# OFFICE AMS: Data storage models

|  |  |
| --- | --- |
| Summary: | Applies to: |
|  | * Office 365 Multi Tenant (MT) * Office 365 Dedicated (d) * SharePoint 2013 on-premises |
| Solution: |  |
| Author: | Todd Baginski (Canviz LLC)  Cloris Sun (Canviz LLC)  Tyler Lu(Canviz LLC)  Lucas Smith (Canviz LLC)  Cindy Yan (Canviz LLC)  Michael Sherman (Canviz LLC) |
| //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // THIS CODE IS PROVIDED \*AS IS\* WITHOUT WARRANTY OF  // ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING ANY  // IMPLIED WARRANTIES OF FITNESS FOR A PARTICULAR  // PURPOSE, MERCHANTABILITY, OR NON-INFRINGEMENT.  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* | |

# Scenario: Data storage models – Composite Business Applications

This provider-hosted sample application for SharePoint demonstrates the differences, advantages, and disadvantages between different data storage patterns associated with the App Model and how they are built. It also illustrates limitations associated with certain data storage components that should be considered when deciding which data storage components to use when building with the App Model.

The following diagram illustrates the different data storage components in the sample. These data storage components, the APIs used to access them, and the user interfaces used to interact with them are described in detail later in this document.



# Configuration & Deployment

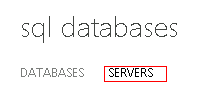
In order for the data storage models sample to function correctly, you must configure SQL Azure, Azure Storage, and a SharePoint site collection. The following sections describe how to do this.

The configuration and deployment steps associated with this sample take between 10-20 minutes to accomplish.

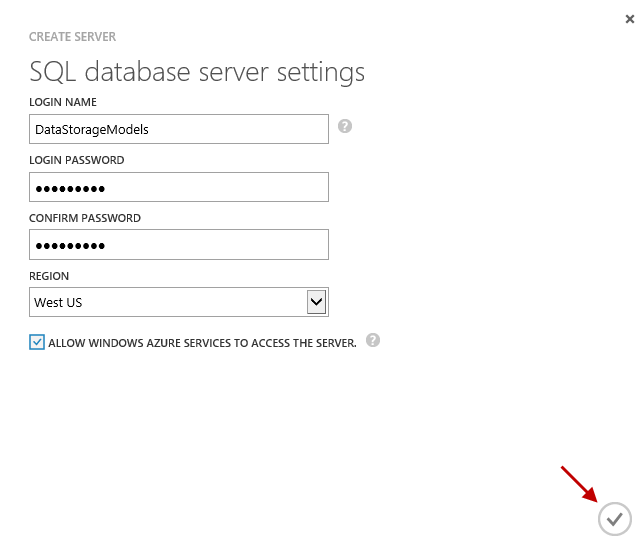
## Deploy northwind database to sql azure

To deploy the Northwind database to an Azure SQL Database perform the following steps.

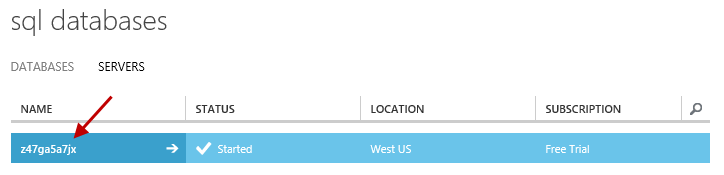
1. Log into the [Azure Management Portal](https://manage.windowsazure.com/)
2. Click **SQL DATABASES**.
3. Click **SERVERS**.



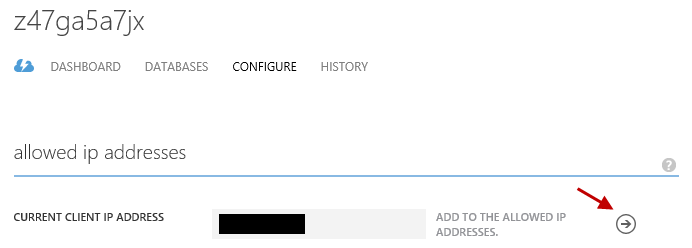
1. Click **CREATE A SQL DATABASE SERVER**.
2. In the CREATE SERVER form, enter a **login name** and create **new password** for this server and choose a **region**.



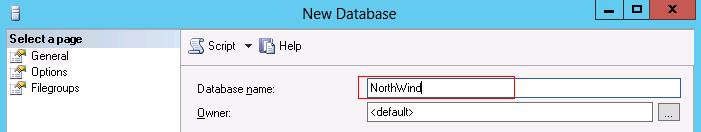
1. Click the **checkmark button** (indicated with the red arrow in the screenshot above) to create the server.
2. After the server is created, click the **server name** in the list of servers.



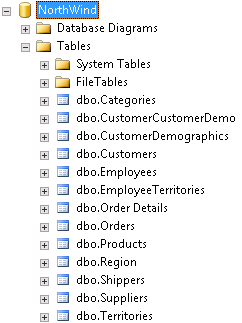
1. Click **CONFIGURE**.
2. Click the **arrow** to the right of ADD TO THE ALLOWED IP ADDRESSES, this arrow is pictured in the screenshot below.



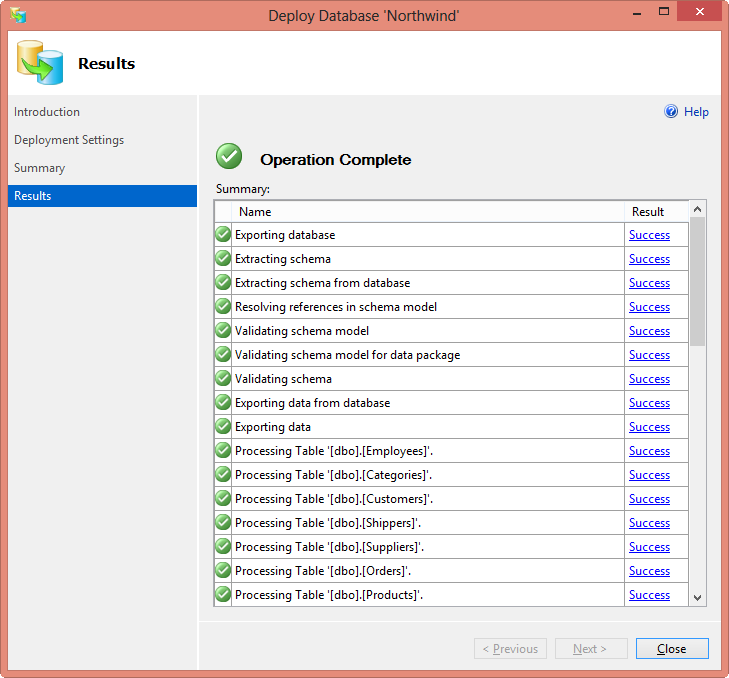
1. At the bottom of the page, click **SAVE**.
2. Open SQL Server Management Studio (2012) on your local development machine and create a new database named **NorthWind**.



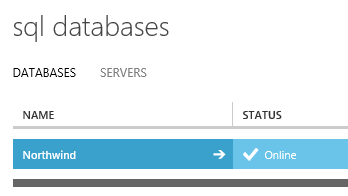
1. In the **Object Explorer**, select the **Northwind** database.
2. Click **New Query**.
3. In a text editor, open the **northwind.sql SQL script** provided with the sample.
4. **Copy** all the text in the **northwind.sql** file.
5. **Paste** all the text into the **SQL Query window** in the SQL Server Management Studio.
6. Click **Execute**.



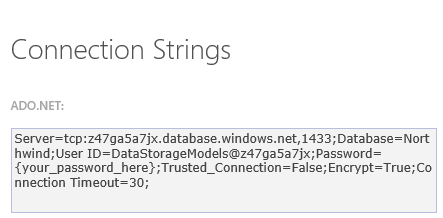
1. In the Object Explorer, right click on the **Northwind** database
2. Select **Tasks**, then select **Deploy Database to SQL Azure**.
3. Click the **Next** **>** buttonon the Introduction screen.
4. Click the **Connect…** button, and enter the Server name for the SQL Azure Database Server you previously created.
5. In the Authentication dropdown select **SQL Server Authentication**.
6. Enter the **Login** and **Password** you previously specified when you created the Azure SQL Database server.
7. Click the **Connect** button.
8. Click the **Next >** button
9. Click the **Finish** button.
10. Wait until the database is created. After the database is successfully created, click the **Close** button to close the wizard.



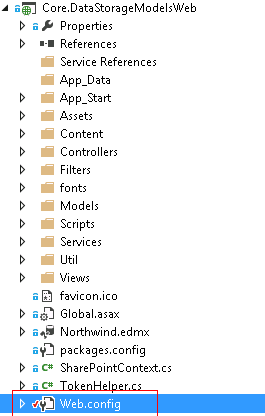
1. Return to the Azure Management Portal <https://manage.windowsazure.com/>, to verify the Northwind Database has been successfully created.



1. Click the **Northwind** database.
2. Select **View SQL Database connection strings**.
3. Copy the **ADO.NET connection string** and paste it into a text file on your local machine; you will need it later.



1. Click the **X button** to close the dialog.
2. Click the **Set up Windows Azure firewall rules for this IP address** link to add your IP address to the firewall rules to allow you to access the database.
3. Open the ***Core.DataStorageModels.sln***file in Visual Studio 2013*.*
4. Open the **Web.config** file.



1. Refer to the template below to update the **connectionString** named **NorthWindEntities**. Replace the highlighted portions with connection string information you saved to the text file on your local machine.

<add name="NorthWindEntities" connectionString="metadata=res://\*/Northwind.csdl|res://\*/Northwind.ssdl|res://\*/Northwind.msl;provider=System.Data.SqlClient;provider connection string=&quot;data source=<Your Server Here>.database.windows.net;initial catalog=NorthWind;user id=<Your Username Here>@<Your Server Here>;password=<Your Password Here>;MultipleActiveResultSets=True;App=EntityFramework&quot;" providerName="System.Data.EntityClient" />

1. **Save** the file.
2. **Do not close** Visual Studio 2013.

## CREATE AN AZURE STORAGE ACOUNT

1. Return to the [Azure Management Portal](https://manage.windowsazure.com/) and click **ALL ITEMS**.

AT THE BOTTOM OF THE NAVIGATION PANE, CLICK **NEW**.

+new

1. Click **DATA SERVICES**, then **STORAGE**, and then click **QUICK CREATE**.
2. In the **URL** textbox enter **datastoragemodel**
3. Click **CREATE STORAGE ACCOUNT.**
4. After the Storage Account is created successfully, at the bottom of the navigation pane click **MANAGE ACCESS KEYS**.



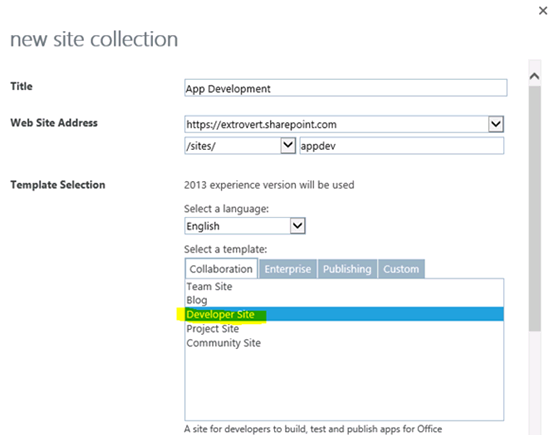
1. Copy the **STORAGE ACCOUNT NAME** and **PRIMARY ACCESS KEY** values and paste them into a text file on your local machine; you will need them later.
2. Return to the **web.config** file you previously edited in Visual Studio.
3. Refer to the template below to update the **key** named **StorageConnectionString**. Replace the highlighted portions with the Storage Account information you just saved.

<add key="StorageConnectionString" value="DefaultEndpointsProtocol=https;AccountName=<Your Data Storage Account Name Here>;AccountKey=<Your Primary Access Key Here>" />

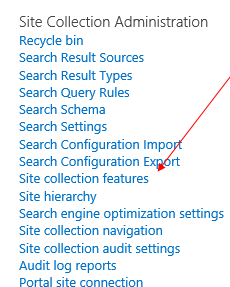
1. Save the **web.config** file.
2. **Do not close** Visual Studio 2013.

## SharePoint

1. Navigate to your O365 SharePoint tenancy and **create a new site collection** using the **Developer Site template** in the **Collaboration** **tab**.



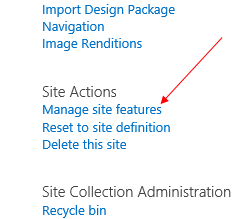
1. When you create the site collection, enter **1300** in the **Storage Quota** textbox**.**
2. Once the site collection is created, navigate to **Site settings** and select **Site collection features** under the **Site Collection Administration** heading.



1. In the **Site collection features** page, locate the **SharePoint Server Publishing Infrastructure** feature and click **Activate**. Be patient, sometimes this feature may take several minutes to activate.



1. Return to **Site settings** and select **Manage Site Features** under the **Site Actions** heading.



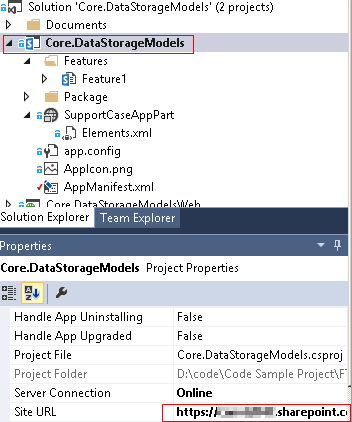
1. Locate the **SharePoint Server Publishing** feature and select **Activate**.



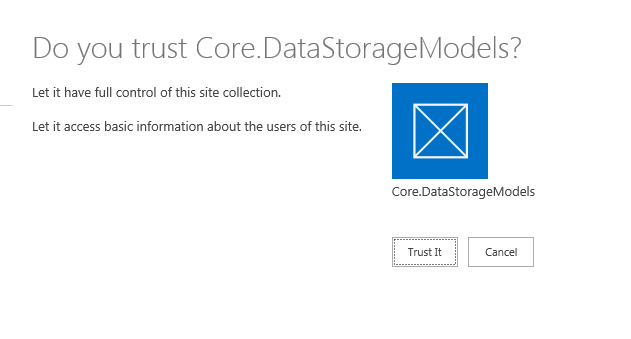
# Installing the App

Now that all of the prerequsites are completed the SharePoint app may be deployed.

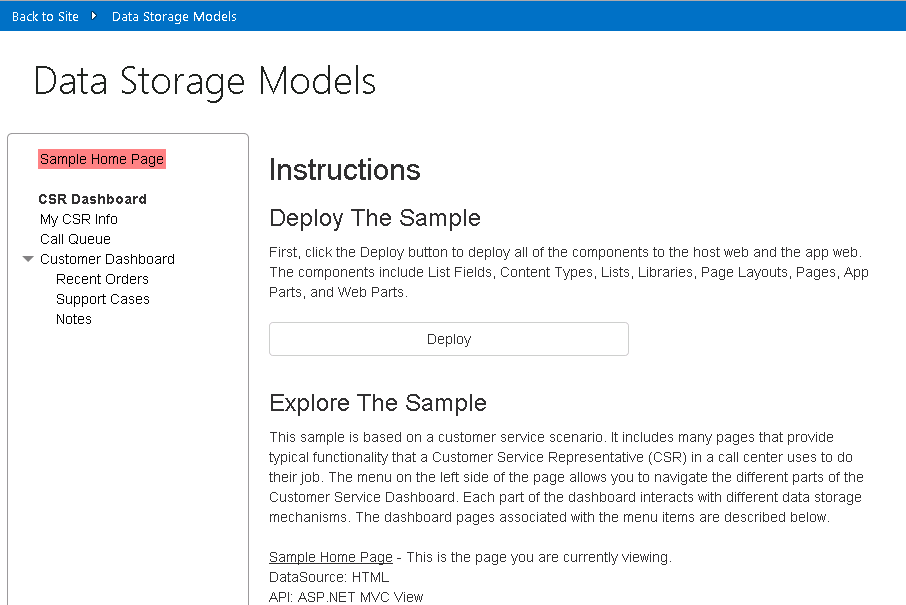
1. Return to the ***Core.DataStorageModels***solution in Visual Studio 2013.
2. In the Solution Explorer, select the ***Core.DataStorageModelsWeb***project.
3. In the **Properties** window, set the **Site URL** property to the site collection you previously created and configured.



1. Press **F5** or click the **Start button** in Visual Studio 2013.
2. Enter your **user name** and **password** to connect to your SharePoint site collection.
3. After your username and password have been verified, the trust dialog is displayed. Click the **Trust It** button.

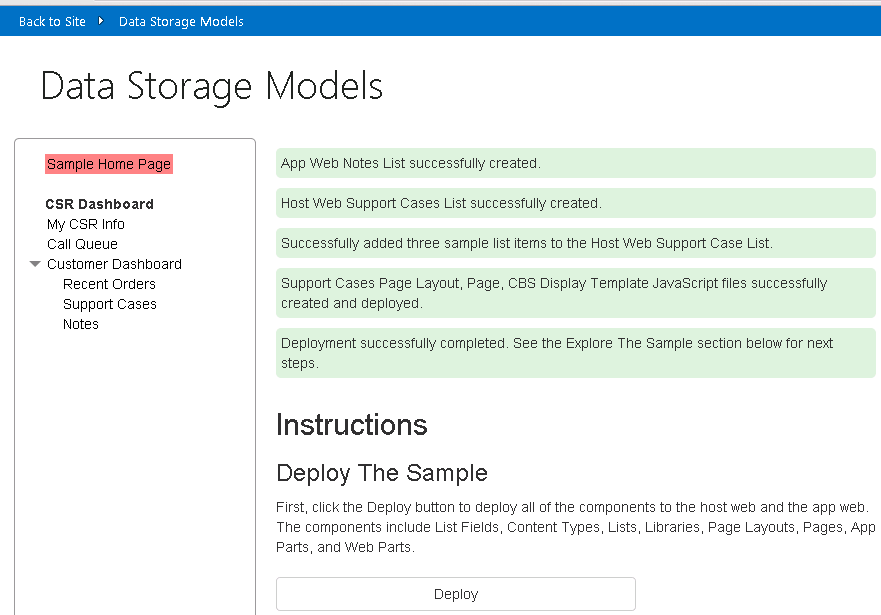


1. After app installation, a new page will be displayed. This page describes how to deploy, explore, and interact with the sample. It is pictured below.



# Deploying COMPONENTS to the host and App Webs

1. Click the **Deploy** button to deploy all of the SharePoint components in the sample. Success messages are displayed once the tasks are completed.



The Explore the Sample section on landing page in the SharePoint app includes high level documentation which describes the components in the sample. The following documentation provides additional details and instructions which describe and illustrate how to interact with the data storage model scenarios in the app.

# Data Storage Models Scenarios

The purpose of this sample is to show the advantages and disadvantages of using different types of storage models to hold information.

There are 6 different scenarios in this sample. Each scenario is supported with specialized information. The diagram in the first section in this document illustrates the different data storage models associated with these scenarios.

* Customer Dashboard
* Recent Orders
* Customer Service Representative Survey
* Notes
* Support Cases
* Call Queue

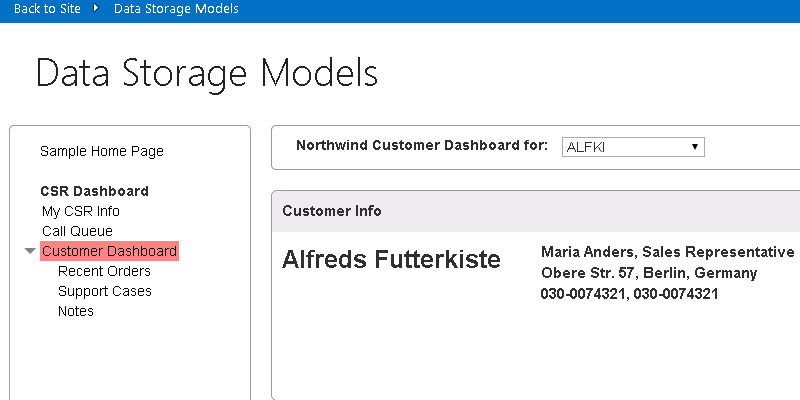
# Customer Dashboard

The Customer Dashboard scenario uses JQuery AJAX to invoke the NorthWind OData Service to return a customer’s information.

The advantages of using this method of data storage and delivery include all the typical advantages of a service oriented architecture. For more information about SOA and it’s benefits see the following MSDN article. <http://msdn.microsoft.com/en-us/library/bb833022.aspx>

Specifically related to the App Model, the advantages of using this method of data storage and delivery include:

1. Design
   1. A single service may be used by more than one SharePoint app.
   2. Services may be updated independently of SharePoint apps. This allows developers to update business logic without redeploying the SharePoint app.
2. Performance
   1. Service performance is not affected by SharePoint and hosting services in an environment such as Microsoft Azure allows services to scale easily to ensure good performance.
3. Backup/Restore
   1. Services may be backed up and restored seperately from a SharePoint infrastructure.
   2. Services which do not access SharePoint data are not affected when a SharePoint app is uninstalled unless the SharePoint App has code added to it which explicitly interacts with the service or the data it accesses.
4. To access customer details, click the **Customer Dashboard** link in the left menu.
5. Select a C**ustomer** in the drop down menu.



## Code

This page is an MCV view defined in the CustomerDashboard\Home.cshtml file. This page uses JQuery AJAX to invoke the NorthWind OData Service. The JavaScript code is located in the Scripts/CustomerDashboard.js file.

First, when the page loads, the Northwind OData Service is called to retrieve all of the customers and their associated CustomerIDs.

Note: This dropdown control exists on many pages in the sample.

var getCustomerIDsUrl = "https://odatasampleservices.azurewebsites.net/V3/Northwind/Northwind.svc/Customers?$format=json&$select=CustomerID";

$.get(getCustomerIDsUrl).done(getCustomerIDsDone)

.error(function (jqXHR, textStatus, errorThrown) {

$('#topErrorMessage').text('Can\'t get customers. An error occurred: ' + jqXHR.statusText);

});

When a customer is selected in the dropdown list, the Northwind OData Service is called to retrieve the details for the currently selected customer.

var url = "https://odatasampleservices.azurewebsites.net/V3/Northwind/Northwind.svc/Customers?$format=json" + "&$select=CustomerID,CompanyName,ContactName,ContactTitle,Address,City,Country,Phone,Fax" + "&$filter=CustomerID eq '" + customerID + "'";

$.get(url).done(getCustomersDone)

.error(function (jqXHR, textStatus, errorThrown) {

alert('Can\'t get customer ' + customerID + '. An error occurred: ' +

jqXHR.statusText);

});

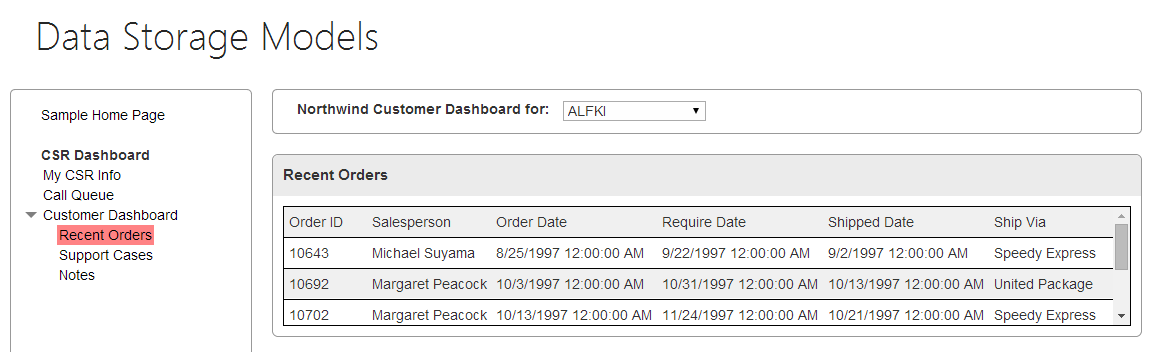
# Recent Orders

The Recent Orders scenario uses a direct call to the Northwind SQL Azure Database to return all the orders for a given customer.

The advantages of using this method of data storage and delivery include:

* Design
  + The database scenario can utilize many-to-many relationships.
  + Tooling is available for database design.
  + A single database may be used by more than one SharePoint app.
  + Database may be updated independently of SharePoint apps as long as the schema changes do not affect the SharePoint app. This allows developers to update data stores without redeploying the SharePoint app.
* Performance
  + Databases typically offer better performance when executing queries that involve many joins and other operations such as calculations when compared to SharePoint lists.
* Backup/Restore
  + The SQL database allows for backup and restore functionality, making it easier to roll back the data if necessary.
  + External databases are not affected when a SharePoint app is uninstalled unless the SharePoint App has code added to it which explicitly interacts with database it accesses.
* Import/Export
  + The SQL database allows for importing and exporting the data and columns, which enables administrators to easily move and manage the database.

1. To access the Recent Orders, expand the Customer Dashboard by clicking the arrow and select **Recent Orders**.
2. Select a C**ustomer** in the drop down menu to view the orders for a customer.



## Code

This page is an MCV view defined in the Orders.cshmtl file. The code in the CustomerDashboardController uses the Entity Framework to query the Orders table and joins the Customer, Employee and Shipper tables. The customer ID is retrieved from the query string in the URL (set by the dropdown control) and passed as a query parameter to the query. Finally, the result of the query is returned to the MVC view where the results are rendered.

public ActionResult Orders(string customerId)

{

Order[] orders;

using (var db = new NorthWindEntities())

{

orders = db.Orders

.Include(o => o.Customer)

.Include(o => o.Employee)

.Include(o => o.Shipper)

.Where(c => c.CustomerID == customerId)

.ToArray();

}

ViewBag.SharePointContext =   
 SharePointContextProvider.Current.GetSharePointContext(HttpContext);

return View(orders);

}

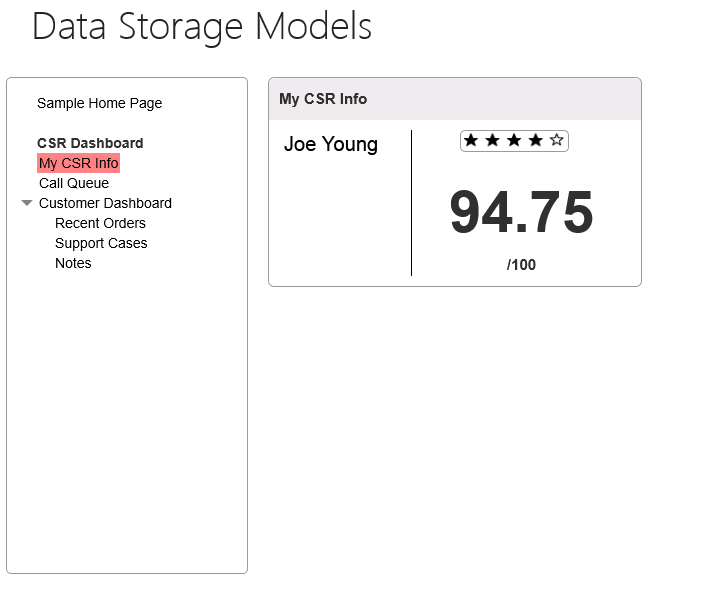
# Customer Service Representative Survey Scenario

The CSR survey scenario allows a customer service representative to see their rating based on customer surveys and utilizes Azure Table Storage and the Microsoft.WindowsAzure.Storage.Table.CloudTable API to store and interact with the data.

The advantages of using this method of data storage and delivery include:

* Design
  + Azure Storage Tables may be used by more than one SharePoint app.
  + Azure Storage Tables may be updated independently of SharePoint apps as long as the schema changes do not affect the SharePoint app. This allows developers to update data stores without redeploying the SharePoint app.
* Performance
  + Azure Storage Table performance is not affected by SharePoint and scales easily to ensure good performance.
* Backup/Restore
  + Azure Storage Tables may be backed up and restored seperately from a SharePoint infrastructure.
  + Azure Storage Tables are not affected when a SharePoint app is uninstalled unless the SharePoint App has code added to it which explicitly interacts with Azure Storage Tables it accesses.

1. To see the CSR Rating, click the **My CSR Info** link in the left menu.
2. The MVC controller calls the SurveyRatingsService.cs class which uses the Azure Table Storage API to retrieve the information from the Azure Table Storage.



## Code

This page is an MCV view defined in the CSRInfo\Home.cshmtl file. The CSRInfoController class includes the Home method which is decorated with the SharePointContextFilter attribute. This attribute provides the SharePoint Context to the method when it is invoked. The SharePoint Context is used to retrieve the current user’s NameId. The current user’s NameId is passed to the GetUserScore method in SurveyRatingsService.cs to return the current user’s ratings to the MVC view.

Note: If the current user’s NameId is not present in the Azure Storage Table the code adds information for the user to support the sample. This occurs in the AddSurveyRatings method.

### CSRINFOController.cs

[SharePointContextFilter]

public ActionResult Home()

{

var context =

SharePointContextProvider.Current.GetSharePointContext(HttpContext);

var sharePointService = new SharePointService(context);

var currentUser = sharePointService.GetCurrentUser();

ViewBag.UserName = currentUser.Title;

var surveyRatingsService = new SurveyRatingsService();

ViewBag.Score = surveyRatingsService.GetUserScore(currentUser.UserId.NameId);

return View();

}

### SurveyRatingsService.cs

public SurveyRatingsService(string storageConnectionStringConfigName =

"StorageConnectionString")

{

var connectionString = Util.GetConfigSetting("StorageConnectionString");

var storageAccount = CloudStorageAccount.Parse(connectionString);

this.tableClient = storageAccount.CreateCloudTableClient();

this.surveyRatingsTable = this.tableClient.GetTableReference("SurveyRatings");

this.surveyRatingsTable.CreateIfNotExists();

}

public float GetUserScore(string userName)

{

var query = new TableQuery<Models.Customer>()

.Select(new List<string> { "Score" })

.Where(TableQuery.GenerateFilterCondition("Name",

QueryComparisons.Equal, userName));

var items = surveyRatingsTable

.ExecuteQuery(query)

.ToArray();

if (items.Length == 0)

return AddSurveyRatings(userName);

return (float)items.Average(c => c.Score);

}

private float AddSurveyRatings(string userName)

{

float sum = 0;

int count = 4;

var random = new Random();

for (int i = 0; i < count; i++)

{

var score = random.Next(80, 100);

var customer = new Models.Customer(Guid.NewGuid(), userName, score);

var insertOperation = TableOperation.Insert(customer);

surveyRatingsTable.Execute(insertOperation);

sum += score;

}

return sum / count;

}

# Notes Scenario

The Notes list scenario is engineered to reflect how lists perform in a SharePoint App Web. The Notes list is created in the App Web with a Title and Description field. Using the SharePoint REST API, the Notes list is queried and returns all the notes based on a Customer ID.

Using lists in the App Web has some advantages over other storage solutions.

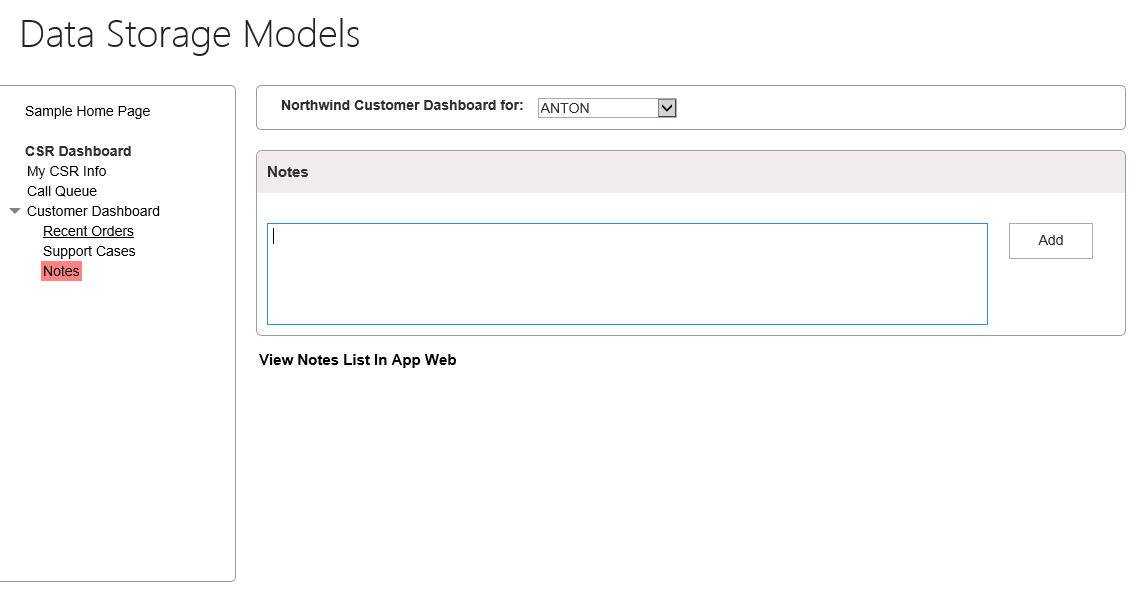
* Data can be queried with simple object model calls like the SharePoint REST API.

However, there are disadvantages as well.

* Design
  + Making an update to a SharePoint list in the App Web requires making an update to the SharePoint app.
  + Data is not searchable with the SharePoint Search Service.
  + Data and query limits exist on lists in an App Web.
    - These limits may make this choice of a data storage model an option which does not fit every business scenario. You should carefully consider how much data you need to store and query before choosing the proper data storage model.
    - This sample illustrates and provides more details about this concept. It is described in subsequent sections in this document.
* Performance
  + Databases typically offer better performance when executing queries that involve many joins and other operations such as calculations when compared to SharePoint lists.
* Backup/Restore
  + Backing up and restoring data in a SharePoint list in an App Web is not as straightforward as in a database.
  + Data in a SharePoint list in an App Web is deleted when a SharePoint app is uninstalled unless the SharePoint App has code added to it which explicitly backs up the data when the app is uninstalled.

## Notes list object model calls

1. To access the Notes for a customer, expand the Customer Dashboard by clicking the arrow and select **Notes**.



1. Select a **customer** in the drop down list to access notes for the customer. Initially, none exist.
2. Enter some text in the Notes text area and click **Add** to save the note to the Notes list in the App Web.
3. Click the **View Notes List in App Web** link to see the out of the box view of the Notes list. This is helpful for comparing the data in the Notes page with the data in the Notes list in the App Web.

## Code

This page is an MCV view defined in the CustomerDashboard\Notes.cshmtl file. The calls to the SharePoint REST API are written in JavaScript and are located in the Scripts/CustomerDashboard.js file. The functions the Notes scenario uses require the SP.RequestExecutor function to execute the cross domain request.

The getNotesAndShow function returns all the notes for a customer.

function getNotesAndShow() {

var executor = new SP.RequestExecutor(appWebUrl);

executor.executeAsync(

{

url: appWebUrl + "/\_api/web/lists/getByTitle('Notes')/items/" +

"?$select=FTCAM\_Description,Modified,Title,Author/ID,Author/Title" +

"&$expand=Author/ID,Author/Title" +

"&$filter=(Title eq '" + customerID + "')",

type: "GET",

dataType: 'json',

headers: { "accept": "application/json;odata=verbose" },

success: function (data) {

var value = JSON.parse(data.body);

showNotes(value.d.results);

},

error: function (error) { console.log(JSON.stringify(error)) }

}

);

}

The addNoteToList function creates a list item in the Notes list.

function addNoteToList(note, customerID) {

var executor = new SP.RequestExecutor(appWebUrl);

var bodyProps = {

'\_\_metadata': { 'type': 'SP.Data.NotesListItem' },

'Title': customerID,

'FTCAM\_Description': note

};

executor.executeAsync({

url: appWebUrl + "/\_api/SP.AppContextSite(@target)/web/lists/getbytitle('Notes')/items?@target='" + appWebUrl + "'",

contentType: "application/json;odata=verbose",

method: "POST",

headers: {

"accept": "application/json;odata=verbose",

"content-type": "application/json;odata=verbose",

"X-RequestDigest": $("#\_\_REQUESTDIGEST").val()

},

body: JSON.stringify(bodyProps),

success: getNotesAndShow,

error: addNoteFailed

});

}

## LIST QUERY THRESHOLDS

To demonstrate the data limits associated with list storage in the App Web you can load enough data to exceed the list query threshold limit. To do this, follow these steps.

1. In the left menu, click **Sample Home Page**.
2. In the **List Query Thresholds** section, click the **Add list items to the Notes list in the App Web** button.



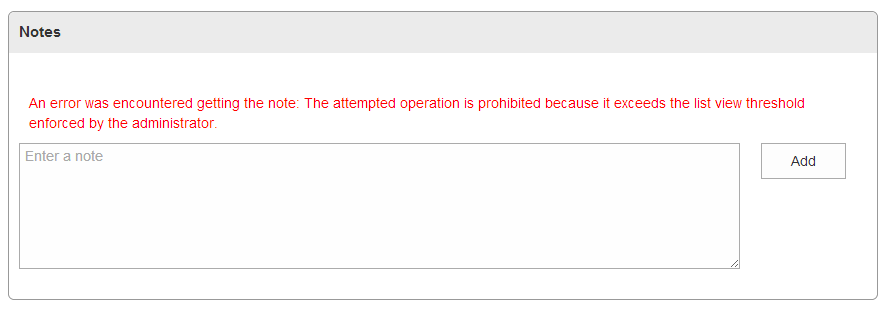
1. See the instructions above the button which describe why and how you will need to perform this operation 10 times.
2. Once the Notes list is updated you will see a message at the top of the page indicating how many list items (Notes) you added and how many are left to add.



Note: This operation typically takes 1 minute to execute each time you click the button. If you do not have time (10 minutes) to wait for this operation to complete 10 times then refer to the screenshot below to see the end result.

When the list has 5000 or more items in it the following status message is displayed when the operation completes.



1. After 5001 items are added to the list, click the Notes link in the left menu. When the page loads you will see the following error. This error message comes directly from the SharePoint REST API.
2. Click the View Notes List in App Web link, page through the list to see the Notes list has 5000 rows (or more if you added list items via the Add button on the Notes page). Although the SharePoint List Views can accommodate browsing this much data, the REST API fails due to the list query throttle threshold.

## DATA STORAGE LIMITS

To demonstrate the data limits associated with list storage you can load enough data to exceed the data storage limit. To do this, follow these steps.

1. In the left menu, click **Sample Home Page**.
2. In the **Data Threshold** section, click the **Fill the App Web Notes list with 1GB of data** button.

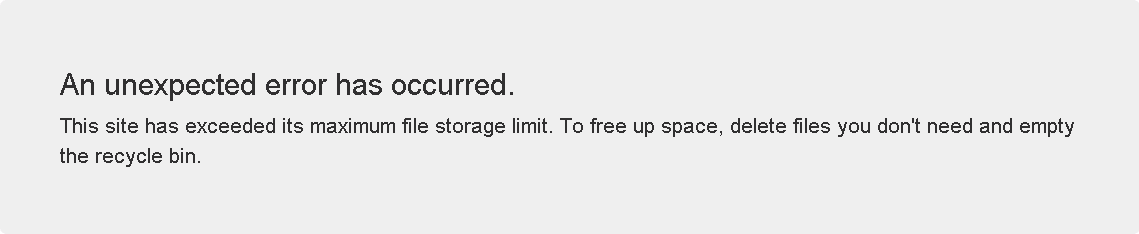


1. See the instructions above the button which describe why and how you will need to perform this operation 11 times.
2. Once the Notes list is updated you will see a message at the top of the page indicating how many list items (Notes) you added and how many are left to add.



Note: This operation typically takes 1 minute to execute each time you click the button. If you do not have time (11 minutes) to wait for this operation to complete 11 times then refer to the screenshot below to see the end result.

1. Eventually, you will receieve the following error message when you click the button.



Note: Recall when you created the site collection, you gave the site collection 1300 MB of storage space. Once the 1300 MB of storage space is exceeded the data threshold limit is enforced.

1. After the data threshold has been exceeded, click the **back button** in the web browser, then click the **Notes link** in the left menu.
2. Click the **View Notes List In App Web** link.
3. When the page loads you will see the following error at the top of the page. 

# Support Cases Scenario

The Support Cases scenario displays support cases for a customer. The data is stored in a SharePoint list in the Host Web and utilizes two different patterns to access and interact with the data. The first pattern includes the SharePoint Search Service and the Content By Search Web Part with a custom Display Template applied. The second pattern includes an App Part (Client Web Part) that displays an MVC view which uses the SP.RequestExecutor to call the SharePoint REST API. Both patterns are illustrated here to demonstrate how data stored in a SharePoint list in the Host Web is accessible via these two commonly used patterns.

The advantages of using this method of data storage and delivery include:

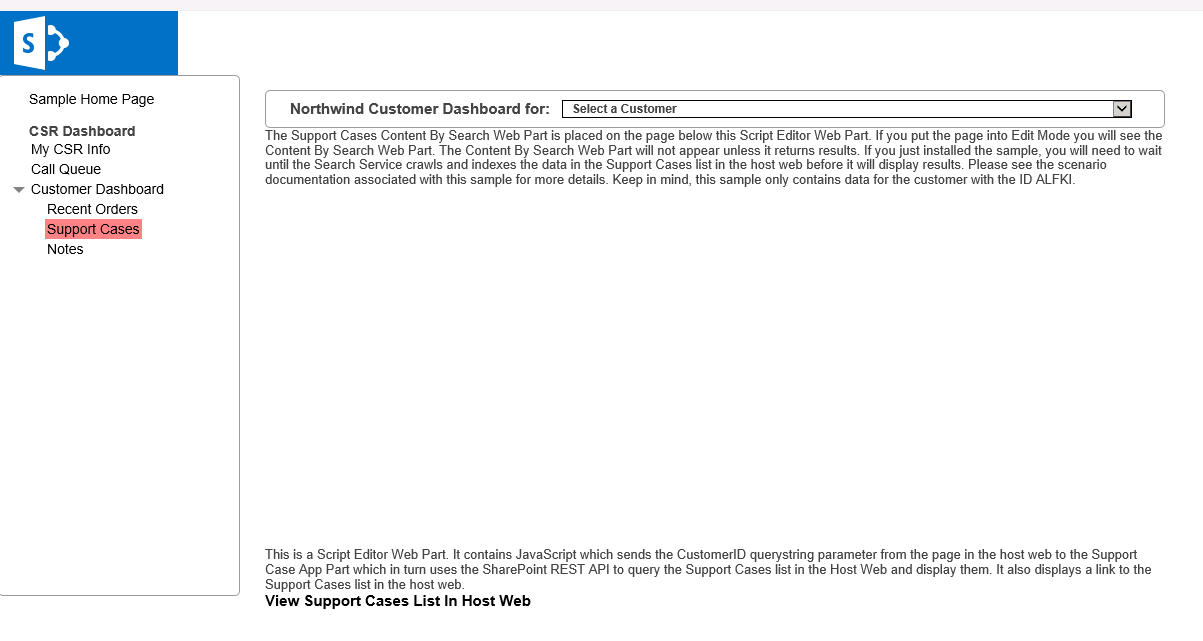
Using lists in the Host Web has some advantages over other storage solutions.

* Design
  + Data can be queried with simple object model calls like the SharePoint REST API.
  + Data is searchable with the SharePoint Search Service.
  + Making an update to a SharePoint list in the Host Web does not require making an update to the SharePoint app as long as the changes do not affect the SharePoint app.
    - For example:
      * Adding a view to a list in the host web will not break a SharePoint app that uses the list.
* Backup/Restore
  + Data in a SharePoint list in a Host Web is not deleted when a SharePoint app is uninstalled unless the SharePoint App has code added to it which explicitly deletes the data when the app is uninstalled.

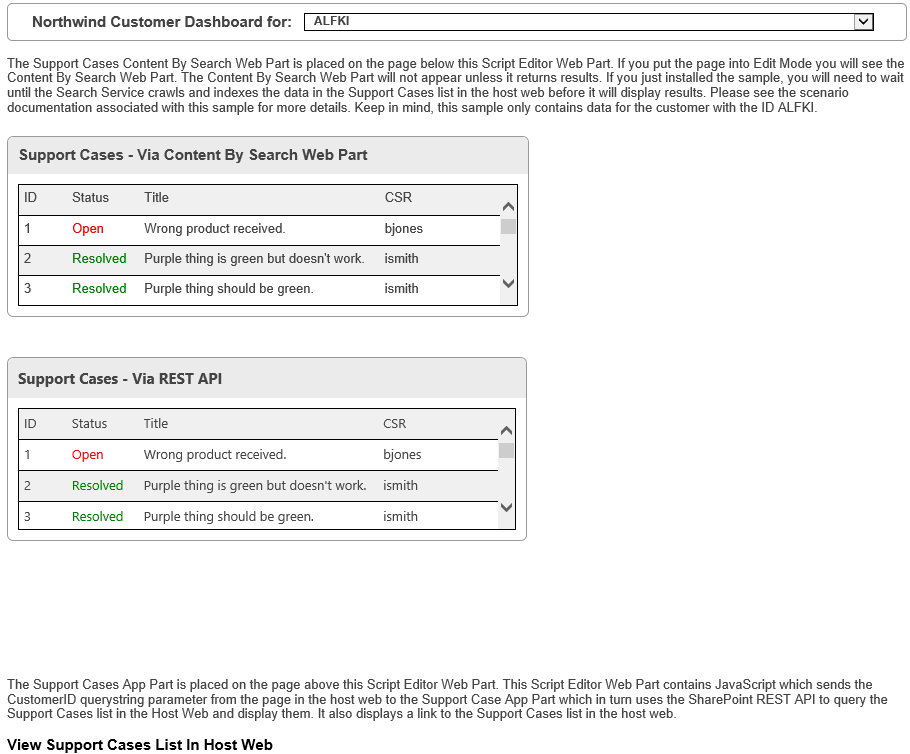
However, there are disadvantages as well.

* Design
  + Data and query limits exist on lists in the Host Web.
    - These limits may make this choice of a data storage model an option which does not fit every business scenario. You should carefully consider how much data you need to store and query before choosing the proper data storage model.
    - This sample illustrates and provides more details about this concept. It is described in subsequent sections in this document.
* Performance
  + Databases typically offer better performance when executing queries that involve many joins and other operations such as calculations when compared to SharePoint lists.
* Backup/Restore
  + Backing up and restoring data in a SharePoint list in an Host Web is not as straightforward as in a database.

1. To see the Support Cases for a customer, click the **Support Cases** link in the left menu.



1. Select a customer in the drop down menu to see the Support Cases displayed in a Content By Search Web Part and an App Part.



**IMPORTANT NOTE:** The sample only contains data for the customer ALFKI and the Content By Search web part only appears if content is returned. Please see the text on this page for more information and why the data may not appear immediately after you install the sample. Sometimes it may take more than 24 hours for the SharePoint Search Service to index the data on an O365 site.

## Code

This App Part displays an MCV view defined in the SupportCaseAppPart\Index.cshtml file. The MVC view uses the SharePoint REST API to access the Support Cases list in the Host Web and returns the results to the MVC view.

function execCrossDomainRequest() {

var executor = new SP.RequestExecutor(appWebUrl);

executor.executeAsync(

{

url: appWebUrl + "/\_api/SP.AppContextSite(@@target)" +

"/web/lists/getbytitle('Support Cases')/items" +

"?$filter=(FTCAM\_CustomerID eq '" + customerID + "')" +

"&$top=30" +

"&$select=Id,Title,FTCAM\_Status,FTCAM\_CSR" +

"&@@target='" + hostWebUrl + "'",

method: "GET",

headers: { "Accept": "application/json; odata=verbose" },

success: successHandler,

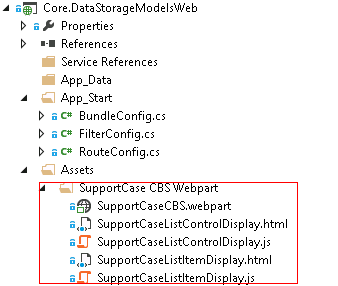
error: errorHandler

}

);

}

The Content By Search Web Part and custom Display Template are included in the solution. They are found in the Assets\SupportCase CBS Webpart folder.



## LIST QUERY THRESHOLDS

To demonstrate the data limits associated with list storage in the Host Web you can load enough data to exceed the list query threshold limit. To do this, follow these steps.

**Important Note**: If you have already filled the Notes list with the file attachements which caused the site data threshold to be exceeded you must delete the Notes list or all the items in the Notes list to proceed.

1. In the left menu, click **Sample Home Page**.
2. In the **List Query Thresholds** section, click the **Add list items to the Support Cases list in the Host Web** button.



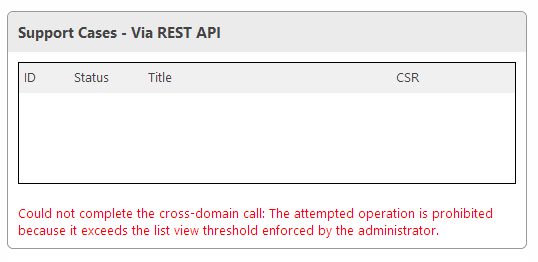
1. See the instructions above the button which describe why and how you will need to perform this operation 10 times.
2. Once the Support Cases list is updated you will see a message at the top of the page indicating how many list items (Support Cases) you added and how many are left to add.

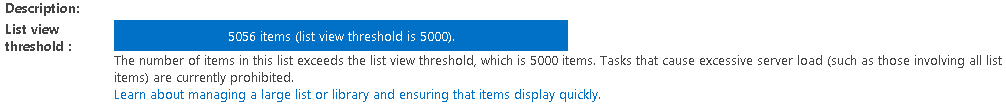


Note: This operation typically takes 1 minute to execute each time you click the button. If you do not have time (10 minutes) to wait for this operation to complete 10 times then refer to the screenshot below to see the end result.

When the list has 5000 or more items in it the following status message is displayed when the operation completes.



1. After 5001 items are added to the list, click the Support Cases link in the left menu. When the page loads you will see the following error. This error message comes directly from the SharePoint REST API. 
2. Click the View Support Cases List in Host Web link, page through the list to see the Support Cases list has 5000 rows. Although the SharePoint List Views can accommodate browsing this much data, the REST API fails due to the list query throttle threshold.
3. Select the View Support Cases List in Host Web button, the Support Cases list has 5000 rows, click list settings, you will see the list has execeed thrshold.



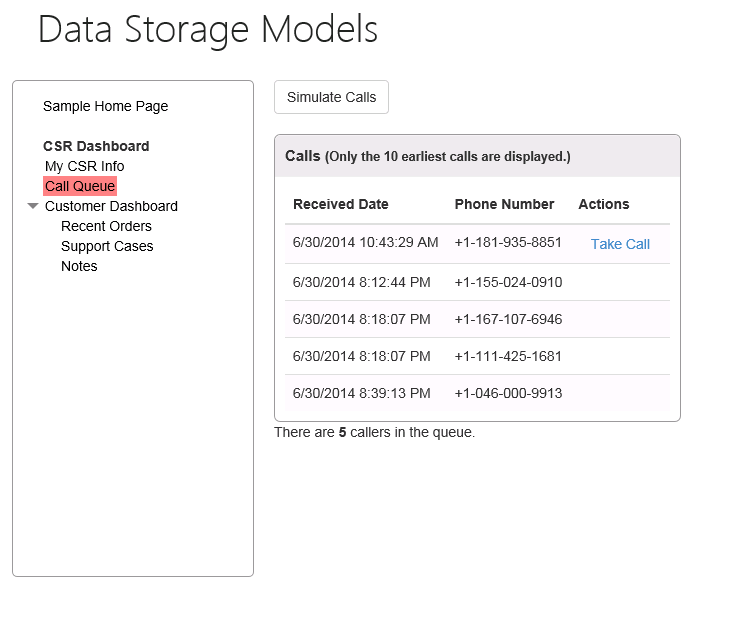
# Call Queue Scenario

The Call Queue scenario lists callers in the support queue and simulates taking calls. The Call Queue scenario utilizes Azure Storage Queues for data storage and the Microsoft.WindowsAzure.Storage.Queue.CloudQueue API with MVC.

The advantages of using this method of data storage and delivery include:

* Design
  + Azure Storage Queues may be used by more than one SharePoint app.
  + Azure Storage Queues may be updated independently of SharePoint. This allows developers to update data stores without redeploying the SharePoint app.
* Performance
  + Azure Storage Queue performance is not affected by SharePoint and scales easily to ensure good performance.
* Backup/Restore
  + Azure Storage Queues may be backed up and restored seperately from a SharePoint infrastructure.
  + Azure Storage Queues are not affected when a SharePoint app is uninstalled unless the SharePoint App has code added to it which explicitly interacts with Azure Storage Queues it accesses.

1. To see the call queue, click **Call Queue** in the left menu.
2. To simulate calls being added to the call queue, click the **Simulate Calls** button.
3. To simulate taking the first call in the call queue, click the **Take Call** link.



This page is an MCV view defined in the CallQueue\Home.cshmtl file. The CallQueueController includes the methods which call the CallQueueService.cs class to interact with the Azure Storage Queue. These methods return, add, and delete items in the Azure Storage Queue.

## CallQueueController.cs

public class CallQueueController : Controller

{

public CallQueueService CallQueueService { get; private set; }

public CallQueueController()

{

CallQueueService = new CallQueueService();

}

// GET: CallQueue

public ActionResult Home(UInt16 displayCount = 10)

{

var calls = CallQueueService.PeekCalls(displayCount);

ViewBag.DisplayCount = displayCount;

ViewBag.TotalCallCount = CallQueueService.GetCallCount();

return View(calls);

}

[HttpPost]

public ActionResult SimulateCalls(string spHostUrl)

{

int count = CallQueueService.SimulateCalls();

TempData["Message"] = string.Format("Successfully simulated {0} calls and added them to the call queue.", count);

return RedirectToAction("Index", new { SPHostUrl = spHostUrl });

}

[HttpPost]

public ActionResult TakeCall(string spHostUrl)

{

CallQueueService.DequeueCall();

TempData["Message"] = "Call taken successfully and removed from the call queue!";

return RedirectToAction("Index", new { SPHostUrl = spHostUrl });

}

}

## CallQueueService.cs

Each method uses the CallQueueService.cs to call the Azure Storage Queue API.

public class CallQueueService

{

private CloudQueueClient queueClient;

private CloudQueue queue;

public CallQueueService(string storageConnectionStringConfigName = "StorageConnectionString")

{

var connectionString = CloudConfigurationManager.GetSetting(storageConnectionStringConfigName);

var storageAccount = CloudStorageAccount.Parse(connectionString);

this.queueClient = storageAccount.CreateCloudQueueClient();

this.queue = queueClient.GetQueueReference("calls");

this.queue.CreateIfNotExists();

}

public int? GetCallCount()

{

queue.FetchAttributes();

return queue.ApproximateMessageCount;

}

public IEnumerable<Call> PeekCalls(UInt16 count)

{

var messages = queue.PeekMessages(count);

var serializer = new JavaScriptSerializer();

foreach (var message in messages)

{

Call call = null;

try

{

call = serializer.Deserialize<Call>(message.AsString);

}

catch { }

if (call != null) yield return call;

}

}

public void AddCall(Call call)

{

var serializer = new JavaScriptSerializer();

var content = serializer.Serialize(call);

var message = new CloudQueueMessage(content);

queue.AddMessage(message);

}

public void DequeueCall()

{

var message = queue.GetMessage();

queue.DeleteMessage(message);

}

public int SimulateCalls()

{

Random random = new Random();

int count = random.Next(1, 6);

for (int i = 0; i < count; i++)

{

int phoneNumber = random.Next();

var call = new Call

{

ReceivedDate = DateTime.Now,

PhoneNumber = phoneNumber.ToString("+1-000-000-0000")

};

AddCall(call);

return count;

}

}