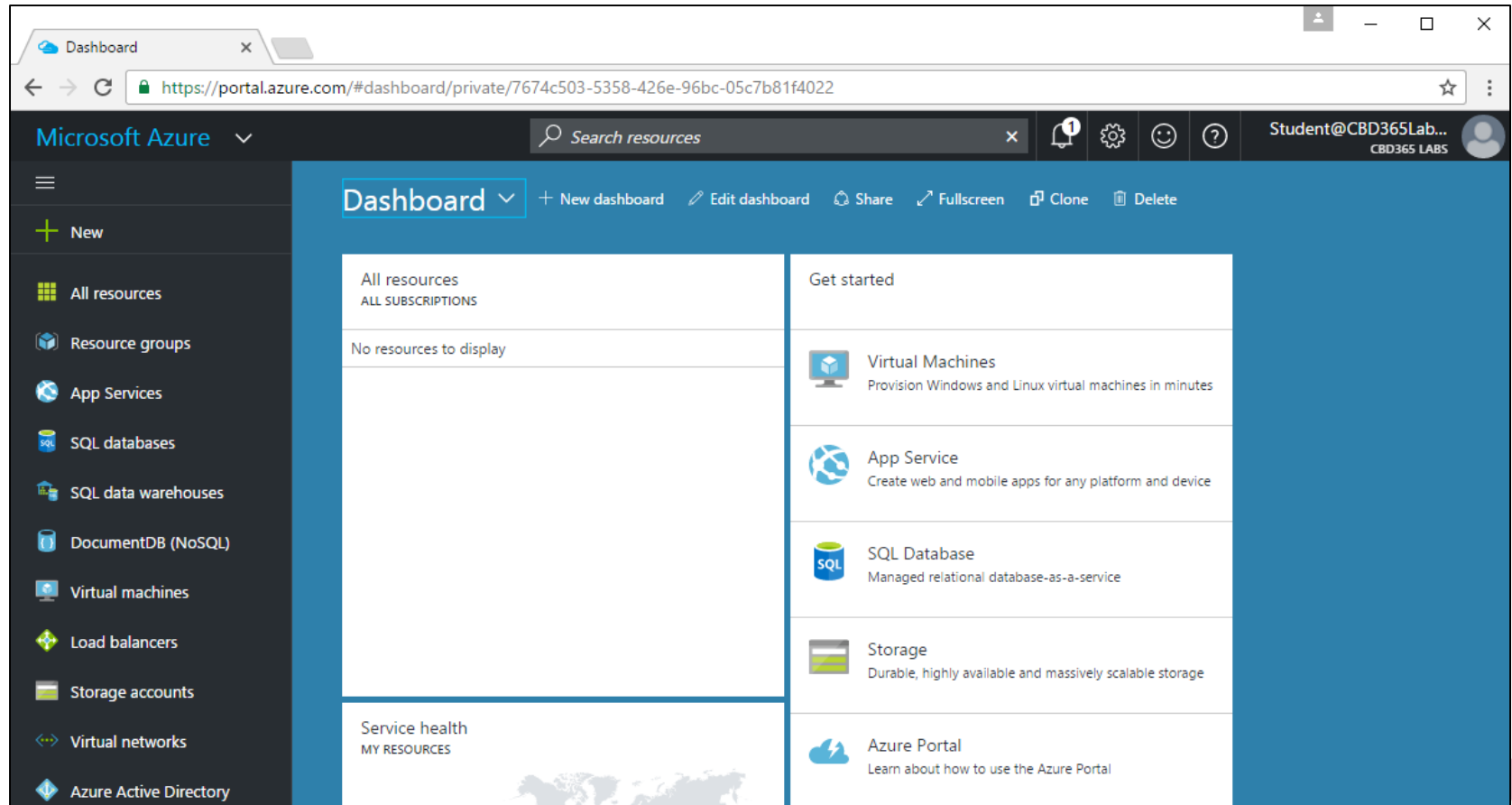
# Obtaining an Azure Subscription

- Getting an Azure Subscription
  - Sign up with paid-for account
  - Get free Azure subscription with a MSDN Subscription
  - Sign up for free 30-day trial account
- Signing up for free trial account
  - Navigate to Azure Portal using Office 365 credentials
  - When prompted, sign up for a trial



# Azure Portal

- You can work with Azure using the new portal
  - Uses newer Resource Manager infrastructure
  - Located at <https://portal.azure.com>



# Agenda

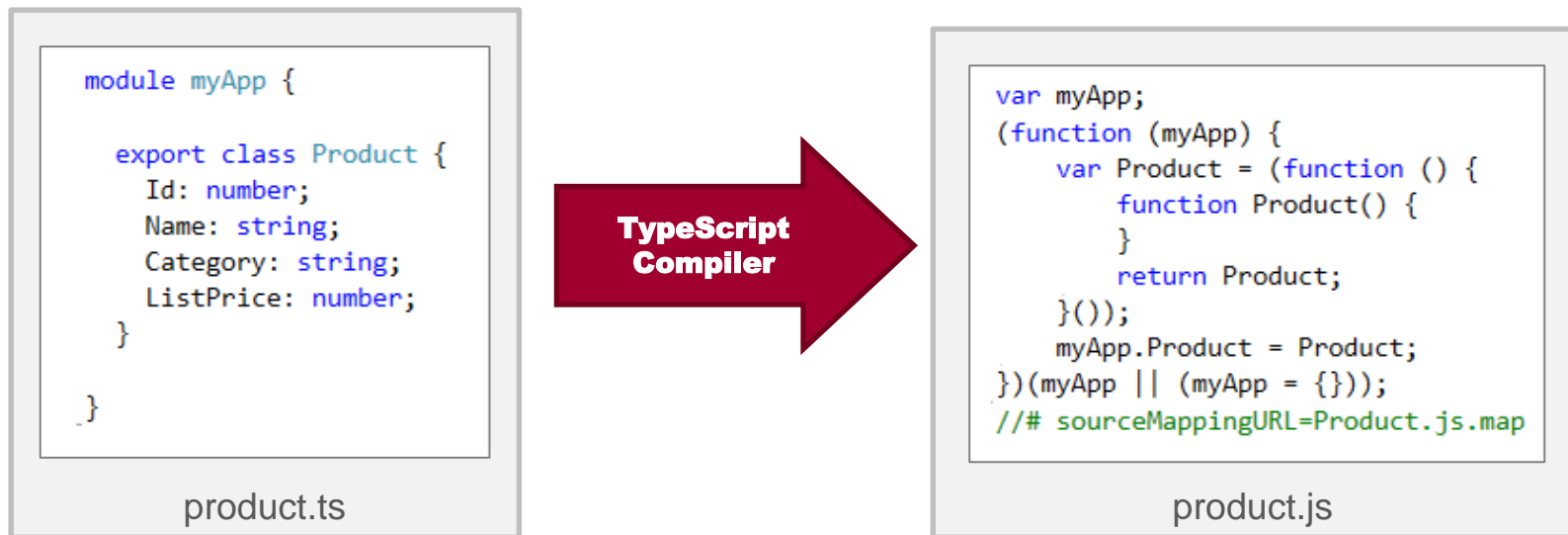
- ✓ Understanding SharePoint Development Strategies
- ✓ Creating a SharePoint Development Environment
- ✓ Programming the Client-side Object Model (CSOM)
- ✓ Understanding Azure as a Development Platform
- Developing with TypeScript and Interfaces





# What is TypeScript?

- A programming language which compiles into plain JavaScript
- A superset of JavaScript that adds a strongly-typed dimension
- It can be compiled into ECMAScript3, ECMAScript3 or ECMAScript 6
- It runs in any browser, in any host and on any OS



# Type Annotation

- TypeScript allows you to annotate types
  - Provides basis for strongly-typed programming
  - Type annotations used by compiler for type checking
  - Type annotations are erased at the end of compile time

```
// define strongly-typed function
var myFunction = function (param1: number): string {
  | return "You passed " + param1;
};

// define strongly-typed variables
var myNumber: number = 2017;
var myMessage: string = myFunction(myNumber);
var myContent: JQuery = $("<p>").text(myMessage);
var contentBox: JQuery = $("#content-box");

contentBox.empty().append(myContent);
```



# Assignment with let versus var

- var does not recognize nor honor scope
- let will recognize and honor scope

```
var x:number = 2016;  
let y: number = 2016;  
  
{  
  var x:number = 2017;  
  let y:number = 2017;  
}  
  
let message = "x=" + x + " and " + "y=" + y;
```



x=2017 and y=2016



# Arrow Function Syntax

- TypeScript supports arrow function syntax
  - Concise syntax to define anonymous functions
  - Can be used to retain this pointer in classes

```
// create anonymous function using function arrow syntax
let myFunction = () => {
  console.log("Hello world");
};

// use function arrow syntax with typed parameters
let myOtherFunction = (param1: number, param2: string) : string => {
  return param1 + " - " + param2;
};

// create function to assign to DOM event
window.onresize = (event: Event) => {
  let window: Window = event.target as Window;
  console.log("Window width: " + window.outerWidth);
  console.log("Window height: " + window.outerHeight);
};
```



# Classes

- TypeScript supports defining classes
  - Class defines type for object
  - Export keyword makes class created across files
  - Class can be passed as factory function
  - Default accessibility is public

```
export class Product {  
  Id: number;  
  Name: string;  
  Category: string;  
  ListPrice: number;  
}
```

```
// create new Product instance  
let product1: Product = new Product();  
product1.Id = 1;  
product1.Name = "Batman Action Figure";  
product1.Category = "Action Figure";  
product1.ListPrice = 14.95;
```



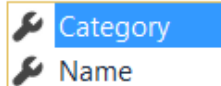
# Class Constructors

- Constructor parameters become fields in class

```
export class Product {  
  
    constructor(private Id: number, public Name: string, public Category: string, private ListPrice: number) {  
        // no need to do anything here  
    }  
  
    MyPublicMethod() {  
        // access to private fields  
        let id: number = this.Id  
        let price: number = this.ListPrice  
    }  
}
```

- Client-side code calls constructor using new operator

```
// create new Product instance  
let product1: Product = new Product(1, "Batman Action Figure", "Action Figure", 14.95);  
  
// access public properties  
let product1Name: string = product1.Name;  
let product1Category: string = product1.
```



# Interfaces

- Interface defines a programming contract
  - Classes can implement interfaces

```
export interface IProductDataService {  
  GetAllProducts(): Product[];  
  GetProduct(id: number): Product;  
  AddProduct(product: Product): void;  
  DeleteProduct(id: number): void;  
  UpdateProduct(product: Product): void;  
}
```

```
export class MyProductDataService implements IProductDataService {  
  
  private products: Product[] = [];  
  
  GetAllProducts(): Product[] {  
    return this.products;  
  }  
  
  GetProduct(id: number): Product {  
    return this.products.find(p => p.id === id);  
  }  
  
  AddProduct(product: Product): void {  
    this.products.push(product);  
  }  
  
  DeleteProduct(id: number): void {  
    this.products = this.products.filter(p => p.id !== id);  
  }  
  
  UpdateProduct(product: Product): void {  
    const index = this.products.findIndex(p => p.id === product.id);  
    this.products[index] = product;  
  }  
}
```

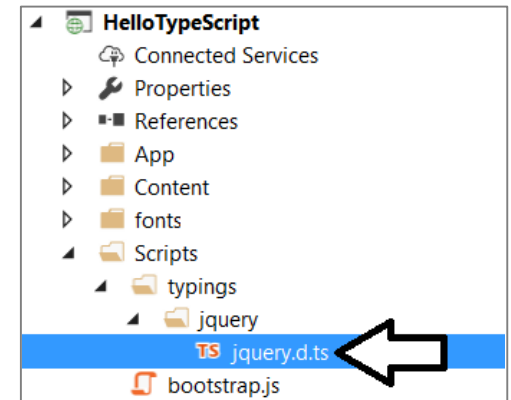
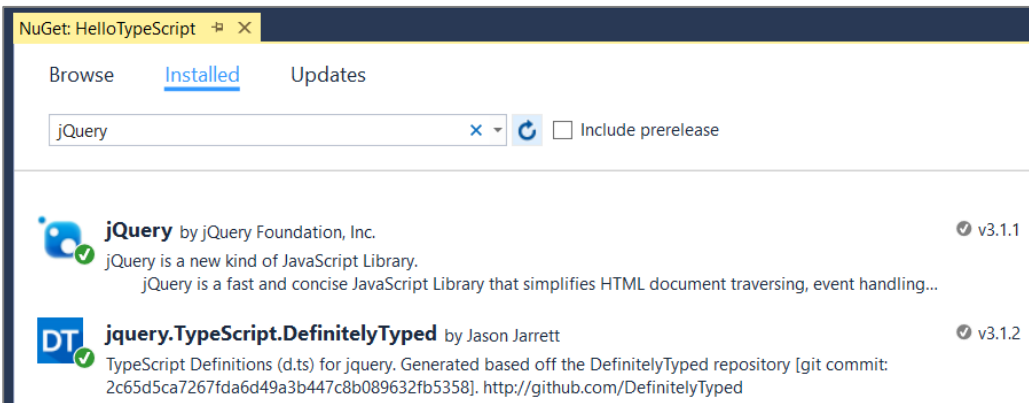
- Client code can be decoupled from concrete classes

```
// program against variables based on interface type  
let productService: IProductDataService = new MyProductDataService();  
  
// client code is decoupled from underlying data access class implementations  
let products: Product[] = productService.GetAllProducts();  
let product1: Product = productService.GetProduct(1);
```



# TypeScript Definition Files (d.ts)

- What are TypeScript definition files
  - Typed definitions for 3rd party JavaScript libraries
  - DefinitelyTyped provides great community resource
  - Typed definition files have a **d.ts** extension



```
// define strongly-typed variables
var myNumber: number = 2017;
var myMessage: string = myFunction(myNumber);
var myContent: JQuery = $("<p>").text(myMessage);
var contentBox: JQuery = $("#content-box");
```





# Interface-based Design

- Interfaces define programming contracts

```
export interface IViewPort {  
  width: number;  
  height: number;  
}
```

```
export interface ICustomVisual {  
  name: string;  
  load(container: HTMLElement): void;  
  update(viewport: IViewPort): void;  
}
```

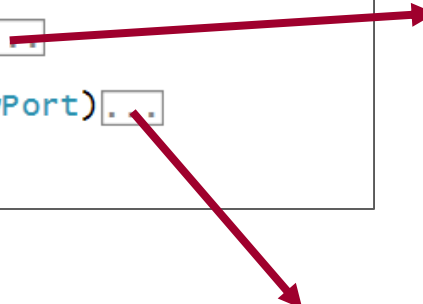
- Application design can use interfaces instead of concrete classes

```
module myApp {  
  
  var leftNavCollapsed: boolean = true;  
  var loadedVisual: ICustomVisual;  
  
  var visuals: ICustomVisual[] = [  
    new Viz01(), new Viz02(), new Viz03(), new Viz04()  
  ];  
  
  function LoadVisual(visual: ICustomVisual) ...  
  
  $(( ) => ...);  
}
```



# Sample Custom Visual using jQuery

```
export class Viz01 implements ICustomVisual {  
    public name: string = "Visual 1: Hello jQuery";  
    private container: JQuery;  
    private message: JQuery;  
  
    load(container: HTMLElement) {  
        ...  
    }  
  
    public update(viewport: IViewport) {  
        ...  
    }  
}
```



```
load(container: HTMLElement) {  
    this.container = $(container);  
    this.message = $("

")  
        .text("Hello jQuery")  
        .css({  
            "display": "table-cell",  
            "text-align": "center",  
            "vertical-align": "middle",  
            "text-wrap": "none",  
            "background-color": "yellow"  
        });  
    this.container.append(this.message);  
}


```

```
public update(viewport: IViewport) {  
    let paddingX: number = 2;  
    let paddingY: number = 2;  
    let fontSizeMultiplierX: number = viewport.width * 0.15;  
    let fontSizeMultiplierY: number = viewport.height * 0.4;  
    let fontSizeMultiplier: number = Math.min(...[fontSizeMultiplierX,  
                                                    fontSizeMultiplierY]);  
  
    this.message.css({  
        "width": viewport.width - paddingX,  
        "height": viewport.height - paddingY,  
        "font-size": fontSizeMultiplier  
    });  
}
```

# Summary

- ✓ Understanding SharePoint Development Strategies
- ✓ Creating a SharePoint Development Environment
- ✓ Programming the Client-side Object Model (CSOM)
- ✓ Understanding Azure as a Development Platform
- ✓ Developing with TypeScript and Interfaces

