

# Developing Provider-hosted Add-ins



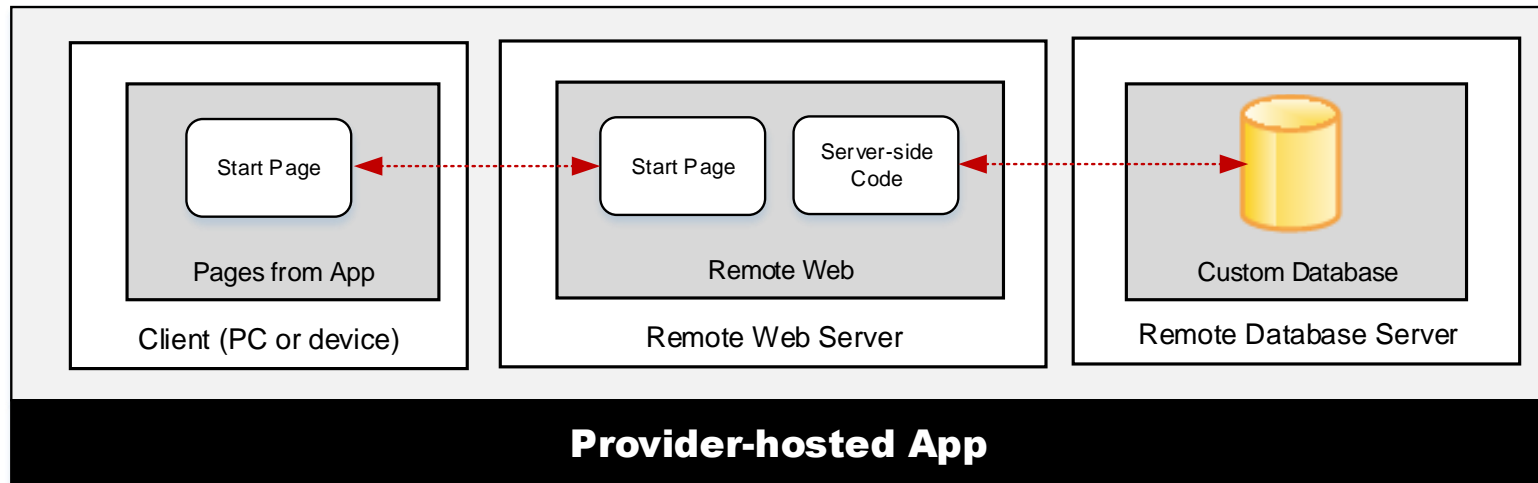
# Agenda

- Getting Started with Provider-hosted Apps
- User Interface Design for the Remote Web
- Working with ASP.NET MVC
- Creating Provider-hosted Apps using MVC5



# Provider-Hosted App

- Developer responsible for deploying remote web
  - App deployed to remote web on remote web server
  - Developer deploys remote web prior to app installation
  - Developer often required to deploy database as well



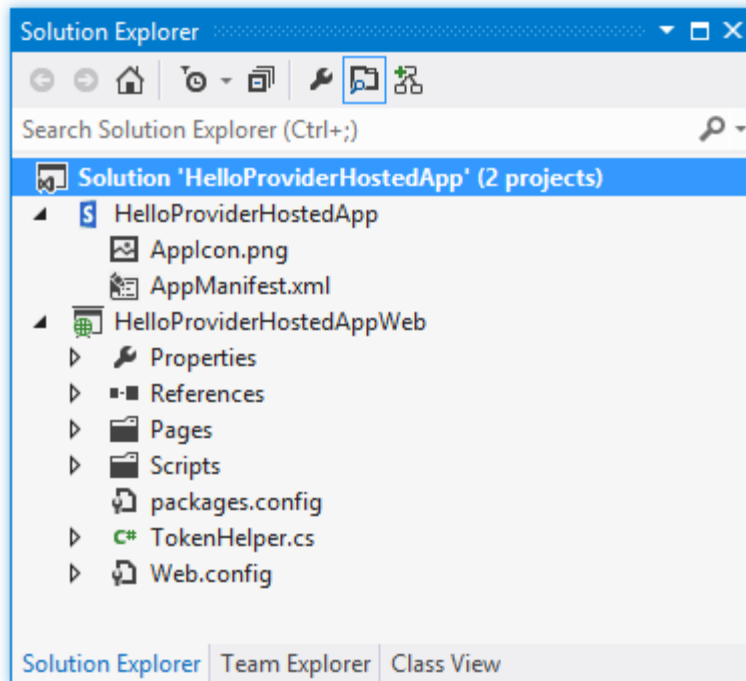
# Pros and Cons of Provider-hosted Apps

- Benefits of provider-hosted over SharePoint-hosted apps
  - You can write server-side .NET code using C# or VB.NET ([wahoo!])
  - Your server-side code can access data in a custom database
  - Your server-side code has no cross-domain scripting constraints
  - You can leverage the support for remote event receivers
  - You can make CSOM/REST API calls using App-only permissions
- Negatives when compared to SharePoint-hosted apps
  - You must deal with hosting (i.e. deploy & manage) remote web
  - Requires extra code to acquire and manage security tokens



# Provider-hosted App Projects

- Visual Studio create solution with two projects
  - SharePoint app project
  - ASP.NET Website project for remote web  
*this project is known as the “web project”*



# AppManifest.xml

- Provider-hosted app adds requirements to App Manifest
  - StartPage must point to page in remote web
  - AppPrincipal requires app authentication settings
  - External app authentication can be disabled using Internal setting

```
<App xmlns="http://schemas.microsoft.com/sharepoint/2012/app/manifest"
      Name="HelloProvider-HostedApp"
      ProductID="{8d587998-fdbf-4d97-a739-613a647bed83}"
      Version="1.0.0.0"
      SharePointMinVersion="15.0.0.0" >

  <Properties>
    <Title>Hello Provider-Hosted App</Title>
    <StartPage>~remoteAppUrl/Pages/Default.aspx?{StandardTokens}</StartPage>
  </Properties>

  <AppPrincipal>
    <!-- turn off external app authentication -->
    <Internal />
  </AppPrincipal>

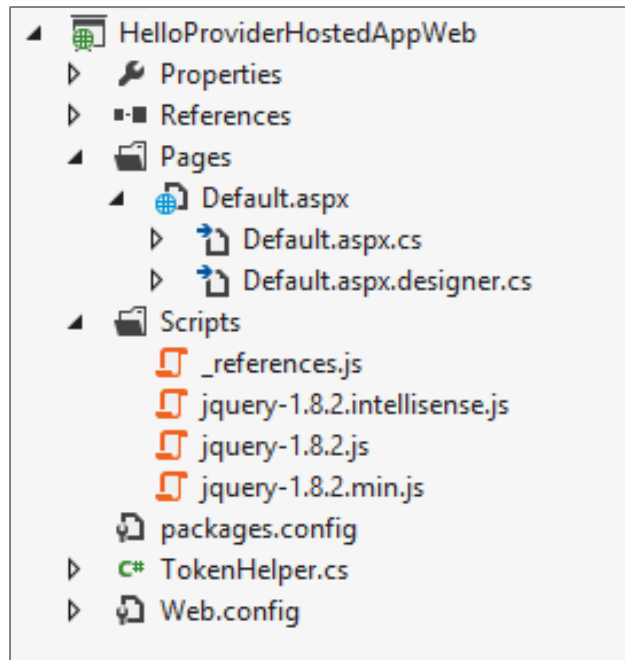
</App>
```





# Implementing the Remote Web

- Web project provides start point for remote web
  - `web.config` for site configuration of remote web
  - `Default.aspx` for start page with HTML layout
  - `Default.aspx.cs` for server-side C# code behind start page
  - `Scripts` folder with jQuery library already added
  - `TokenHelper` class for security-related programming



# A Sample Start Page

```
<%@ Page Language="C#" AutoEventWireup="true"
    CodeBehind="Default.aspx.cs" Inherits="HelloProviderHostedAppWeb.Pages.Default" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">
    <title>My Start Page</title>
</head>

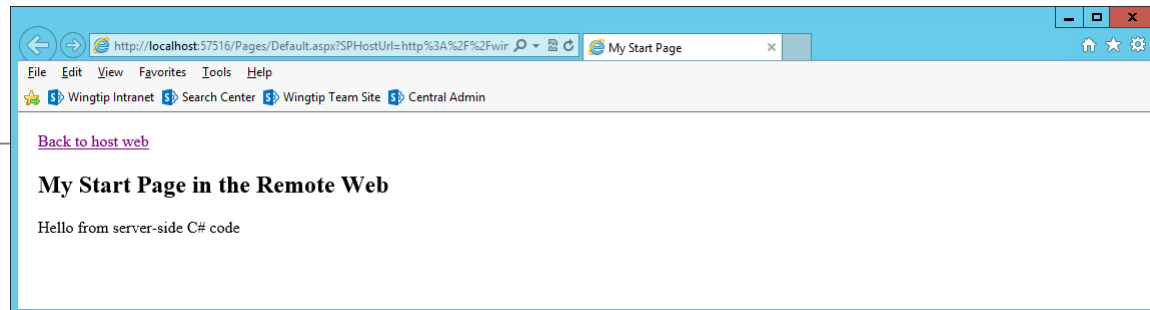
<body>
    <form id="form1" runat="server">
        <div>

            <!-- add HyperLink control to link back to host web-->
            <div><asp:HyperLink ID="linkHostWeb" runat="server">Back to host web</asp:HyperLink></div>

            <!-- add some HTML content to page -->
            <h2>My Start Page in the Remote Web</h2>

            <!-- -->
            <asp:PlaceHolder ID="PlaceHolderMain" runat="server"></asp:PlaceHolder>

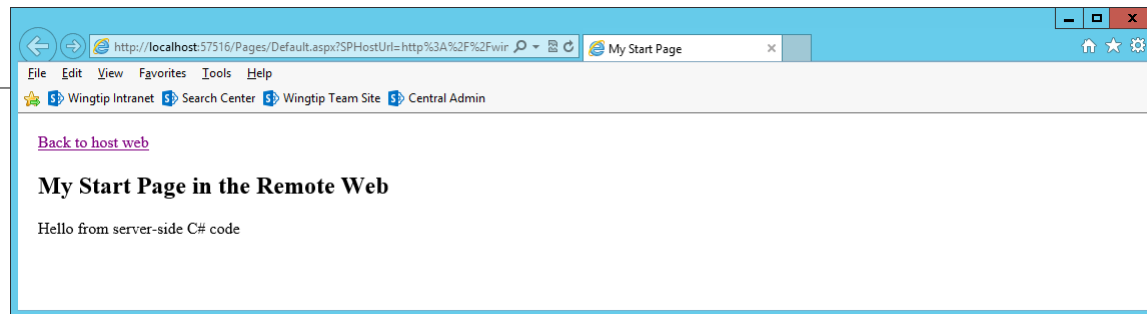
        </div>
    </form>
</body>
</html>
```





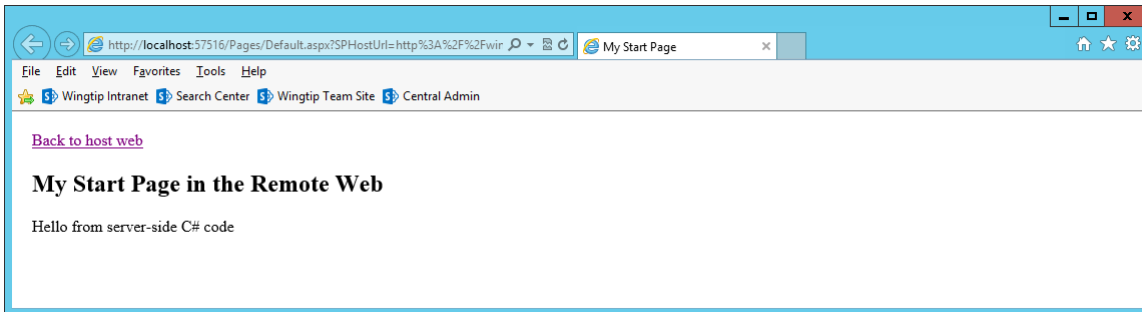
# C# Code Behind Sample Start Page

```
namespace HelloProviderHostedAppWeb.Pages {  
    public partial class Default : System.Web.UI.Page {  
  
        protected void Page_Load(object sender, EventArgs e) {  
  
            // delete all existing code added by Visual Studio - it requires authentication  
  
            // Configure ASP.NET Hyperlink control with value from SPHostUrl querystring  
            linkHostWeb.NavigateUrl = Request.QueryString["SPHostUrl"];  
  
            // add some content to the page using server-side code  
            PlaceholderMain.Controls.Add( new LiteralControl("Hello from server-side C# code"));  
  
        }  
    }  
}
```

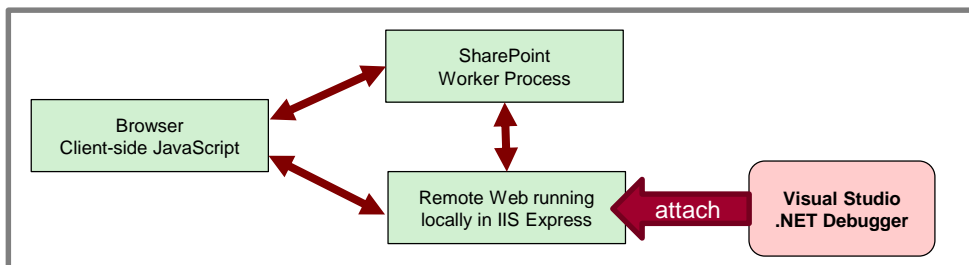


# Debugging the Remote Web in IIS Express

- Visual Studio debugging involves IIS Express
  - URL created in **localhost** domain (e.g. **https://localhost:57516**)
  - Port number for Remote Web project selected automatically behind scenes

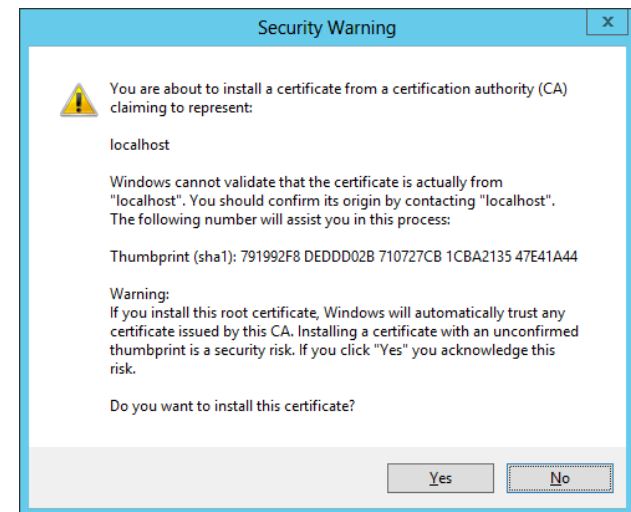
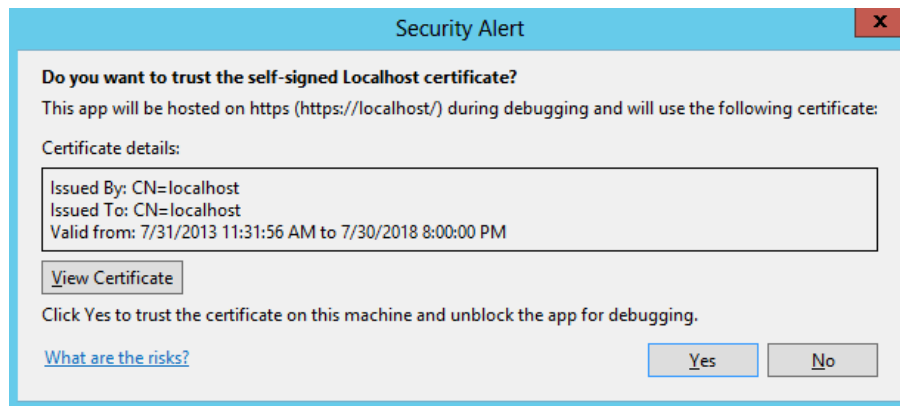


- IIS Express debugging support allows you to debug...
  - Server-side C# code behind pages in the remote web
  - Web service calls from browser to remote web
  - Web service calls from SharePoint to remote web



# Debugging Code in the Remote Web

- Remote Web can optionally use SSL
  - Pages can be served using HTTPS or HTTP
  - Use of SSL is usually preferred
  - IIS Express can configure SSL through <https://localhost>
  - Visual Studio registers self-signed certificate on first use







**DEMO**

# Creating the 'Hello World' Provider-Hosted App

# Agenda

- ✓ Getting Started with Provider-hosted Apps
- User Interface Design for the Remote Web
  - Working with ASP.NET MVC
  - Creating Provider-hosted Apps using MVC5



# Designing the Remote Web User Interface

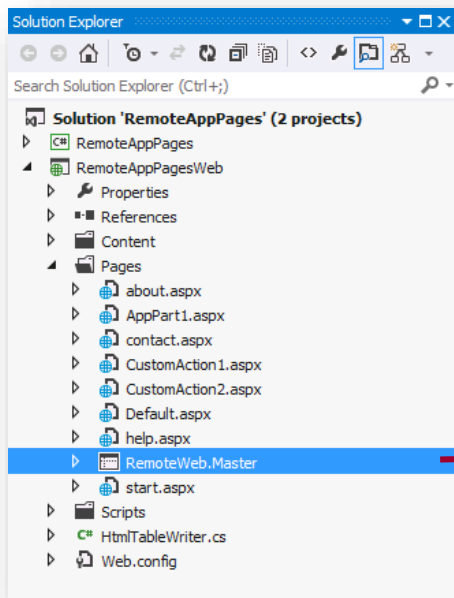
- Create user interface using single page app pattern
  - Start page serves as exclusive user entry point into remote web
  - Start page designed with HTML markup and ASP.NET controls
  - Start page can be written with server-side C# code
  - Start page can also include client-side JavaScript code
  - JavaScript code can call to custom web services in remote web
- Create user interface as a multi-page app
  - Users can navigate from start page to other pages in remote web
  - ASP.NET master page can create consistent UI across pages
  - Master page can leverage SharePoint Chrome control
  - Multi-page app loses major benefits of single-page app pattern
  - Extra work required to track start page POST data across requests





# Using Master Pages in the Remote Web

- Pages in remote web live outside SharePoint
  - You have responsibility and freedom to build entire UI
  - Often makes sense to create remote web master page



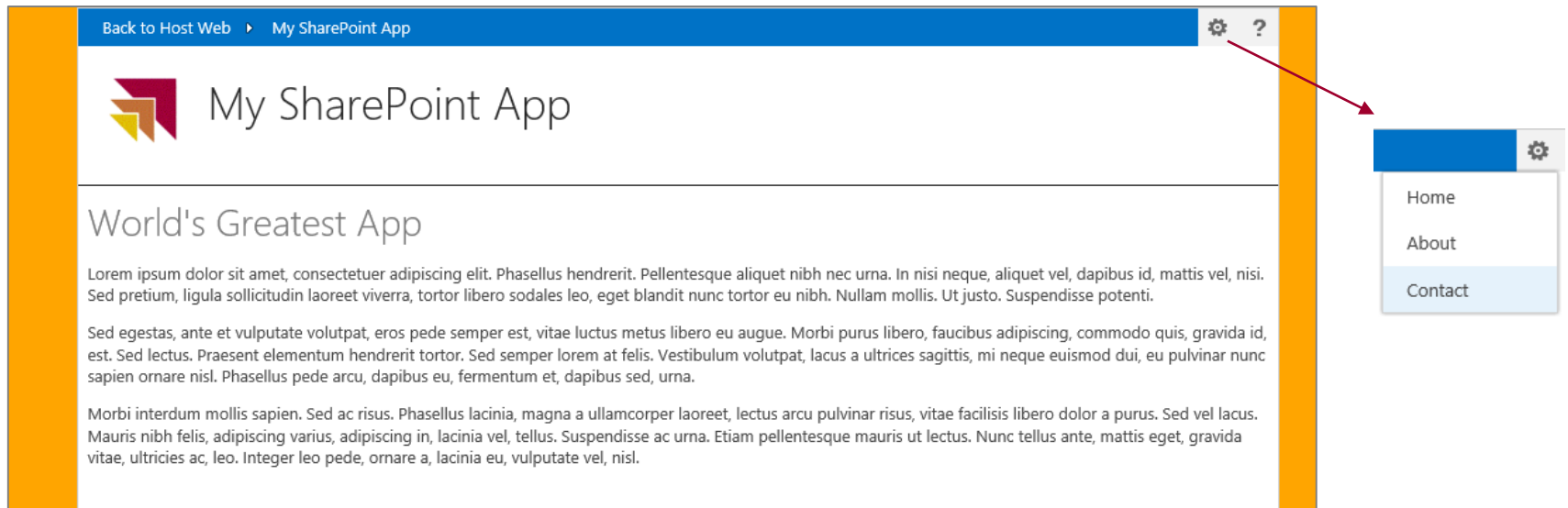
```
<%@Master language="C#" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
<head>
  <title>Remote App Pages</title>
  <link href="/Contents/app.css" type="text/css" rel="stylesheet" />
  <asp:ContentPlaceHolder ID="PlaceholderAdditionalPageHead" runat="server" />
</head>

<body>
  <form id="form1" runat="server">
    <div id="pageWidth">
      <div id="chrome_ctrl_container"></div>
      <div id="contentBody">
        <asp:ContentPlaceHolder ID="PlaceholderMain" runat="server" />
      </div>
    </div>
  </form>
</body>
</html>
```

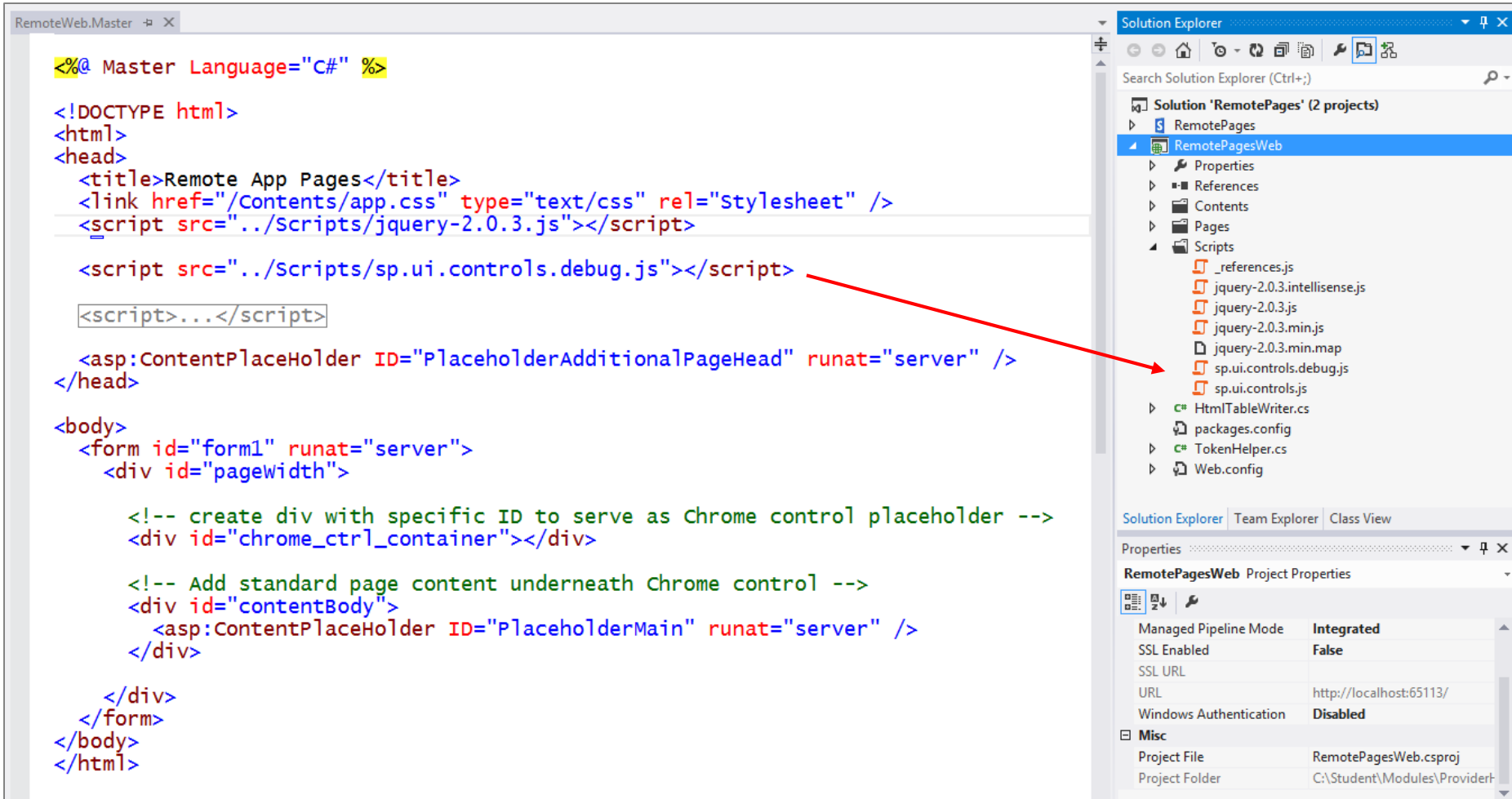


# Leveraging the Chrome Control

- Optional JavaScript component for use in Remote Web
  - Used to create top banner of page with SharePoint look and feel
  - Provides link to host web, configurable navigation and help link
  - Can pull custom styles used in host web



# Using the Chrome control



The screenshot displays the Visual Studio IDE with a C# ASP.NET Master Page file named `RemoteWeb.Master` open in the editor. The code defines the page structure, including a title, CSS link, and JavaScript references. A red arrow points from the `sp.ui.controls.debug.js` script reference in the code to its corresponding file in the Solution Explorer. The Solution Explorer shows the project structure for 'RemotePages' (2 projects), with 'RemotePagesWeb' selected. The 'Scripts' folder under 'RemotePagesWeb' contains the referenced files. The Properties window at the bottom right shows the 'RemotePagesWeb' Project Properties, including the URL `http://localhost:65113/`.

```
<%@ Master Language="C#" %>

<!DOCTYPE html>
<html>
<head>
    <title>Remote App Pages</title>
    <link href="/Contents/app.css" type="text/css" rel="stylesheet" />
    <script src="../Scripts/jquery-2.0.3.js"></script>

    <script src="../Scripts/sp.ui.controls.debug.js"></script>

    <script>...</script>

    <asp:ContentPlaceholder ID="PlaceholderAdditionalPageHead" runat="server" />
</head>
<body>
    <form id="form1" runat="server">
        <div id="pagewidth">

            <!-- create div with specific ID to serve as Chrome control placeholder -->
            <div id="chrome_ctrl_container"></div>

            <!-- Add standard page content underneath Chrome control -->
            <div id="contentBody">
                <asp:ContentPlaceholder ID="PlaceholderMain" runat="server" />
            </div>

        </div>
    </form>
</body>
</html>
```

**Solution Explorer**

- Solution 'RemotePages' (2 projects)
  - RemotePages
    - RemotePagesWeb
      - Properties
      - References
      - Contents
      - Pages
      - Scripts
        - \_references.js
        - jquery-2.0.3.intellisense.js
        - jquery-2.0.3.js
        - jquery-2.0.3.min.js
        - jquery-2.0.3.min.map
        - sp.ui.controls.debug.js
        - sp.ui.controls.js
      - HtmlTableWriter.cs
      - packages.config
      - TokenHelper.cs
      - Web.config

**Properties**

**RemotePagesWeb Project Properties**

Managed Pipeline Mode	Integrated
SSL Enabled	False
SSL URL	
URL	http://localhost:65113/
Windows Authentication	Disabled
<b>Misc</b>	
Project File	RemotePagesWeb.csproj
Project Folder	C:\Student\Modules\Provider-



# Initializing the Chrome Control

```
function getQueryStringParameter(paramToRetrieve) {
    var params = document.URL.split("?")[1].split("&");
    var strParams = "";
    for (var i = 0; i < params.length; i = i + 1) {
        var singleParam = params[i].split("=");
        if (singleParam[0] == paramToRetrieve)
            return singleParam[1];
    }
}

$(function() {
    // determine URL back to host web
    var hostWebUrl = decodeURIComponent(getQueryStringParameter("SPHostUrl"));

    // create setting object for Chrome control
    var options = {
        siteUrl: hostWebUrl,
        siteTitle: "Host Web",
        appHelpPageUrl: "help.aspx?SPHostUrl=" + hostWebUrl,
        appIconUrl: "/Contents/AppIcon.png",
        appTitle: "wingtip App",
        settingsLinks: [
            { linkUrl: "start.aspx?SPHostUrl=" + hostWebUrl, displayName: "Home" },
            { linkUrl: "about.aspx?SPHostUrl=" + hostWebUrl, displayName: "About" },
            { linkUrl: "contact.aspx?SPHostUrl=" + hostWebUrl, displayName: "Contact" }
        ]
    };

    // create Chrome control instance
    var nav = new SP.UI.Controls.Navigation("chrome_ctrl_container", options);
    nav.setVisible(true);

    // fix RTM bug with help link
    var helpIconUrl = hostWebUrl + "/_layouts/15/1033/images/spintl.png";
    var helpLink = $("#chromeControl_topheader_helplink");
    helpLink.css({ "background-image": "url('" + helpIconUrl + "')" });
});
```







**DEMO**

# **Creating Pages in the Remote Web using the Chrome Control**

# Agenda

- ✓ Getting Started with Provider-hosted Apps
- ✓ User Interface Design for the Remote Web
- Working with ASP.NET MVC
  - Creating Provider-hosted Apps using MVC5

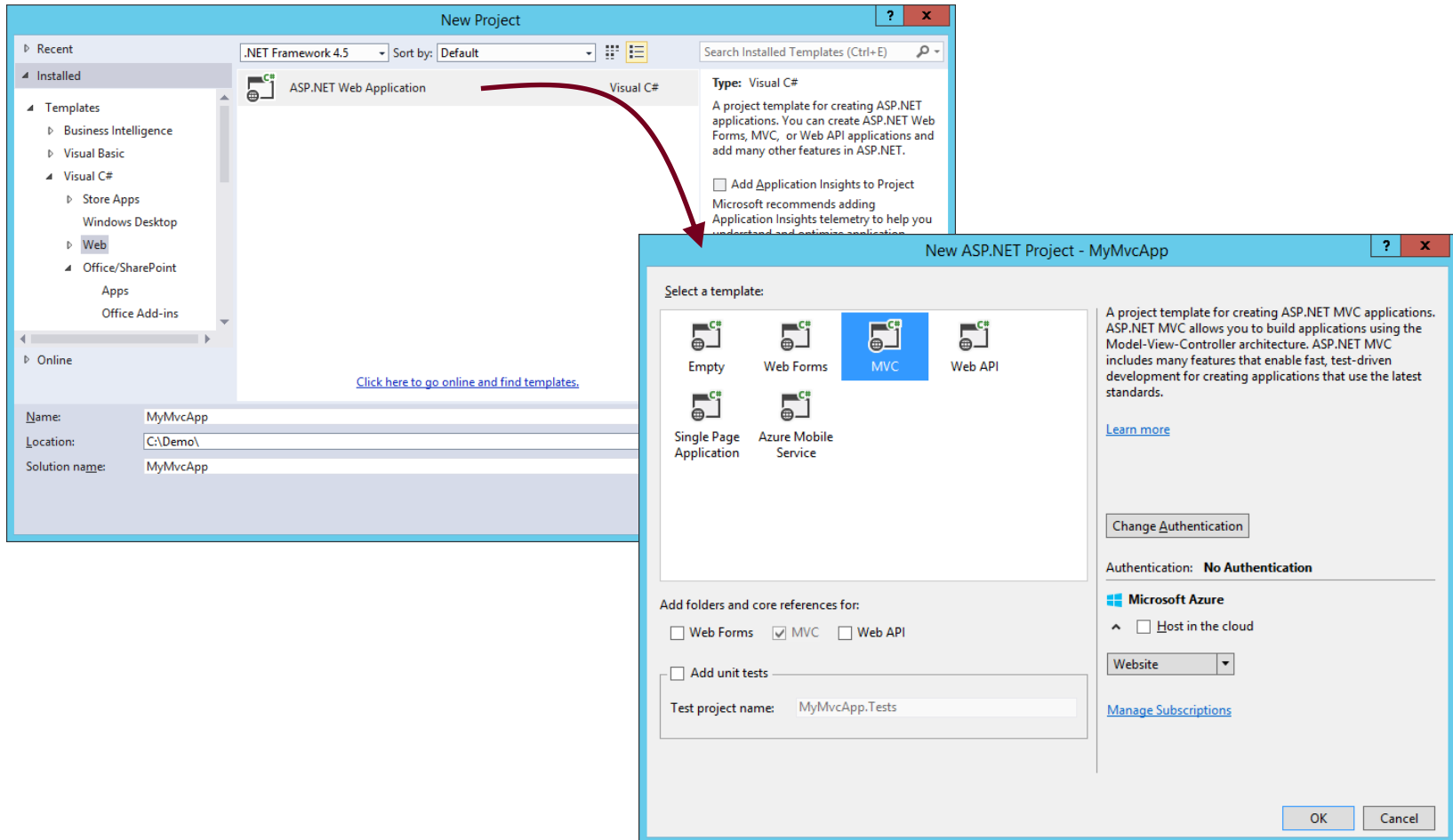


# Web Forms Versus MVC

- ASP.NET provides two different platforms
  - ASP.NET Web Forms (e.g. ASPX files)
  - ASP.NET MVC
- MVC provides better platform for the web
  - More flexible routing
  - Lighter-weight
  - Richer templating
  - Better C# integration
  - Unit testing

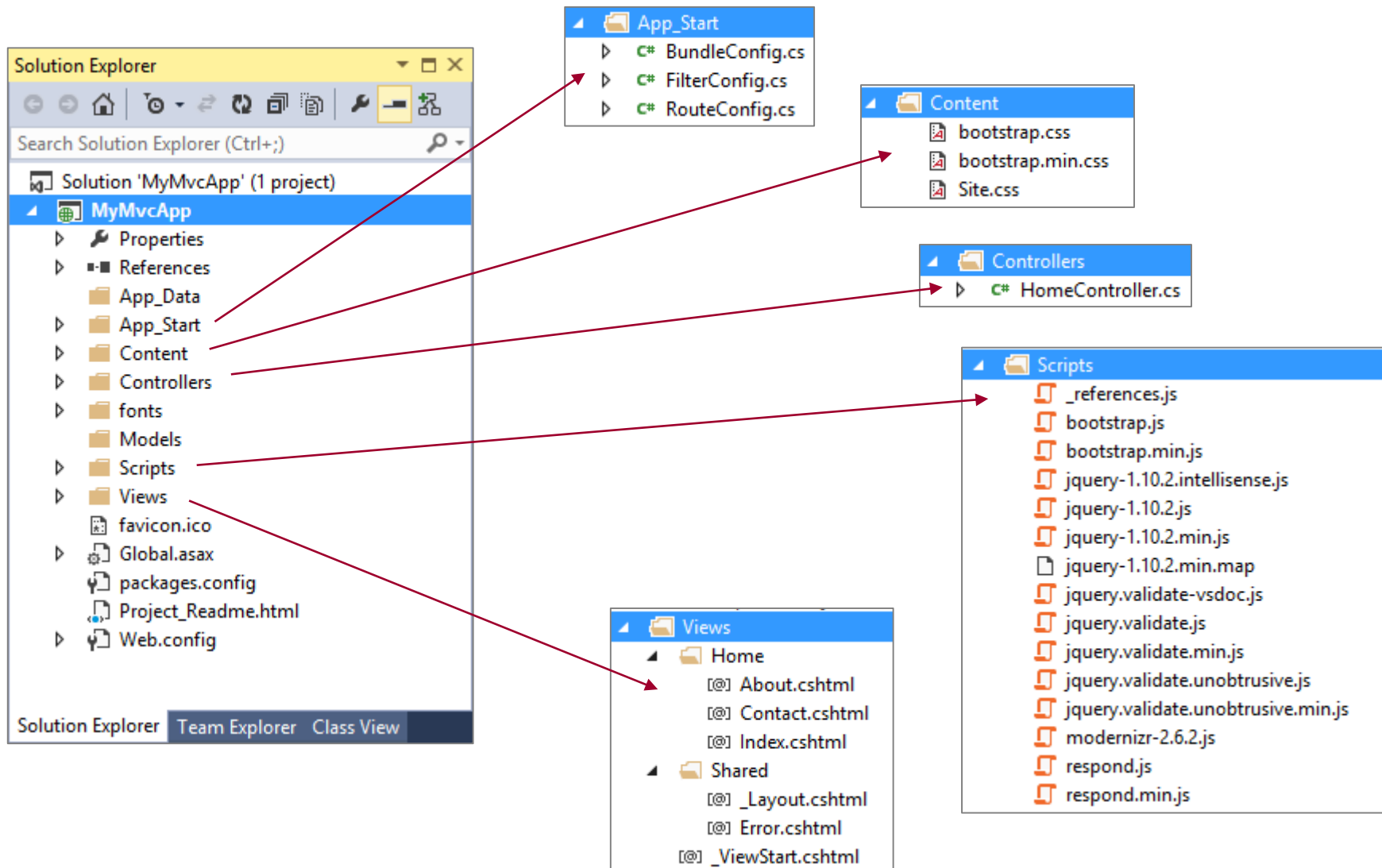


# Creating an MVC Project





# MVC App Project Files





**DEMO**

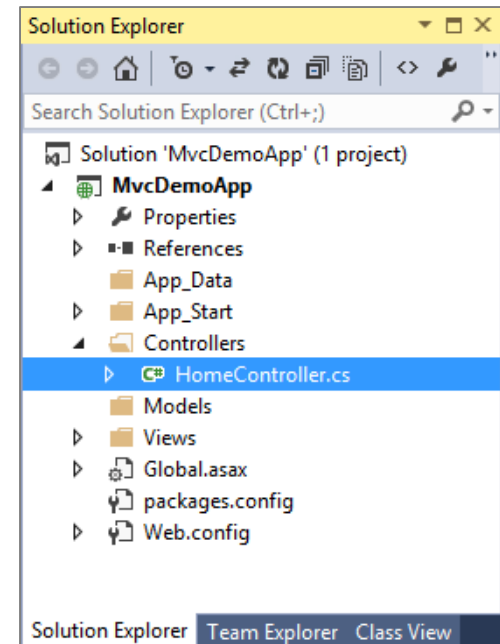
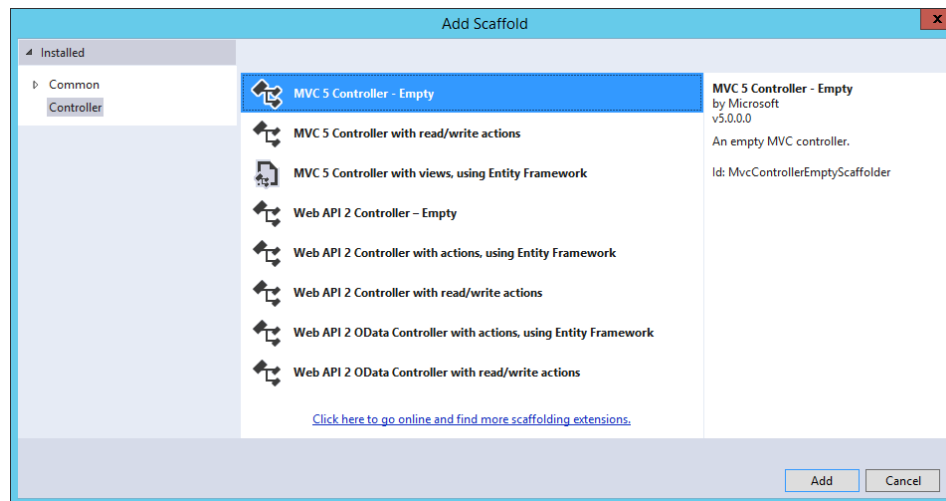
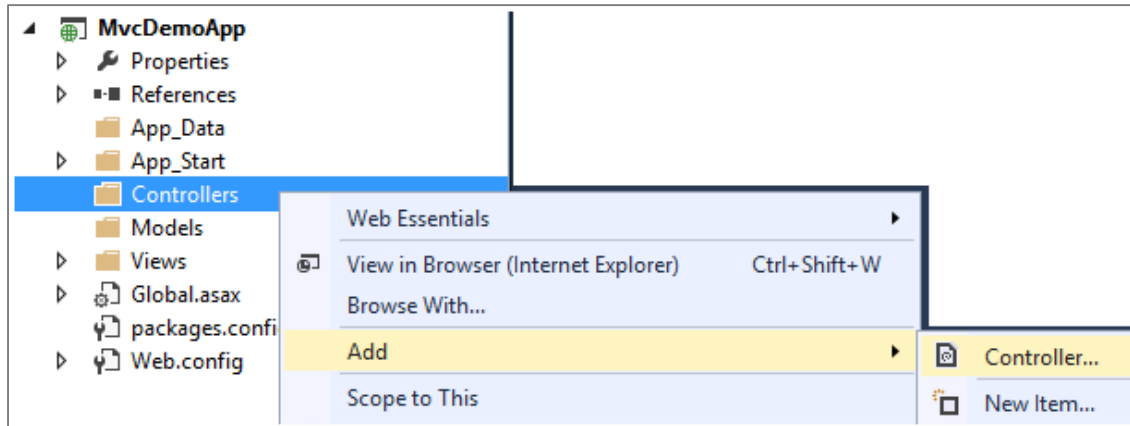
# Creating a Simple App with ASP.NET MVC

# Understanding Controllers

- A set of classes that manage...
  - processing incoming HTTP requests
  - communication to and from user
  - overall application flow and application-specific logic
- Every controller has one or more Actions
  - It's critical to understand the role of Actions in MVC

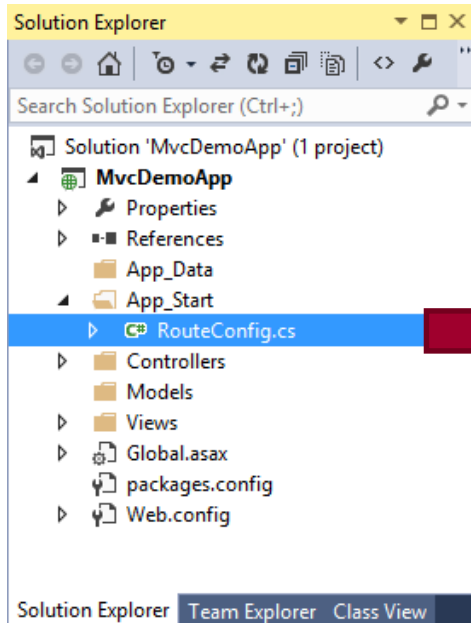


# Adding a Controller



# Wiring Up a Controller

- When you add a controller...
  - Visual Studio updates **RouteConfig** class
  - Routing scheme defined using standard format  
**{controller}/{action}/{id}**



```
public class RouteConfig {  
  
    public static void RegisterRoutes(RouteCollection routes) {  
  
        routes.IgnoreRoute("{resource}.axd/{*pathInfo}");  
  
        routes.MapRoute(  
            name: "Default",  
            url: "{controller}/{action}/{id}",  
            defaults: new { controller = "Home",  
                           action = "Index",  
                           id = UrlParameter.Optional }  
        );  
  
    }  
}
```

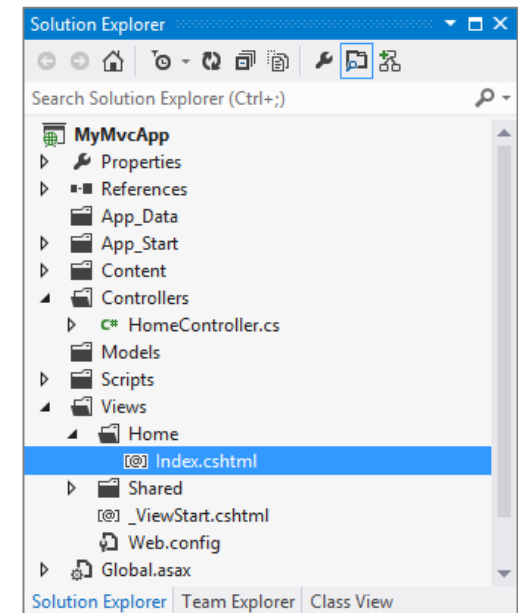
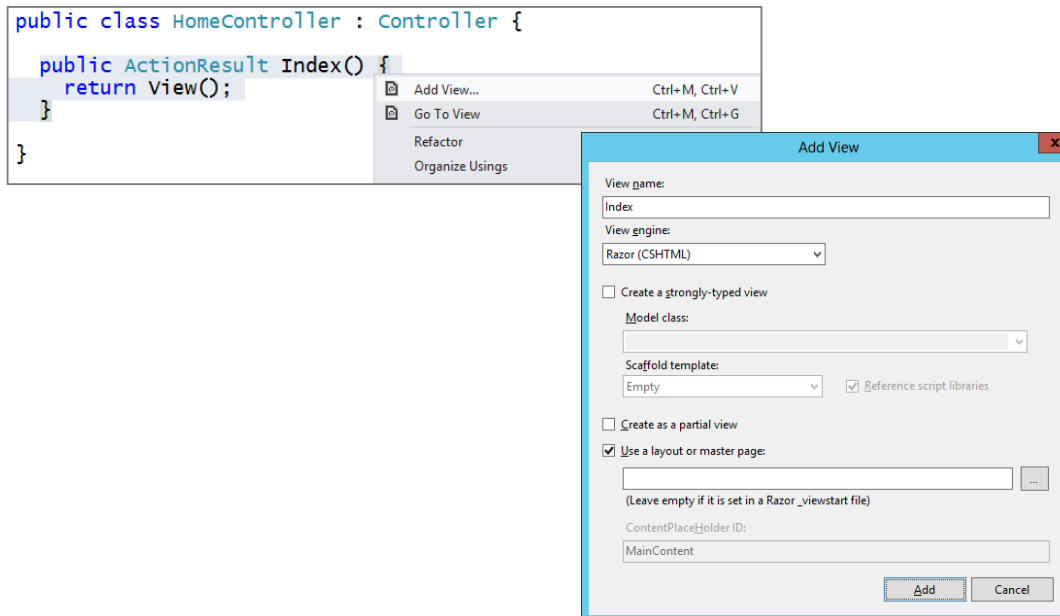


# Creating a View from a Controller Action

- Controller method often return **ActionResult**

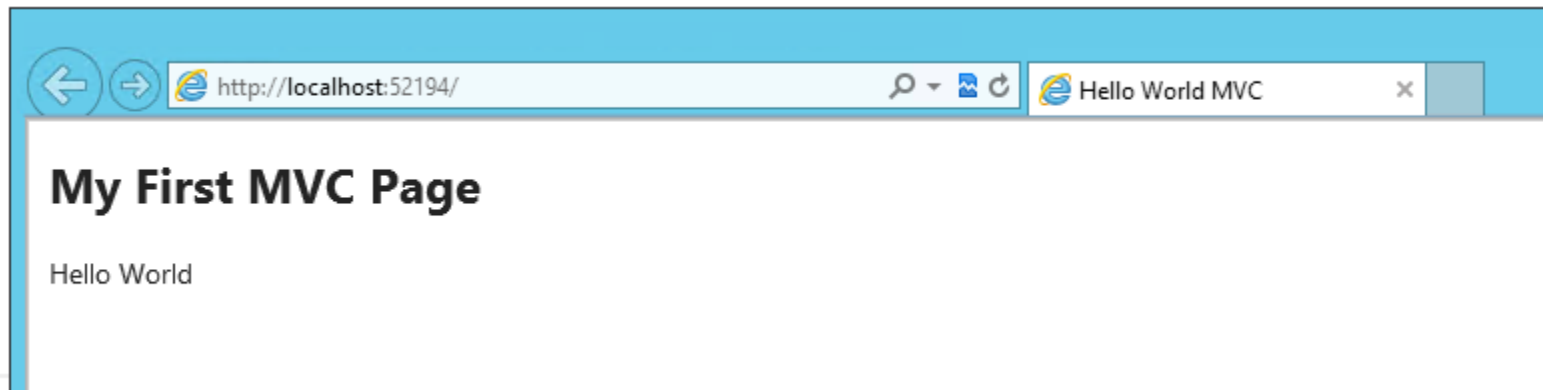
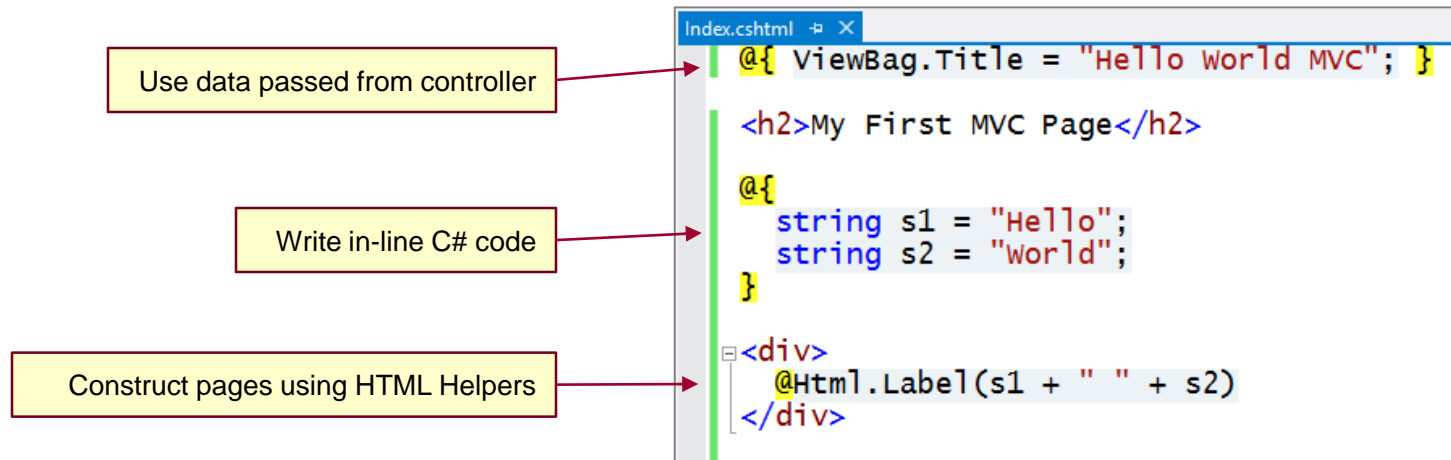
```
namespace MyMvcApp.Controllers {  
    public class HomeController : Controller {  
        public ActionResult Index() {  
            return View();  
        }  
    }  
}
```

- Right-click on Controller method to generate its view



# Customizing the View

- Views are created using the Razor engine
  - Provides a lean and elegant way to create HTML pages





# Customizing the Shared View

- MVC provides Shared Views
  - Provides same purpose as master pages in ASP.NET web forms
  - Default MVC shared view is named **\_ViewStart.cshtml**

```
index.cshtml
@{ ViewBag.Title = "Hello world MVC"; }

<h2>My First MVC Page</h2>

@{
    string s1 = "Hello";
    string s2 = "world";
}

<div>
    @Html.Label(s1 + " " + s2)
</div>
```

```
Layout.cshtml
<!DOCTYPE html>

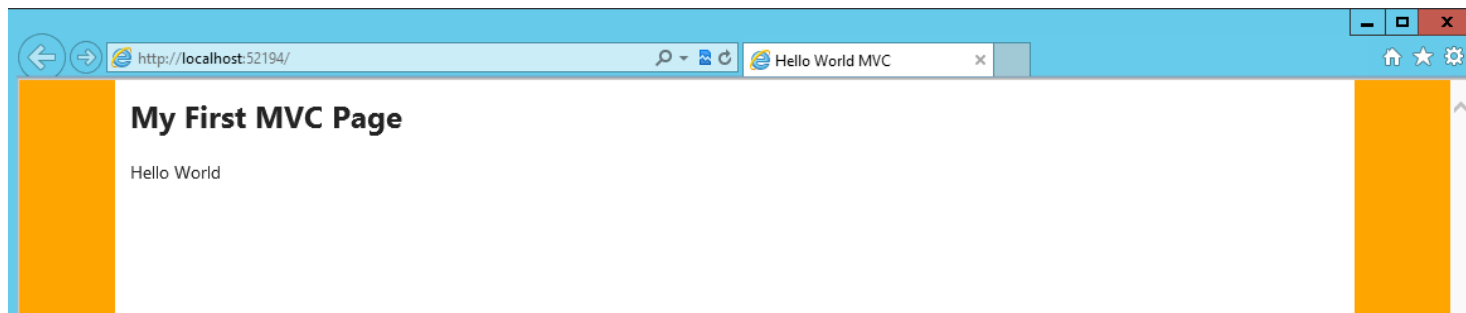
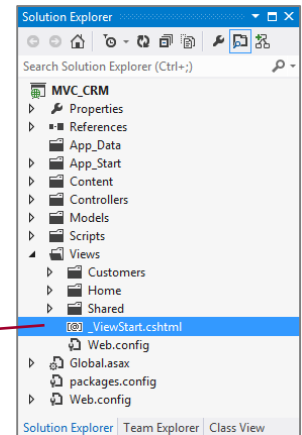
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width" />
    <title>@ViewBag.Title</title>
    @Styles.Render("~/Content/css")
    @Scripts.Render("~/bundles/modernizr")
</head>

<body style="background-color: orange; margin:0px; padding: 0px;">

    <div style="width:960px; margin:auto; background-color:white; min-height:480px;padding: 12px;" >
        @RenderBody()
    </div>

    @Scripts.Render("~/bundles/jquery")
    @RenderSection("scripts", required: false)

</body>
</html>
```



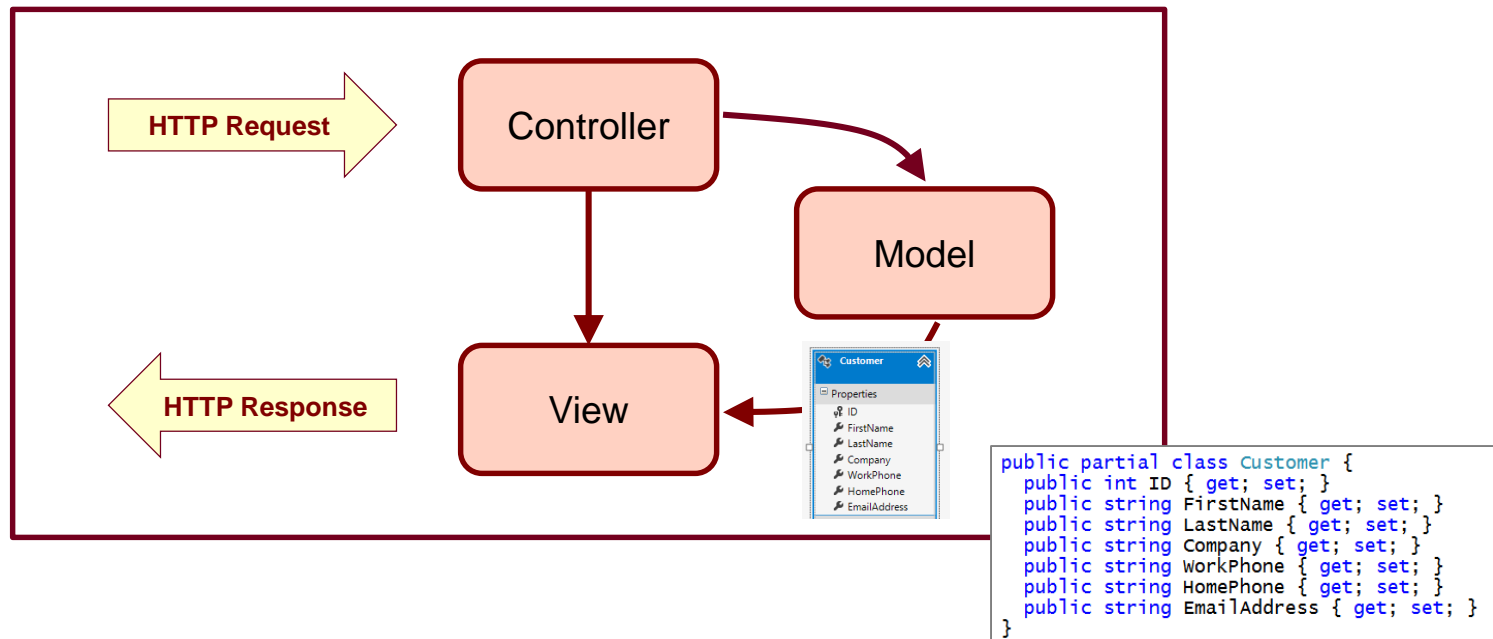


**DEMO**

## **Working with Controllers and Views**

# Motivation for Strongly-typed Models

- MVC designed based on strongly-typed models
  - Controller creates model object and passes it to view
  - Razor view engine supplies IntelliSense for model behind view
  - HTML helpers make it easy to create views and forms







**DEMO**

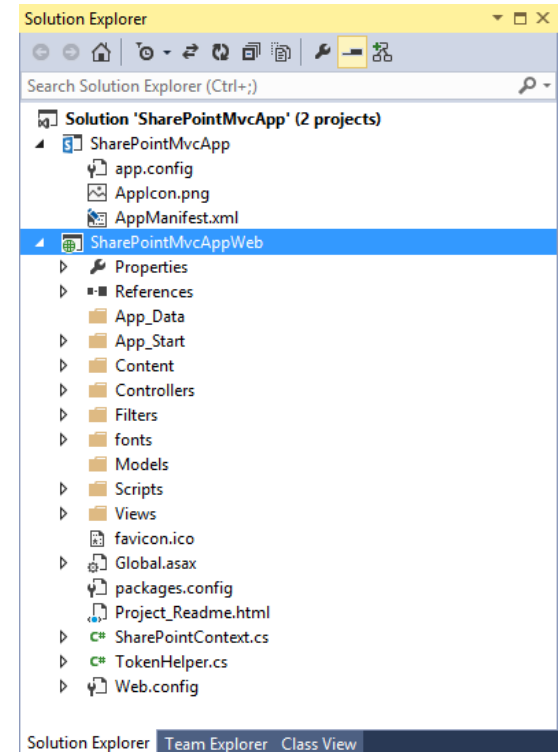
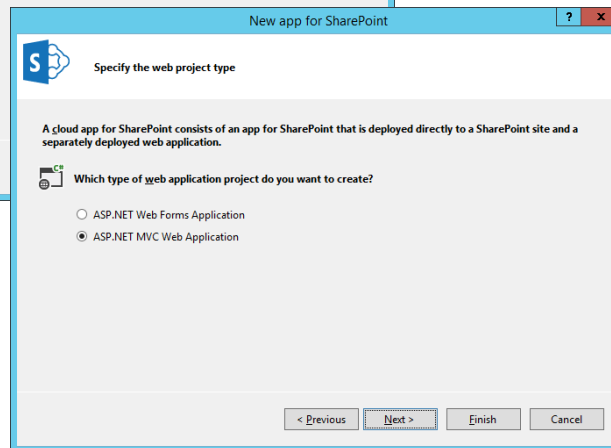
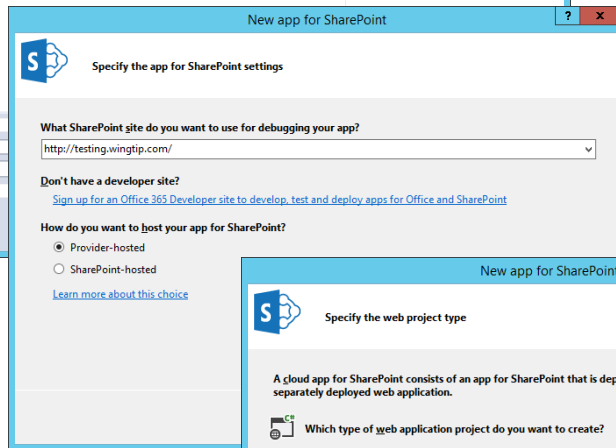
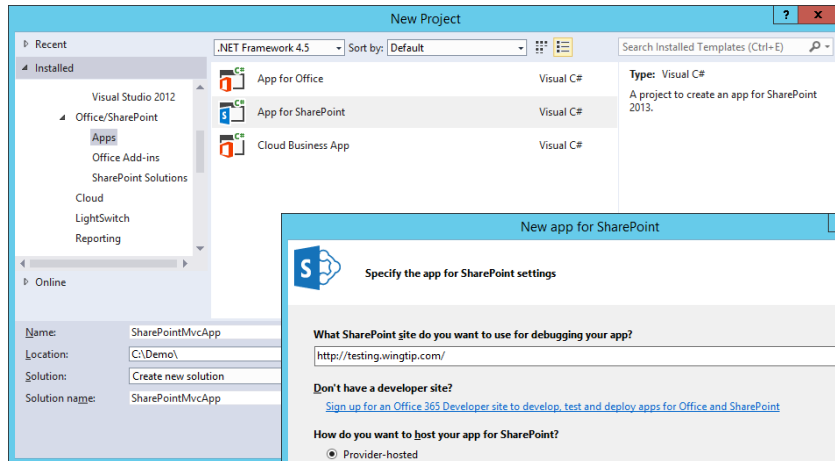
**Using a Strongly-typed Model**

# Agenda

- ✓ Getting Started with Provider-hosted Apps
- ✓ User Interface Design for the Remote Web
- ✓ Working with ASP.NET MVC
- Creating Provider-hosted Apps using MVC5

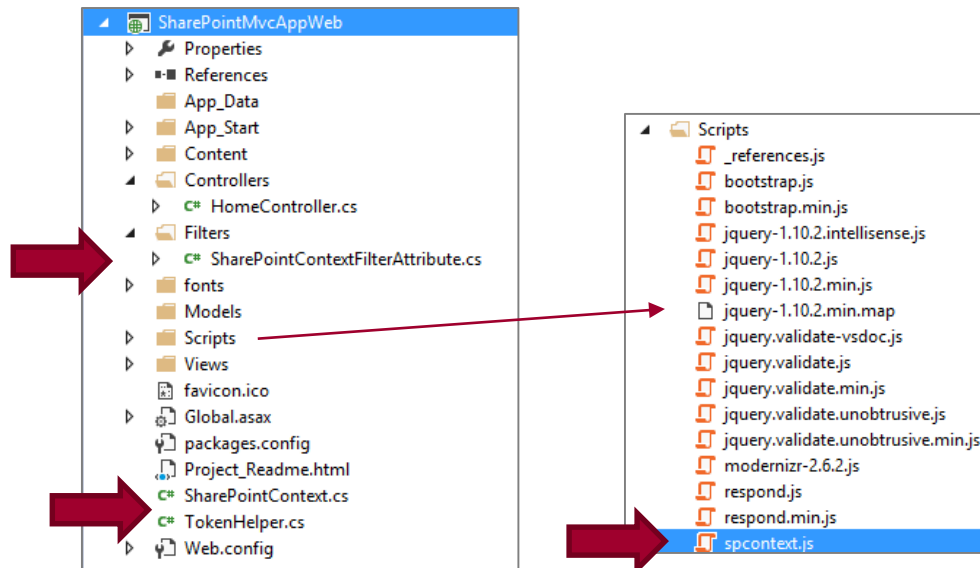


# Creating a Provider-hosted App with MVC



# SharePoint-specific Files

- TokenHelper.cs and SharePointContext.cs
  - Same as discussed earlier in course
- SharePointContextFilterAttribute.cs
  - Provides OAuth redirect logic for when context token is missing
  - Filter has no purpose when using app authentication based on S2S trust
- spcontext.js
  - Client-side JavaScript code to propagate SPHostUrl query string parameter







**DEMO**

# **Creating a Provider-hosted SharePoint App Using MVC5**

# Summary

- ✓ Getting Started with Provider-hosted Apps
- ✓ User Interface Design for the Remote Web
- ✓ Working with ASP.NET MVC
- ✓ Creating Provider-hosted Apps using MVC5

