## Programming with the Server-Side API

**Lab Time**: 60 minutes

**Lab Folder**: [[StudentFolder]]\DevServerSide

**Lab Overview**: In this lab you will learn how to create lists and populate them with data using the SharePoint 2013 server-side API. In addition, you will query the data using the two server-side query methods available: using SPQuery and LINQ for SharePoint.

### Exercise 1: Setup Lab Environment

In this exercise you will setup your environment.

All exercises in this lab assume you will work in a new site collection, http://serverside.wingtip.com.

1. Setup a new site collection for this lab:
   1. Ensure you are logged into the **WingtipServer** server as **WINGTIP\Administrator**.
   2. Run a PowerShell script, found in the root lab folder for this module:
      1. Right-click **SetupModule.ps1** and select **Run with PowerShell**. This file can be found in the files associated with this lab:

[..]\DevServerSide

* 1. When the script completes, it will launch a new browser and navigate to the lab site collection.
  2. Close the PowerShell console window.

### Exercise 2: Creating List Data

In this exercise you will create a feature to provision lists. Because LINQ code is tied to specific list schemas, your solutions will often contain a list-provisioning component.

1. Create a new project in Visual Studio 2012:
   1. Launch **Visual Studio 2012**: **Start 🡪 All Programs 🡪 Microsoft Visual Studio 2012 🡪 Visual Studio 2012**.
   2. Select **File 🡪 New 🡪 Project**.
   3. In the **New Project** dialog:
      1. Find the **SharePoint 2013 – Empty Project** template.
         1. **Templates 🡪 Visual C# 🡪 Office / SharePoint 🡪 SharePoint Solutions**
      2. **Name:** ListCreator
   4. Click **OK** to create the project.
   5. In the **SharePoint Customization Wizard**, use the following values to complete the wizard and click Finish.
      1. **What site do you want to use for debugging?** <http://serverside.wingtip.com>
      2. **What is the trust level for this SharePoint solution?** Deploy as farm solution
2. Create a Feature that will do the work of this solution:
   1. Using the **Solution Explorer** tool window, right-click the **Features** node and select **Add Feature**.
   2. In the Feature designer, set the following values:
      1. **Title:** Module: Server-Side Development – List Creator
      2. **Description:** Creates a new SharePoint list to store data for different data access methods using the server-side API.
      3. **Scope:** Web
   3. Add an image to the Feature:
      1. Using the **Solution Explorer** tool window, right-click the **ListCreator** project and select **Add 🡪 SharePoint “Images” Mapped Folder**.
      2. Right-click the **Images\ListCreator** folder you just created and select **Add 🡪 Existing Item**.
      3. In the **Add Existing Item** dialog, navigate to the following folder and add the **CPT\_FeatureIcon.gif** file:

[..]\ExtraStudentFiles\Images\CPT

* + 1. Using the **Solution Explorer** tool window, right-click **Feature1** and select **View Designer**.
    2. Using the **Properties** tool window, set the **Image URL** property to the following:

ListCreator\CPT\_FeatureIcon.gif

1. Add a Feature receiver that when activated, will create a new list and populate it with data:
   1. Using the **Solution Explorer** tool window, right-click **Feature1** and select **Add Event Receiver**.
   2. In the **Feature1.EventReceiver.cs** file, add the two following helper methods to the **Feature1EventReceiver** class:

private void FixupField(SPList spList, string fieldInternalName)

{

FixupField(spList.Fields.GetFieldByInternalName(fieldInternalName));

}

private void FixupField(SPField spField)

{

// This method takes an InternalName of a field in a spList

// and process a few things we want all fields to have by default

// for example setting them to show into the default view

spField.ShowInDisplayForm = true;

spField.ShowInEditForm = true;

spField.ShowInListSettings = true;

spField.ShowInNewForm = true;

spField.ShowInVersionHistory = true;

spField.ShowInViewForms = true;

// Add field to default view

SPView defaultView = spField.ParentList.DefaultView;

defaultView.ViewFields.Add(spField);

defaultView.Update();

spField.Update();

}

* 1. In the **Feature1.EventReceiver.cs** file, find the commented out **FeatureActivated()** method.
  2. Uncomment this method.
  3. Add the following code to the body of the **FeatureActivated()** method. The comments within the code explain what it is doing.

You can type this code or find it within in the files associated with this lab exercise: **[..]\DevServerSide\Exercises\Ex2\FeatureActivated.txt**

SPWeb spWeb = properties.Feature.Parent as SPWeb;

//Projects List

Guid pListGuid = spWeb.Lists.Add("Projects", "Company Projects", SPListTemplateType.GenericList);

spWeb.Update();

//Projects List columns

SPList pList = spWeb.Lists[pListGuid];

pList.OnQuickLaunch = true;

SPField pTitleIDField = pList.Fields["Title"];

FixupField(pList, pList.Fields.Add("Description", SPFieldType.Text, false));

FixupField(pList, pList.Fields.Add("Due Date", SPFieldType.DateTime, false));

SPFieldDateTime dueDateField = pList.Fields["Due Date"] as SPFieldDateTime;

dueDateField.DisplayFormat = SPDateTimeFieldFormatType.DateOnly;

dueDateField.Update();

pList.Update();

// Employees List

Guid eListGuid = spWeb.Lists.Add("Employees", "Employees", SPListTemplateType.GenericList);

spWeb.Update();

// Employees List columns

SPList eList = spWeb.Lists[eListGuid];

eList.OnQuickLaunch = true;

SPField titleIDField = eList.Fields["Title"];

titleIDField.Title = "Fullname";

titleIDField.Update();

FixupField(eList, eList.Fields.Add("JobTitle", SPFieldType.Text, false));

FixupField(eList, eList.Fields.Add("Team", SPFieldType.Text, false));

FixupField(eList, eList.Fields.Add("Contribution (in Milestones)", SPFieldType.Number, false));

string projectFieldInternalName = eList.Fields.AddLookup("Project", pListGuid, false);

SPFieldLookup projectField = eList.Fields.GetFieldByInternalName(projectFieldInternalName) as SPFieldLookup;

projectField.LookupField = pTitleIDField.InternalName;

FixupField(projectField);

eList.Update();

// Project Manager field (Project to Employee lookup)

string employeeFieldInternalName = pList.Fields.AddLookup("Primary Contact", eListGuid, false);

SPFieldLookup managerField = pList.Fields.GetFieldByInternalName(employeeFieldInternalName) as SPFieldLookup;

managerField.LookupField = titleIDField.InternalName;

FixupField(managerField);

pList.Update();

* 1. In the **Feature1.EventReceiver.cs** file, find the commented out **FeatureDectivating()** method.
  2. Uncomment this method.
  3. Add the following code to the **FeatureDectivating()** method to tear down the lists.

SPWeb spWeb = properties.Feature.Parent as SPWeb;

SPList empList = spWeb.Lists["Employees"];

empList.Delete();

spWeb.Update();

SPList projList = spWeb.Lists["Projects"];

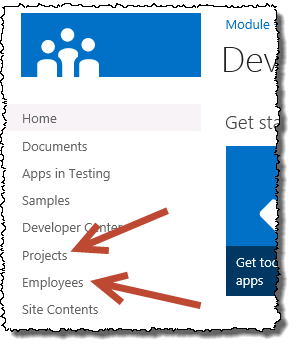
projList.Delete();

spWeb.Update();

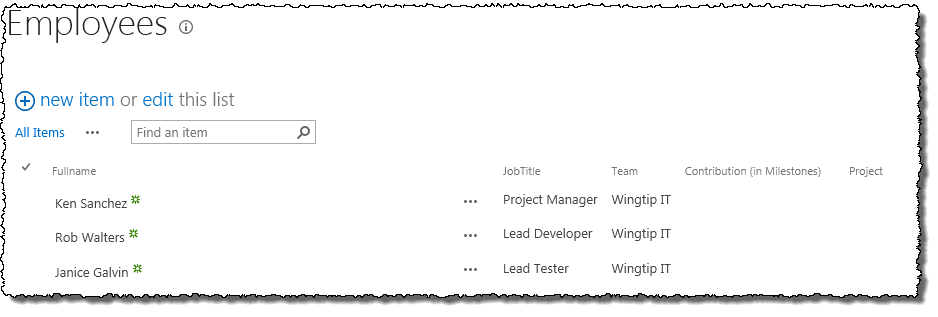
1. Save all changes: **File 🡪 Save All**.

#### Build and Test the Project

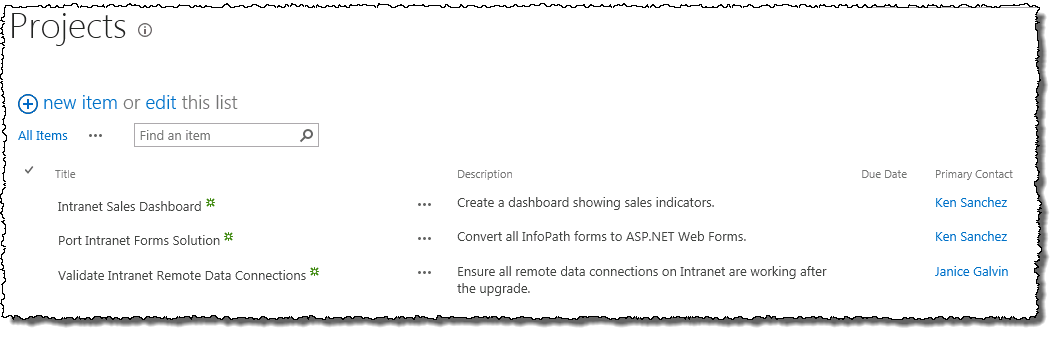
1. Build and test your application by pressing **[F5]** or **Debug 🡪 Start Debugging**.
2. Once the solution has been deployed, Internet Explorer will launch and navigate to the <http://serverside.wingtip.com> site.
3. Part of the debugging process involved activating the Feature you created so there should be a few new lists in the **Quick Launch** as shown:



1. Close the browser to stop the debugger and go back to Visual Studio.
2. In closing the browser, the Feature was deactivated which means the lists were deleted. Since everything is working, deploy the solution and manually activate the Feature:
   1. Using the **Solution Explorer** tool window, right-click the **ListCreator** project and select **Deploy**.
   2. After the solution is deployed, using Internet Explorer, navigate to <http://serverside.wingtip.com> and ensure the lists are showing in the Quick Launch navigation.
3. Add some sample data to the list:
   1. Using the **Quick Launch** navigation, select the **Employees** list.
   2. Add the following users to the list:



1. Using the **Quick Launch** navigation, select the **Projects** list.
2. Add the following projects to the list:



In this exercise you created two lists and added some data to them for future exercises.

### Exercise 3: Querying a List Using CAML & SPQuery

In this exercise you will create a Web Part that you will use to query the SharePoint lists using CAML and SPQuery.

1. Create a new project in Visual Studio 2012:
   1. Launch **Visual Studio 2012**: **Start 🡪 All Programs 🡪 Microsoft Visual Studio 2012 🡪 Visual Studio 2012**.
   2. Select **File 🡪 New 🡪 Project**.
   3. In the **New Project** dialog:
      1. Find the **SharePoint 2013 - Empty Project** template under the **Templates 🡪 Visual C# 🡪 Office / SharePoint 🡪 SharePoint Solutions** section.
      2. **Name:** CamlQueryWebPart
   4. Click **OK** to create the project.
   5. In the **SharePoint Customization Wizard**, use the following values to complete the wizard and click Finish.
      1. **What site do you want to use for debugging?** <http://serverside.wingtip.com>
      2. **What is the trust level for this SharePoint solution?** Deploy as farm solution
2. Add a Web Part to the project:
   1. Using the **Solution Explorer** tool window, right-click the **CamlQueryWebPart** project and select **Add 🡪 New Item**.
   2. In the **Add New Item** dialog, select the **Web Part** template from the **Visual C# Items 🡪 Office/SharePoint** category.
      1. **Name:** CamlQueryWebPart
   3. Click **Add**.
   4. Right-click the **CamlQueryWebPart** \ **CamlQueryWebPart.webpart** file and select **Open**.
      1. Update the **<property>** node that has the attribute **name=”Title”** and set the title to **CAML Query Web Part**.
   5. Now, modify the element manifest that will provision the Web Part definition:
      1. Using the **Solution Explorer** tool window, right-click the **CamlQueryWebPart \ Elements.xml** file and select **Open**.
      2. Modify the **<Property>** element by setting the **Value** attribute to **Wingtip Web Parts**.
3. Using the **Solution Explorer** tool window, right-click the **CamlQueryWebPart \ CamlQueryWebPart.cs** file and select **Open**.
4. Add code to the Web Part to query a SharePoint list:
   1. Add the following code to the **CreateChildControls()** method that creates a new **SPQuery** object and passes in a CAML query, selecting only the **Title** & **Description** fields:

SPList list = SPContext.Current.Web.Lists.TryGetList("Projects");

if (list != null)

{

SPQuery query = new SPQuery()

{

ViewFields = "<FieldRef Name='Title' /><FieldRef Name='Description' />",

Query = @"<OrderBy>

<FieldRef Name='Title' />

</OrderBy>"

};

SPListItemCollection listItems = list.GetItems(query);

}

1. Add the following code to the end of the **if** statement you just added to the **CreateChildControls()** method to display the results of the query:

foreach (SPListItem item in listItems)

{

string html = string.Format("<div><b>Project:</b>{0}<br /><em>{1}</em></div>", item["Title"].ToString(), item["Description"].ToString());

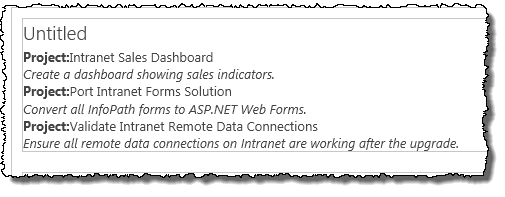
this.Controls.Add(new LiteralControl(html));

}

1. Save all changes: **File 🡪 Save All**.

#### Build and Test the Project

1. Build and test your application by pressing **[F5]** or **Debug 🡪 Start Debugging**.
2. Once the solution has been deployed, Internet Explorer will launch and navigate to the <http://serverside.wingtip.com> site.
3. Add the Web Part to the page:
   1. Using the **Site Actions** “gear” icon in the top-right corner, select **Edit Page**.
   2. Using the ribbon, select the **Page** tab, then click the **Insert** tab and click the **Web Part** button.
   3. Select the **CAML Query Web Part** from the **Wingtip Web Parts** category and click the **Add** button.
4. Notice that the Web Part is displaying the contents of the SharePoint list:



1. Close the browser to stop the debugger and go back to Visual Studio.

In this exercise you created a Web Part that used the SPQuery object with a CAML query to retrieve data from SharePoint.

### Exercise 4: Querying a List Using LINQ for SharePoint

In this exercise you will create entities for use with LINQ. Entity creation is done by using the command line utility SPMetal.

1. Generate an entity data model for the site containing the sample data:
   1. Open **SharePoint 2013 Management Shell**: **Start 🡪 All Programs 🡪 Microsoft SharePoint 2013 Products 🡪 SharePoint 2013 Management Shel**l.
   2. Execute the following command in the SharePoint 2013 Management Shell:

SPMetal.exe /web:http://serverside.wingtip.com /code:”c:\ServerSideEntities.cs”

1. Create a new project in Visual Studio 2012:
   1. Launch **Visual Studio 2012**: **Start 🡪 All Programs 🡪 Microsoft Visual Studio 2012 🡪 Visual Studio 2012**.
   2. Select **File 🡪 New 🡪 Project**.
   3. In the **New Project** dialog:
      1. Find the **SharePoint 2013 – Empty Project** template under the **Templates 🡪 Visual C# 🡪 Office / SharePoint 🡪 SharePoint Solutions** section.
      2. **Name:** LinqQueryWebPart
   4. Click **OK** to create the project.
   5. In the **SharePoint Customization Wizard**, use the following values to complete the wizard and click Finish.
      1. **What site do you want to use for debugging?** <http://serverside.wingtip.com>
      2. **What is the trust level for this SharePoint solution?** Deploy as farm solution
2. Add the LINQ entity model you created using SPMetal.exe to the project:
   1. Using the **Solution Explorer** tool window, right-click the **LINQQueryWebPart** project and select **Add 🡪 Existing Item**:
   2. In the **Add Existing Item** dialog, select the following file and click **Add**:

c:\ServerSideEntities.cs

1. Add a reference to the Microsoft LINQ assembly:
   1. Using the **Solution Explorer** tool window, right-click the **LinqQueryWebPart** project and select **Add Reference**.
   2. Add a reference to the **Microsoft.SharePoint.Linq.dll** found in the following location:

C:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\15\ISAPI

1. Add a Web Part to the project:
   1. Using the **Solution Explorer** tool window, right-click the **LinqQueryWebPart** project and select **Add 🡪 New Item**.
   2. In the **Add New Item** dialog, select the **Web Part** template from the **Visual C# Items 🡪 Office/SharePoint** category.
      1. **Name:** LinqQueryWebPart
   3. Click **Add**.
   4. Right-click the **LinqQueryWebPart \ LinqQueryWebPart.webpart** file and select **Open**.
      1. Update the **<property>** node that has the attribute **name=”Title”** and set the title to **LINQ Query Web Part**.
   5. Now, modify the element manifest that will provision the Web Part definition:
      1. Using the **Solution Explorer** tool window, right-click the **LINQQueryWebPart \ Elements.xml** file and select **Open**.
      2. Modify the **<Property>** element by setting the **Value** attribute to **Wingtip Web Parts**.
2. Using the **Solution Explorer** tool window, right-click the **LINQQueryWebPart \ LINQQueryWebPart.cs** file and select **Open**.
3. Add the following lines to the top of the **LINQQueryWebPart.cs** file:

using System.Linq;

1. Add code to the Web Part to query a SharePoint list:
   1. Add the following code to the **CreateChildControls()** method that issues a query using a LINQ expression:

ServerSideEntitiesDataContext siteDataContext = new ServerSideEntitiesDataContext(SPContext.Current.Web.Url);

var query = from project in siteDataContext.Projects

orderby project.Title

select new

{

ProjectTitle = project.Title,

ProjectDescription = project.Description,

ProjectContactName = project.PrimaryContact.Title,

ProjectContactJobTitle = project.PrimaryContact.JobTitle

};

Notice how in the previous code sample the anonymous type has two properties coming from a relational list: **ProjectContactName** & **ProjectContactJobTitle**. When you added the code **project.PrimaryContact**, the query essentially jumped over to the related list because the field **PrimaryContact** in the Projects list is a lookup column to the Employees list. LINQ for SharePoint handled all the code that was needed to establish the join for you. Doing this in CAML is much more tedious and prone to errors.

1. Add the following code to the end of the **CreateChildControls()** method to display the results of the query:

foreach (var item in query) {

string html = string.Format("<div><b>Project:</b> {0}<br />" +

"<em>{1}</em><br />&nbsp;&nbsp;<b>Primary Contact:</b> {2} ({3})" +

"</div>",

item.ProjectTitle,

item.ProjectDescription,

item.ProjectContactName,

item.ProjectContactJobTitle);

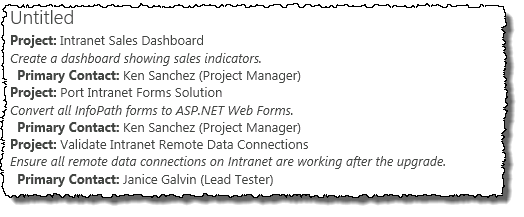
this.Controls.Add(new LiteralControl(html));

}

1. Save all changes: **File 🡪 Save All**.

#### Build and Test the Project

1. Build and test your application by pressing **[F5]** or **Debug 🡪 Start Debugging**.
2. Once the solution has been deployed, Internet Explorer will launch and navigate to the <http://serverside.wingtip.com> site.
3. Add the Web Part to the page:
   1. Using the **Site Actions** “gear” icon in the top-right corner, select **Edit Page**.
   2. Using the ribbon, select the **Page** tab, then click the **Insert** tab and click the **Web Part** button.
   3. Select the **LINQ Query Web Part** from the **Wingtip Web Parts** category and click the **Add** button.
4. Notice that the Web Part is displaying the contents of the SharePoint list:



1. Close the browser to stop the debugger and go back to Visual Studio.

In this exercise you created a Web Part that used a LINQ query to retrieve data from SharePoint.