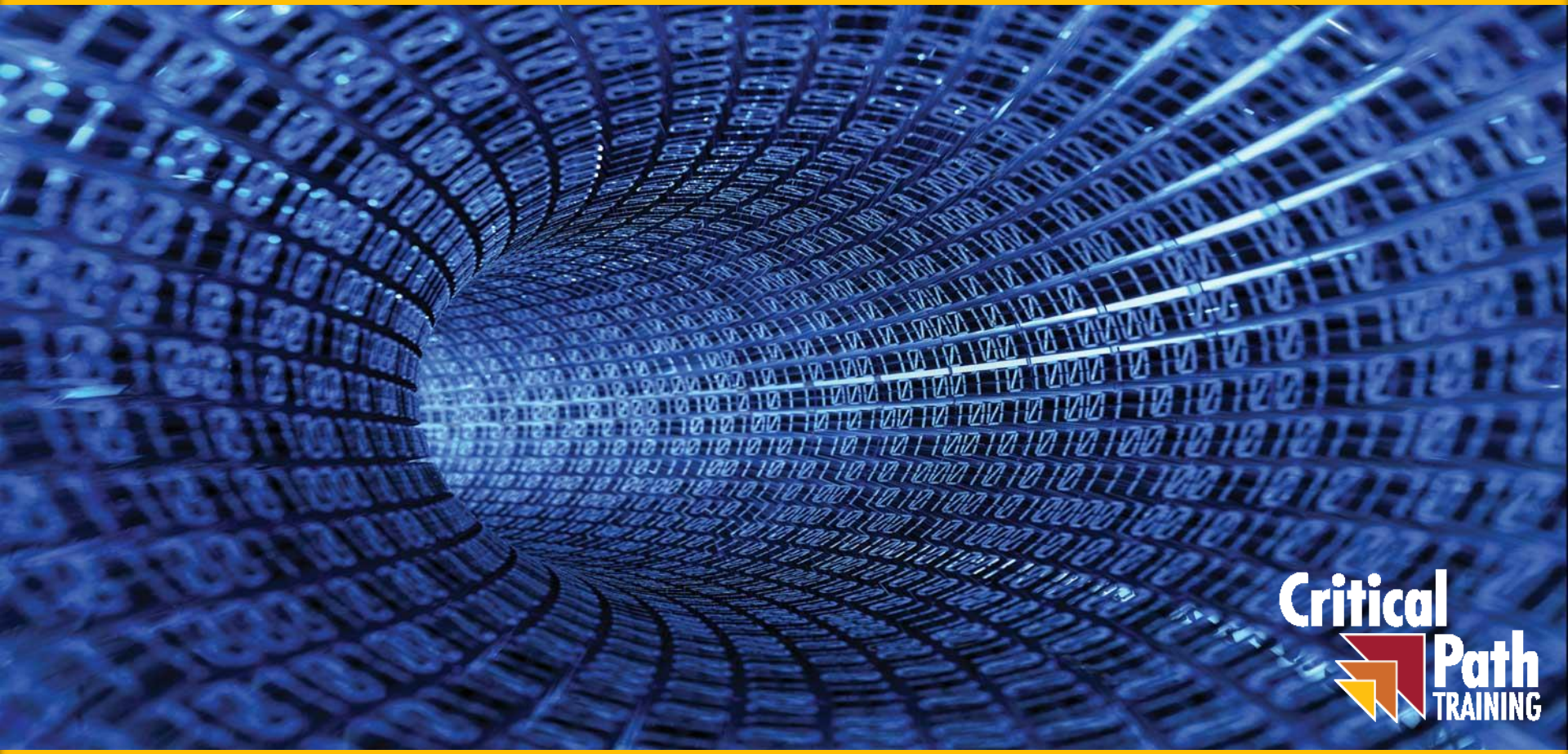
Developing SharePoint Add-ins



Agenda

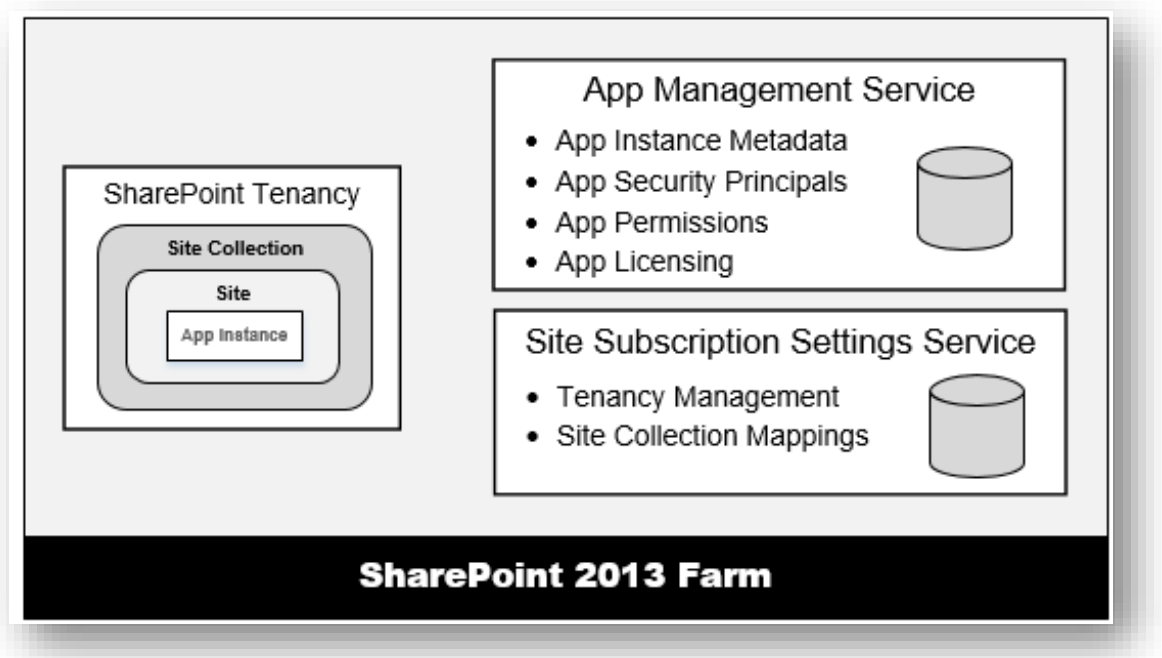
- SharePoint Add-in Model Overview
- SharePoint-hosted vs Provider-hosted Add-ins
- SharePoint Add-in Security
- Extending an Add-in with Permission Request
- Acquiring and Managing Access Tokens

SharePoint Tenancies

- A tenancy is a set of site collections
  - Configured and administrated as a unit
  - Created with administrative site collection
  - A scope for provisioning new site collection
  - Central concept to site management in Office 365
  - A requirement for installing SharePoint apps
- What about tenancies in on-premises farms?
  - Most farms do not have explicitly created tenancies
  - To add support for SharePoint apps, on-premises farm can be configured with a farm-wide default tenancy

Service Application Support for Add-ins

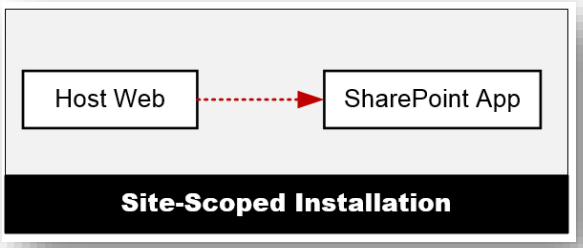
- App support requires two service applications
  - App Management Service
  - Site Subscription Management Service

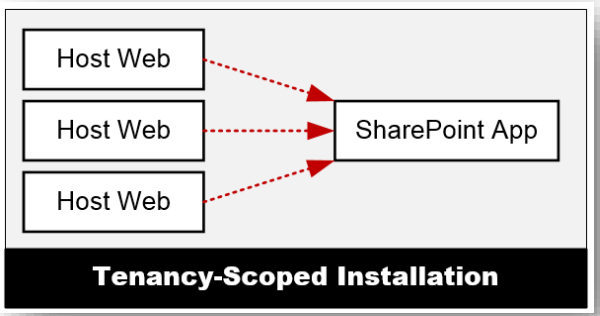


  - These services must be created in on-premises farms to support add-ins

Add-in Installation Scopes

- Site-scoped Installation
  - Add-in installed in SharePoint site which becomes **host web**
  - Add-in can be installed multiple times across site collections
  - Each installed instance of an add-in gets its own app web

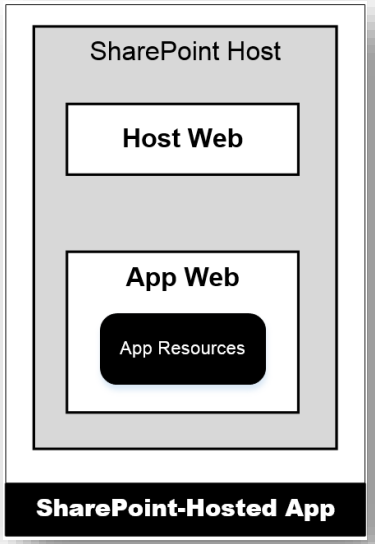




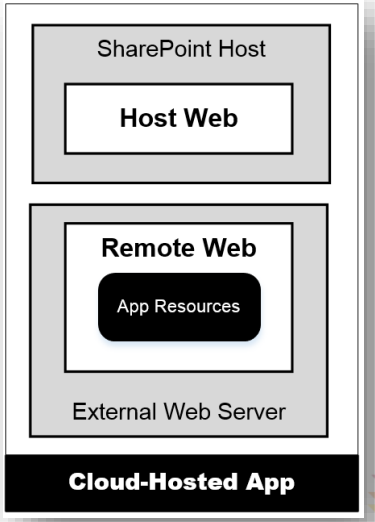
- Tenancy-scoped Installation
  - Provides centralized approach to app deployment & management
  - Requires app to first be installed in an app catalog site
  - Once installed, the app is then configured for use multiple sites
  - Tenancy install scoped to web application in on-premises farms

Hosting Options for SharePoint Add-ins

- SharePoint-Hosted Add-ins
  - App resources added to SharePoint host
  - Stored in child site known as **app web**
  - Add-in can have only client-side code
  - Add-in cannot have server-side code

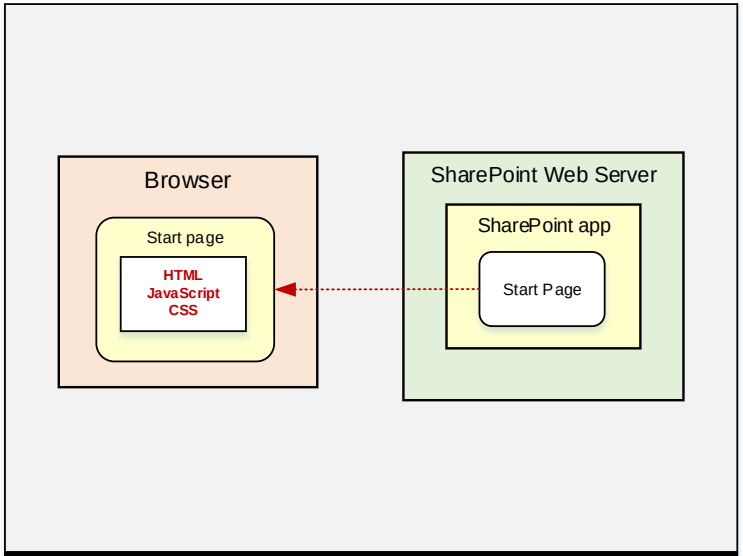


- Provider-Hosted Add-ins
  - Add-in pages deployed to remote server
  - Remote site known as **remote web**
  - Add-in can have client-side code
  - Add-in can have server-side .NET code

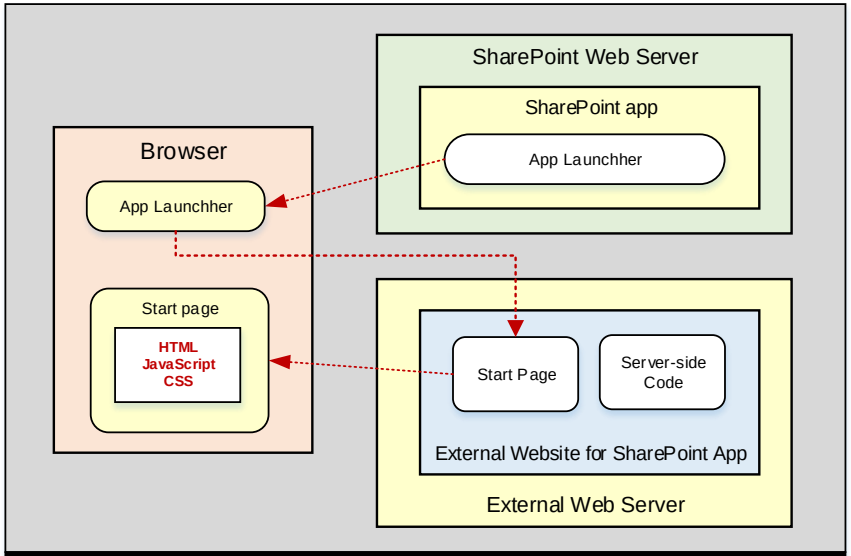


Add-in Start Page

- Every SharePoint add-in requires a start page
  - Start page provides entry point into add-in
  - SharePoint adds app launcher to Site Contents in host web
  - SharePoint-Hosted add-in start page hosted by SharePoint
  - Provider-Hosted add-in start page hosted in remote web



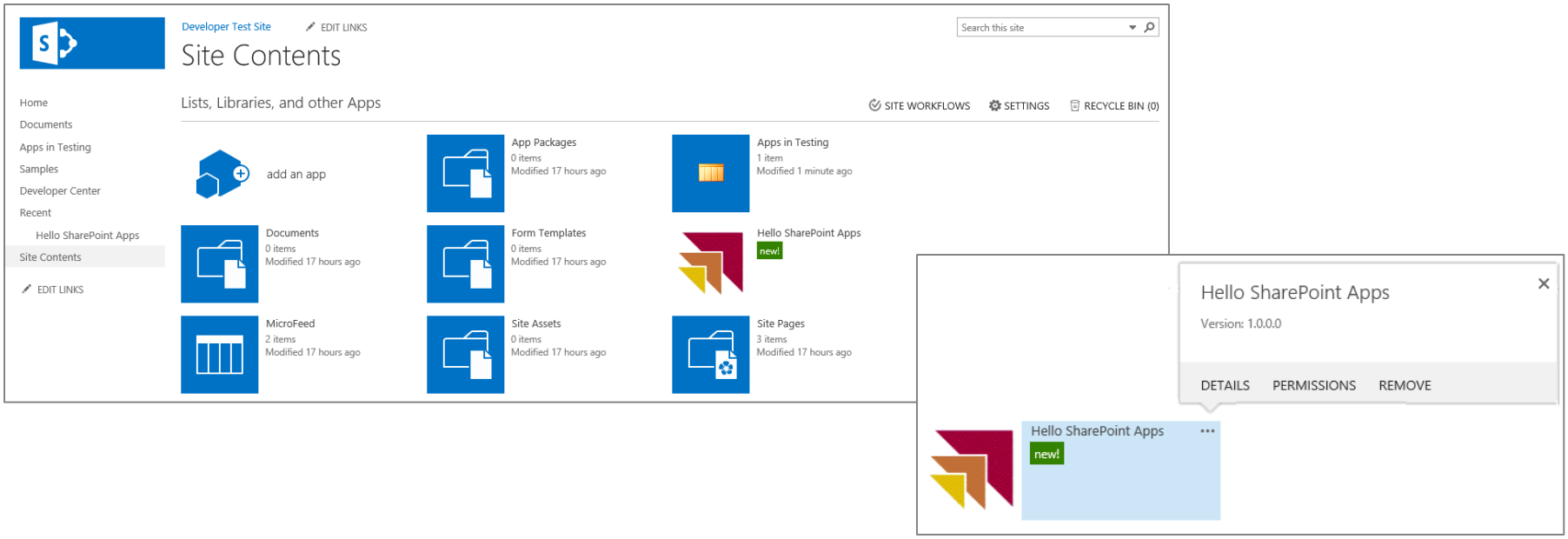
SharePoint-hosted App



Provider-hosted App

User Experience with SharePoint Add-ins

- Users launch add-ins from tile on Site Contents
  - SharePoint add-ins grouped together with lists and libraries
  - Clicking on add-in tile redirects user to app's start page
  - Add-in tile provides fly-out menus for details and uninstall

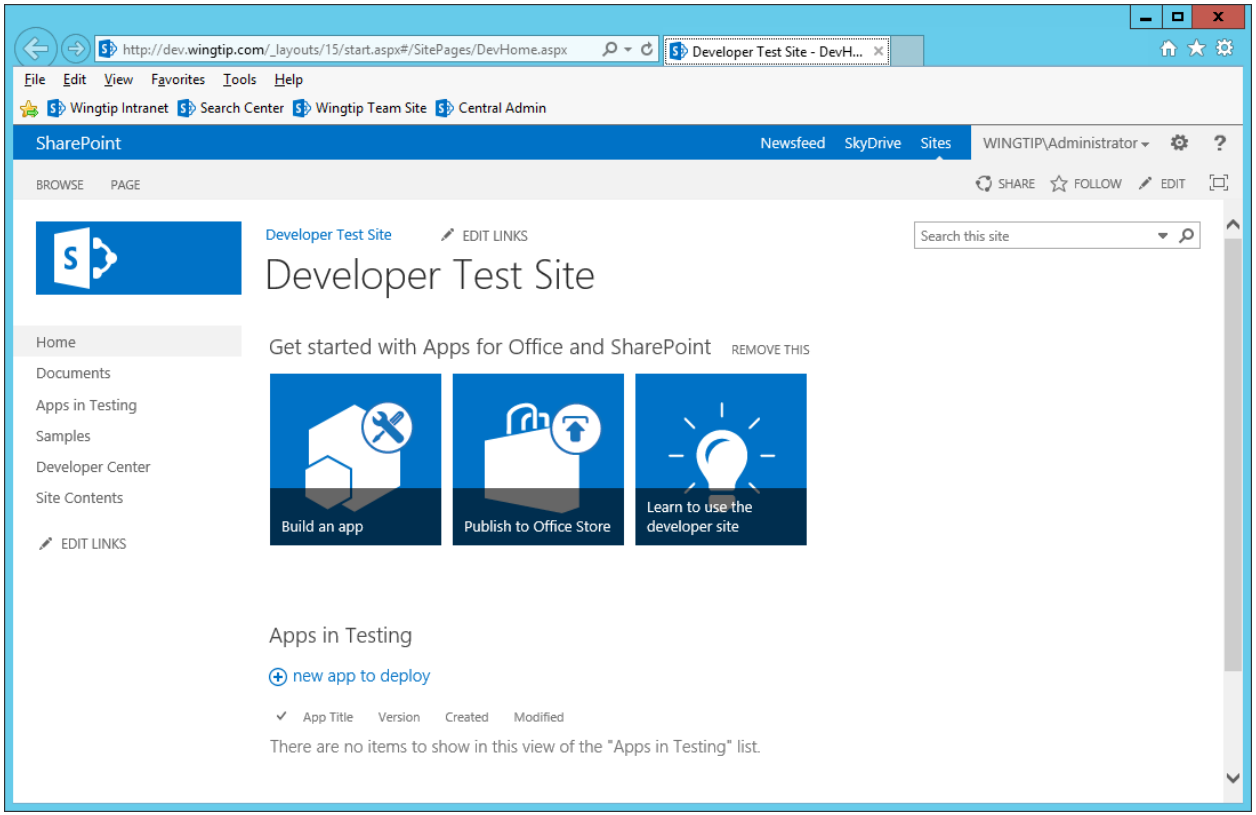


DEMO

Working with Add-ins from the Business User Perspective

Developer Sites

- Allows for remote add-in installation by Visual Studio
  - Required for testing add-ins in SharePoint Online environment



DEMO

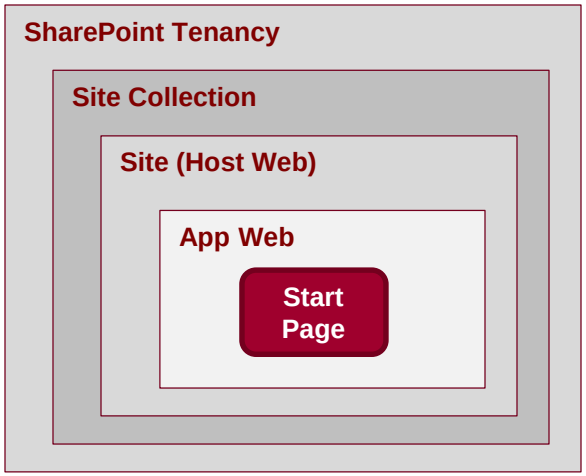
Creating a Developer Site

Agenda

- ✓ SharePoint Add-in Model
- ➢ SharePoint-hosted Add-in Architecture
- User Interface Design Techniques
- Developing Add-in Parts
- Adding User Custom Actions

SharePoint-hosted Add-in Architecture

- SharePoint-hosted app fundamentals
  - SharePoint host creates app web during installation
  - App start page and resources are added into app web
  - All app logic must be written in client-side JavaScript
  - App authentication happens behind the scenes



App Web (aka Add-in Web)

- App web is created during app installation
  - App web created as child to site where app is installed
- SharePoint-hosted apps must create app web
  - App must add start page and related resources
  - App can add other SharePoint elements (e.g. lists)
- Provider-hosted apps *can* create app web
  - Provider-hosted apps will not create app web by default
  - Provider-hosted app can create app web if needed

App Web Hosting Domain

- App web pages served out of isolated domain
  - Isolates JavaScript code on app web pages
  - Allows SharePoint to authenticate callbacks from app

```
https://mytenant-ee060af276f95a.sharepoint.com/MyFirstApp
```

- URL to app web made up of 4 parts
  - **Tenancy name:** mytenant
  - **APPUID:** ee060af276f95a
  - **App web hosting domain:** sharepoint.com
  - **App name:** MyFirstApp

Start Page URL

- Dynamic tokens used in start page URL
  - SharePoint-Hosted apps use **~appWebUrl** token

    `~appWebUrl/Pages/Default.aspx`

  - All apps should use **{StandardTokens}** token

    `~appWebUrl/Pages/Default.aspx?{StandardTokens}`

{StandardTokens}

- **Start Page URL contains {StandardTokens}**
  - **Dynamic placeholder for querystring parameters**

Parameter	Purpose
SPHostUrl	URL back to host web
SPAppWebUrl	URL to app web
SPLanguage	Language in use (e.g. en-US)
SPClientTag	Client cache control number for the current website.
SPProductNumber	Version of SharePoint (e.g. 15.0.4433.1011)

Agenda

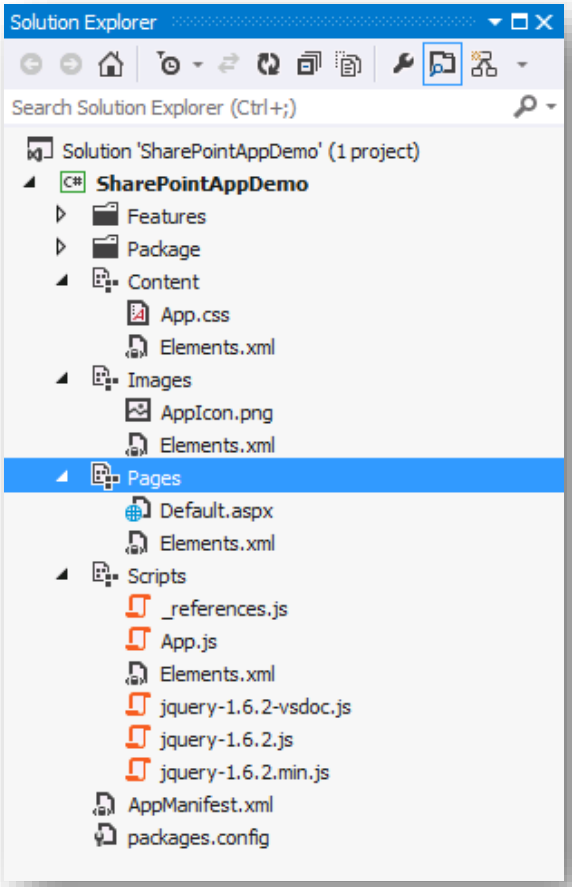
- ✓ SharePoint Add-in Model
- ✓ SharePoint-hosted Add-in Architecture
- ➢ User Interface Design Techniques
- Developing Add-in Parts
- Adding User Custom Actions

SharePoint Add-in User Interface Design

- Start page (required)
  - Represents user entry point into add-in
  - Can be implemented with .aspx file or .htm file
- Add-in Parts
  - External page (e.g. from app web) surfaced in host web
  - Displayed on host web pages using iFrame
- User Custom Actions
  - URL-based command surfaced in host web
  - Used to create ECB commands and ribbon controls

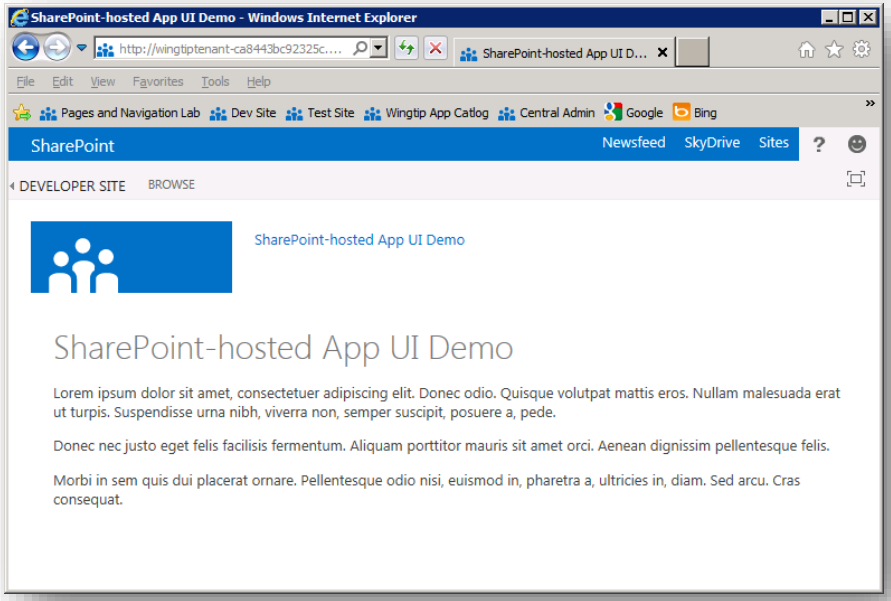
Modules in a SharePoint-Hosted Add-in

- Visual Studio adds Modules to each new project
  1. Content
  2. Images
  3. Pages
  4. Scripts



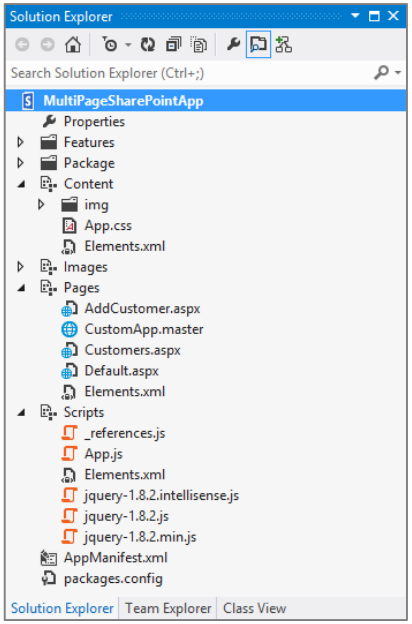
App.master

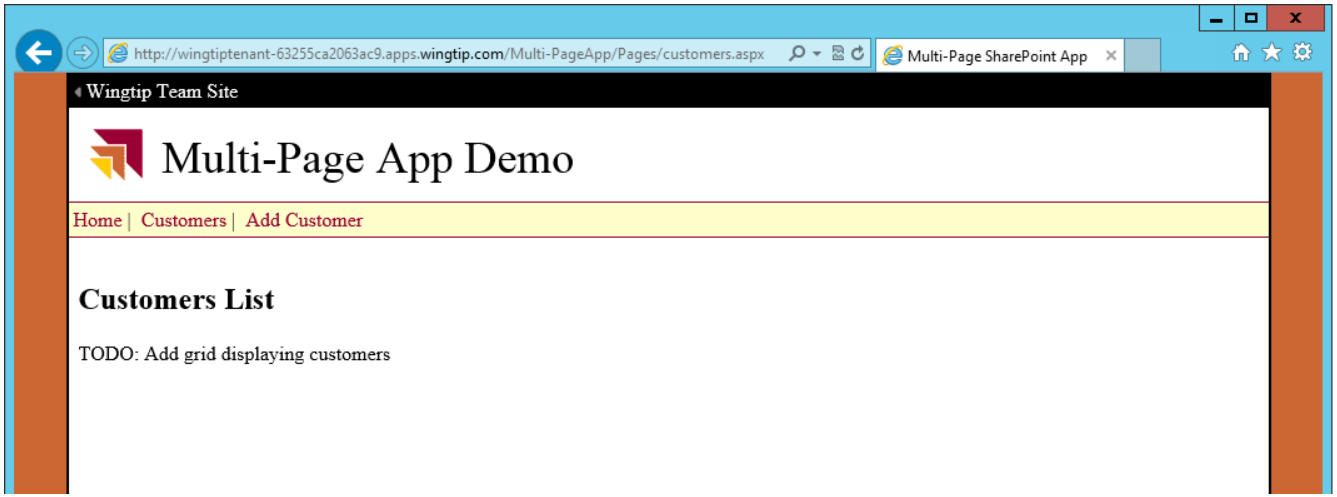
- App web uses `app.master` by default
  - Gives app SharePoint look and feel
  - Provides app with required link back to host web
  - Does not have Site Actions menu or top link bar
  - Does <u>not</u> support adding Office 365 app launcher
  - Should not be used for add-ins targeting SharePoint Online



Multi-page Add-in with Custom Master Pages

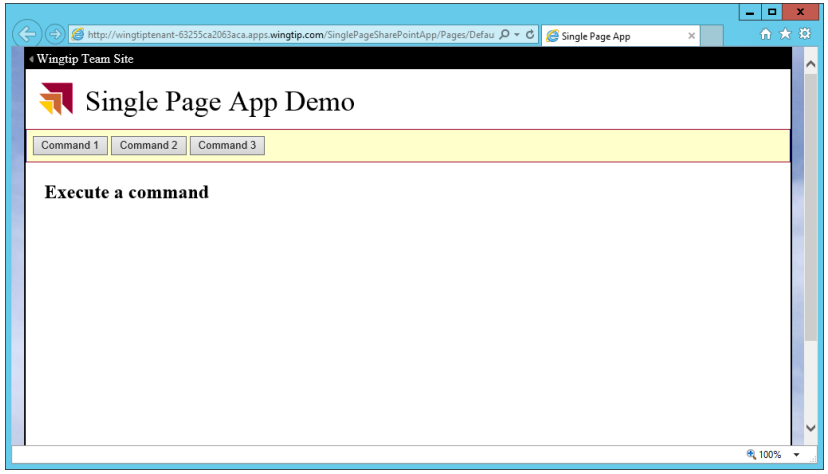
- Multiple pages can use same master page
  - Link to host web can be added to master page
  - Navigation can be added to master page
  - **Issue**: query string parameters only sent to start page

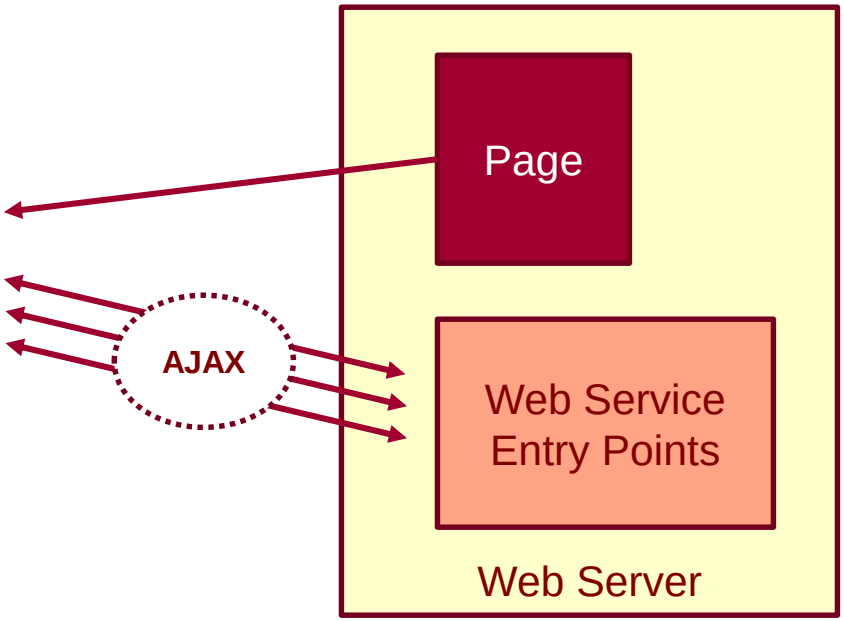




Single Page App (SPA) Model

- Web applications often designed as SPAs
  - Design leads to better and more fluid user experience
  - Request data posted to start page is always there
  - JavaScript variables do not unload/reload
  - App makes AJAX calls and uses client-side JavaScript





DEMO

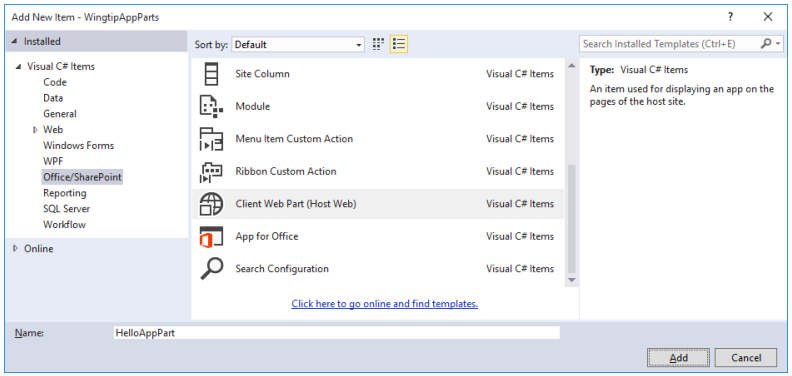
Creating a Single Page App

Agenda

- ✓ SharePoint Add-in Model
- ✓ SharePoint-hosted Add-in Architecture
- ✓ User Interface Design Techniques
- ➢ Developing Add-in Parts
- • Adding User Custom Actions

Adding Add-in Parts to a Project

- Add new item based on Client Web Part project item



- App part requires ClientWebPart definition in element.xml

```xml
<?xml version="1.0" encoding="utf-8"?>

<Elements xmlns="http://schemas.microsoft.com/sharepoint/">

  <ClientWebPart
    Name="HelloAppPart"
    Title="Hello App Part"
    Description="A simple little app part"
    DefaultWidth="600"
    DefaultHeight="200">

    <Content Type="html" Src="~appWebUrl/Pages/HelloAppPart.htm?{StandardTokens}" />

    <Properties></Properties>

  </ClientWebPart>

</Elements>
```

Add-in Parts with Custom Properties

- Add-in part can define custom properties
  - Property defined using Property element
  - Property value sent to add-in part using query string

```
<ClientWebPart Name="BetterAppPart"
               Title="Better App Part"
               Description="A really nice app part"
               DefaultWidth="600"
               DefaultHeight="200">

  <Content Type="html" Src="~appWebUrl/Pages/BetterAppPart.aspx?BackgroundColor=_BackgroundColor_&amp;He

  <Properties>

    <Property
        Name="BackgroundColor"
        WebDisplayName="Add Background Color"
        Type="boolean"
        DefaultValue="false"
        WebCategory="Custom Wingtip Properties"
        RequiresDesignerPermission="true" >
    </Property>

    <Property
        Name="HeaderColor"
        WebDisplayName="Header Color"
        Type="enum"
        DefaultValue="Black"
        WebCategory="Custom Wingtip Properties"
        RequiresDesignerPermission="true" >
      <EnumItems>
        <EnumItem WebDisplayName="Black" Value="Black"/>
        <EnumItem WebDisplayName="Blue" Value="Blue"/>
        <EnumItem WebDisplayName="Green" Value="Green"/>
      </EnumItems>
    </Property>

  </Properties>

</ClientWebPart>
```

Resizing Add-in Parts

- Add-in part displayed in host web using inside iFrame
  - IFrame given initial width and height
  - Dynamic resizing often required to avoid scrollbars
  - Resizing add-in part requires postMessage call to host

```
function resizeAppPart() {

  var pageWidth = $("#appPartContainer").width() + 20;
  var pageHeight = $("#appPartContainer").height() + 20;

  var SPHostUrl = getQueryStringParameter("SPHostUrl");
  var senderId = getQueryStringParameter("SenderId");
  var message = "<message senderId=" + senderId + ">" +
                  "resize(" + pageWidth + ", " + pageHeight + ")" +
                "</message>";

  parent.postMessage(message, SPHostUrl);


}
```

DEMO

Developing Add-in Parts

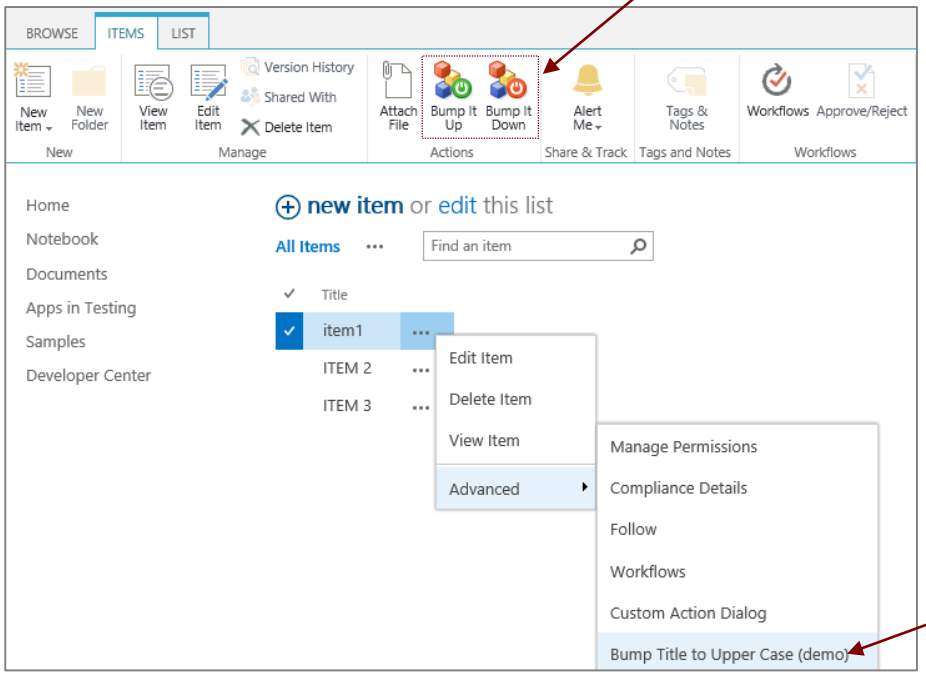
Agenda

- ✓ SharePoint Add-in Model
- ✓ SharePoint-hosted Add-in Architecture
- ✓ User Interface Design Techniques
- ✓ Developing Add-in Parts
- ➢ Adding User Custom Actions

Creating User Custom Actions

- User custom actions used to add commands to host web
  - Custom action can create ECB menu items and ribbon buttons

Custom ribbon buttons



Custom ECB menu item buttons

Creating User Custom Actions

- User custom actions used to add menu items to host web
  - Custom action can create ECB menu items and ribbon buttons
  - Created using declarative CustomAction element
  - UrlAction links to page in app web or remote web
  - UrlAction Url attribute cannot contain any JavaScript code
  - HostWebDialog attribute displays page in model dialog in host web

```
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">

  <CustomAction
    Id="c06090cd-229a-4271-968c-1c67a976d267.TitleBumper"
    RegistrationType="List"
    RegistrationId="100"
    Location="EditControlBlock"
    Sequence="10001"
    Title="Bump Title to Upper Case (demo)"
    HostWebDialog="TRUE"
    HostWebDialogWidth="500"
    HostWebDialogHeight="280" >

    <UrlAction
      Url="~appWebUrl/Pages/TitleBumper.aspx?{StandardTokens}&amp;SPListItemId={ItemId}&amp;SPListId={ListId}" />

  </CustomAction>

</Elements>
```

URL Tokens for User Custom Actions

- Certain tokens must be used with certain actions
  - Token use changes between ECB actions and Ribbon actions

Token	Purpose
{AppWebUrl}	URL of the app web in an app for SharePoint
{HostLogoUrl}	Logo for the host web of an app for SharePoint
{HostTitle}	Title of the host web of an app for SharePoint
{HostUrl}	URL of the host web of an app for SharePoint
{ItemId}	Integer-based ID of item in a list or library **(ECB menu actions only)**
{SelectedItemId}	Array of item IDs in a list or library **(Ribbon menu actions only)**
{ItemUrl}	URL of target item being acted upon **(ECB menu actions only)**
{SelectedItemUrl}	URL array of target items being acted upon (**Ribbon menu actions only)**
{Language}	current language/culture of the host web of an app for SharePoint
{ListId}	ID of the current list (a GUID).
{RecurrenceId}	Recurrence index of a recurring event
{Site}	URL of the current website
{SiteCollection}	URL of the parent site of the current website
{SiteUrl}	URL of the current website

DEMO

Adding User Custom Actions

Summary

- ✓ SharePoint Add-in Model
- ✓ SharePoint-hosted Add-in Architecture
- ✓ User Interface Design Techniques
- ✓ Developing Add-in Parts
- ✓ Adding User Custom Actions

Agenda

- ✓ Understanding REST and ODATA
- ➢ The SharePoint REST API
- Programming the SharePoint REST API
- Paging SharePoint List Items
- Modifying SharePoint List Items

Agenda

- ✓ Understanding REST and ODATA
- ✓ Creating REST URIs for SharePoint Objects
- ➢ Programming the SharePoint REST API
- Paging SharePoint List Items
- Modifying SharePoint List Items

Separating UI Code form Data Access Code

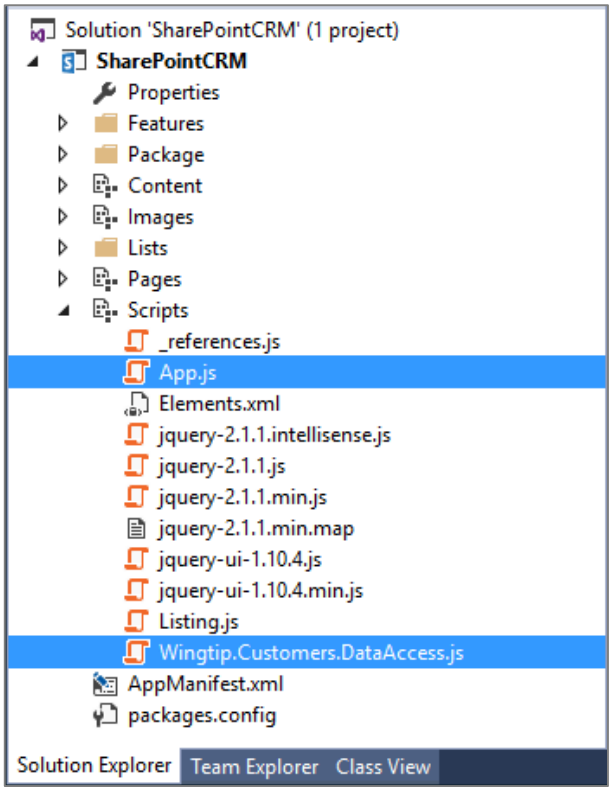
- You should not intermingle UI code and data access code
  - Do this leads to unmaintainable spaghetti code
  - Data access code should be in separate JavaScript library file
  - Best practice is to use the JavaScript revealing module pattern

Wingtip.Customers.DataAccess.js

```
var Wingtip = window.Wingtip || {};

Wingtip.Customers = Wingtip.Customers || {};

Wingtip.Customers.DataAccess = function () {

  var getCustomers = function ()...;

  var getCustomer = function (Id)...

  var addCustomer = function (FirstName, LastName, Company, WorkPhone, HomePhone, Email)...;

  var updateCustomer = function (Id, FirstName, LastName, Company, WorkPhone, HomePhone, Email, ETag)...

  var deleteCustomer = function (Id)...;

  // return object with public interface for revealing module pattern
  return {
    getCustomers: getCustomers,
    addCustomer: addCustomer,
    getCustomer: getCustomer,
    updateCustomer: updateCustomer,
    deleteCustomer: deleteCustomer
  };

}();
```

Creating A Reusable Data Access Library

- Data access code is kept separate from view (i.e. UI code)
  - Data access code returns promise to view code
  - View code encapsulated from details of data access
  - View code responsible for updating user interface when required



View code

Data access code

Service Root URI for the App Web

- Creating the App Web's Service Root URI
  - Use URL relative to **Pages** folder

```
var restURI = "../_api/web/?$select=Id,Title,Url"
```

  - Use URL created from **SPAppWebUrl** query string parameter

```
var restURI = getQueryStringParameter("SPAppWebUrl") +
              "/_api/web/?$select=Id,Title,Url"
```

  - Use URL created from **_spPageContextInfo.webAbsoluteUrl**

```
var restURI = _spPageContextInfo.webAbsoluteUrl +
              "/_api/web/?$select=Id,Title,Url"
```

Querying a List in the App Web

- Create reusable data access function

```
var getCustomers = function () {

  // create URI for request
  var requestUri = "../_api/web/lists/getByTitle('Customers')/items" +
                   "?$select=ID,FirstName,Title,Company,WorkPhone,HomePhone,Email" +
                   "&$orderby=Title,FirstName";

  // send call across network
  return $.ajax({
    url: requestUri,
    type: "GET",
    headers: { "accept": "application/json" }
  });

};
```

- Call the reusable data access function from the add-in's UI code

```
Wingtip.Customers.DataAccess.getCustomers().then(function (data) {
  // get OData result from data.value
  var customers = data.value;
  // create HTML table using OData result
  var table = $("<table>", { ID: "customersTable" });
  // add table header row
  table.append($("<thead>")
          .append($("<td>").text("First Name"))
          .append($("<td>").text("Last Name")));
      // add table data rows
  for (var customer = 1; customer < customers.length; customer++) {
    table.append($("<tr>")
          .append($("<td>").text(customers[customer].FirstName))
          .append($("<td>").text(customers[customer].Title)));
  }
  // append HTML table to div on page
  $("#content_box").append(table);
});
```

Service Root URI for the Host Web

- This code will fail because it attempts a cross-domain call

```
var restURI = getQueryStringParameter("SPHostUrl") +
              "/_api/web/?$select=Id,Title,Url";
```

- This code works in some but not all scenarios

```
var restURI = "/_api/web/?$select=Id,Title,Url";
```

```
var restURI = "../../_api/web/?$select=Id,Title,Url"
```

- This code works in all scenarios and this is what you should use

```
var restURI = "../_api/SP.AppContextSite(@target)/web/" +
              "?$select=Id,Title,Url" +
              "&@target='" + getQueryStringParameter("SPHostUrl") + "'";
```

Querying for Lists within the Host Web

- Use SP.AppContextSite in URI to access host web from app web
  - Call gets routed through app web so there is no cross-domain call
  - SP.AppContextSite allows you to program against site in host web domain

```
var getLists = function () {

  var requestUri = "../_api/SP.AppContextSite(@target)/web/lists/" +
                   "?$filter=(Hidden eq false) and (BaseType eq 0)" +
                   "&$select=Id,Title, ItemCount, EntityTypeName, ListItemEntityTypeFullName" +
                   "&@target='" + hostWebUrl + "'";

  return $.ajax({
    url: requestUri,
    type: "GET",
    headers: { "accept": "application/json;odata=verbose" }
  });

};
```

Using the $expand Query Option

- **$expand** used to create more efficient code
  - Deferred content held back by default
  - **$expand** used to retrieve results with deferred content
  - Effectively reduces round trips

```
var getSiteGroups = function () {

  var requestUri = "../_api/SP.AppContextSite(@target)/web/siteGroups/" +
                   "?$expand=Users" +
                   "&@target='" + hostWebUrl + "'";;

  return $.ajax({
    url: requestUri,
    type: "GET",
    headers: { "accept": "application/json;odata=verbose" },
  });

};
```

DEMO

- Ted Pattison

HostWebExplorer App

Agenda

- ✓ Understanding REST and ODATA
- ✓ Creating REST URIs for SharePoint Objects
- ✓ Programming the SharePoint REST API
- ➢ Paging SharePoint List Items
- • Modifying SharePoint List Items

Paging with SharePoint Lists

- SharePoint does not support **$skip** for list items
  - You cannot create typical OData paging scheme with a SharePoint list
- What do you do instead?
  - Create a custom paging scheme using **$filter**
  - Create a paging scheme using **$skiptoken**

DEMO

Paging with SharePoint List Items

Agenda

- ✓ Understanding REST and ODATA
- ✓ The SharePoint REST API
- ✓ Creating REST URIs for SharePoint Objects
- ✓ Programming the SharePoint REST API
- ✓ Paging SharePoint List Items
- ➢ Modifying SharePoint List Items

Updating SharePoint Objects

- All write operations must pass valid request digest value
- You must include type metadata for inserts & updates
- Sometimes you must pass ETags for updates & deletes

Understanding the Request Digest

- All SharePoint write operations require Request Digest
  - Provides security mechanism to protect again replay attacks
  - Request digest known to SharePoint old timers as “Form Digest”
  - SharePoint adds request digest element with ID **__REQUESTDIGEST**
  - Request digest value passed using **X-RequestDigest** header

```
var requestHeaders = {
  "accept": "application/json;odata=verbose",
  "X-RequestDigest": $("#__REQUESTDIGEST").val()
}
```

Caching the Request Digest

- Request digest queried using **/_api/contextinfo**

```
Wingtip.Customers.DataAccess = function () {

  var requestDigest;

  var initialize = function () {

    var deferred = $.ajax({
      url: "../_api/contextinfo",
      type: "POST",
      headers: { "accept": "application/json;odata=verbose" }
    })

    deferred.then(function (data) {
      requestDigest = data.d.GetContextWebInformation.FormDigestValue
    });

  }
```

Working with List Item Type Metadata

- Each SharePoint list has a unique type for its list items

```
{}
  __metadata
    etag="1"
    id=02d4852e-43b4-4023-9706-07358b789674
    type=SP.Data.CustomersListItem
```

- Verbose syntax requires **type** value be passed with inserts & updates
  - Type value can be omitted with non-verbose syntax (content-type=application/json)

```
var customerData = {
  __metadata: { "type": "SP.Data.CustomersListItem" },
  Title: LastName,
  FirstName: FirstName,
  Company: Company,
  WorkPhone: WorkPhone,
  HomePhone: HomePhone,
  Email: Email
};
```

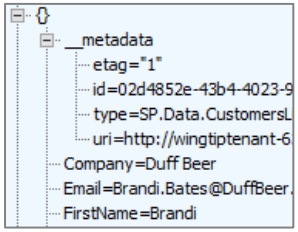
- **type** discoverable using **ListItemEntityTypeFullName** property

Adding a SharePoint List Item

```
var addCustomer = function (FirstName, LastName, Company, WorkPhone, HomePhone, Email) {

  var requestUri = "../_api/web/lists/getByTitle('Customers')/items";

  var requestHeaders = {
    "accept": "application/json;odata=verbose",
    "X-RequestDigest": $("#__REQUESTDIGEST").val()
  }

  var customerData = {
    __metadata: { "type": "SP.Data.CustomersListItem" },
    Title: LastName,
    FirstName: FirstName,
    Company: Company,
    WorkPhone: WorkPhone,
    HomePhone: HomePhone,
    Email: Email
  };

  var requestBody = JSON.stringify(customerData);

  return $.ajax({
    url: requestUri,
    type: "POST",
    contentType: "application/json;odata=verbose",
    headers: requestHeaders,
    data: requestBody,
  });

};
```

ETags and Optimistic Concurrency

- OData v2 requires items to carry ETags
  - ETag is integer value in that it identities version of item
  - ETag is automatically incremented with each update



- ETag use to support for optimistic concurrency control
  - ETag works to eliminate the “lost update” scenario
  - ETag must be tracked in order to post updates in most scenarios

```
// store item metadata values into hidden controls
$("#customer_id").val(data.d.ID);
$("#etag").val(data.d.__metadata.etag);
```

ETags and the If-Match Header

- Update and Delete operations require If-Match Header
  - Allows you to pass ETag value during an update
  - Update fails if ETag value changed due to update by other user

```
var requestHeaders = {
  "accept": "application/json;odata=verbose",
  "X-HTTP-Method": "MERGE",
  "X-RequestDigest": $("#__REQUESTDIGEST").val(),
  "If-Match": ETag
}
```

- You can pass wildcard (*) value inside If-Match Header
  - Done to disable optimistic concurrency control
  - This is commonly done with delete operations

```
var requestHeaders = {
  "accept": "application/json;odata=verbose",
  "X-RequestDigest": $("#__REQUESTDIGEST").val(),
  "If-Match": "*"
}
```

Updating a SharePoint List Item

```
var updateCustomer = function (Id, FirstName, LastName, Company, WorkPhone, HomePhone, Email, ETag) {

  var requestUri = "../_api/web/lists/getByTitle('Customers')/items(" + Id + ")";

  var requestHeaders = {
    "accept": "application/json;odata=verbose",
    "X-HTTP-Method": "MERGE",
    "X-RequestDigest": $("#__REQUESTDIGEST").val(),
    "If-Match": ETag
  }

  var customerData = {
    __metadata: { "type": "SP.Data.CustomersListItem" },
    Title: LastName,
    FirstName: FirstName,
    Company: Company,
    WorkPhone: WorkPhone,
    HomePhone: HomePhone,
    Email: Email
  };

  var requestBody = JSON.stringify(customerData);

  return $.ajax({
    url: requestUri,
    type: "POST",
    contentType: "application/json;odata=verbose",
    headers: requestHeaders,
    data: requestBody,
  });

};
```

Deleting a SharePoint List Item

```
var deleteCustomer = function (Id) {

  var requestUri = "../_api/web/lists/getByTitle('Customers')/items(" + Id + ")";

  var requestHeaders = {
    "accept": "application/json;odata=verbose",
    "X-RequestDigest": $("#__REQUESTDIGEST").val(),
    "If-Match": "*"
  }

  return $.ajax({
    url: requestUri,
    type: "DELETE",
    headers: requestHeaders,
  });

};
```

DEMO

Exploring the SharePointCRM App

Summary

- ✓ The SharePoint REST API
- ✓ Creating REST URIs for SharePoint Objects
- ✓ Programming the SharePoint REST API
- ✓ Paging SharePoint List Items
- ✓ Modifying SharePoint List Items