# Module 8: Designing a Communication Strategy by Using Queues and Service Bus

# Lab: Using Queues and Service Bus to Manage Communication Between Web Applications in Azure

## Exercise 1: Creating an Azure Service Bus Namespace

#### Task 1: Sign in to the Azure Portal

1. On the Start screen, click the **Internet Explorer** tile.
2. Go to *(*[*https://portal.azure.com*](https://portal.azure.com)*)*.
3. In the email address box, type the email address of your Microsoft account.
4. Click **Next**.
5. In the password box, type the password for your Microsoft account.
6. Click **Sign In**.

* **Note**: If this is your first time logging in to the Portal, you may be prompted with a “Welcome” dialog. You can safely close this dialog and continue.

#### Task 2: Create Service Bus Namespace

1. In the navigation pane on the left side of the Azure Portal, click **All services**.
2. In the **All services** blade that displays, click **Service Bus**.
3. In the **Service Bus** blade that displays, view your list of Service Bus namespaces.
4. At the top of the **Service Bus** blade, click the **Add** button.
5. In the **Create namespace** blade that displays, perform the following steps:
   1. In the **Name** field, provide the value **bus20532[your name in lowercase here]**.
   2. Click the **Pricing Tier** link.
   3. In the **Choose your pricing tier** blade, select the **Basic** option.
   4. Click the **Select** button.
   5. Back in the **Create namespace** blade, locate the **Resource group** section. In the **Resource group** section, select the **Create new** option.
   6. In the **Resource group** section, locate the dialog box and enter the value **MOD08QBUS**.
   7. In the **Location** list, select the region that is closest to your location.
   8. Click the **Create** button.

* **Note**: Wait for Azure to finish creating the Service Bus namespace prior to moving forward with the lab. You will receive a notification when the *Service Bus Namespace* is created.

1. In the navigation pane on the left side of the Azure Portal, click **All services**.
2. In the **All services** blade that displays, click **Service Bus**.
3. In the **Service Bus** blade that displays, select the Service Bus namespace instance that has a prefix of **bus20532**.
4. In the *Service Bus Namespace* blade, locate the **Settings** section on the left-side of the blade and click the **Shared access policies** link.
5. In the **Shared access policies** pane, click the **Add** button at the top of the pane.
6. In the **Add SAS Policy** popup that appears, perform the following actions:
   1. In the **Policy name** field, enter the value **ReadWriteOnly**.
   2. Ensure the checkbox for the **Send** option is selected.
   3. Ensure the checkbox for the **Listen** option is selected.
   4. Click the **Create** button to create the new policy.

* **Note**: Wait for Azure to finish creating the policy prior to moving forward with the lab. You will receive a notification when the *policy* is created.

1. Back in the **Shared access policies** pane, click the newly created **ReadWriteOnly** policy.
2. In the **SAS Policy** popup that appears, record the value of the **Primary Connection String** field. You will use this value later in this lab.
3. Close the **SAS Policy** popup.
4. In the *Service Bus Namespace* blade, locate the **Entities** section on the left-side of the blade and click the **Queues** link.
5. In the **Queues** pane, click the **+ Queue** button at the top of the pane.
6. In the **Create queue** popup that appears, perform the following actions:
   1. In the **Name** field, enter the value *documentqueue*.
   2. Click the **Create** button to create the new policy.

* **Note**: Wait for Azure to finish creating the queue prior to moving forward with the lab. You will receive a notification when the *queue* is created.

#### Task 3: Create Other Azure Assets

1. In the navigation pane on the left side of the Azure Portal, click **All services**.
2. In the **All services** blade that displays, click **Storage Accounts**.
3. In the **Storage accounts** blade that displays, view your list of Storage instances.
4. At the top of the **Storage accounts** blade, click the **Add** button.
5. In the **Create storage account** blade that displays, perform the following steps:
   1. In the **Name** field, provide the value **star20532[your name in lowercase here]**.
   2. In the **Deployment model** section, ensure that the *Resource manager* option is selected.
   3. In the **Account kind** list, ensure that the *StorageV2 (general purpose v2)* option is selected.
   4. In the **Location** list, select the region closest to your current location.
   5. Click on the **Replication** list and select the **Locally-redundant storage (LRS)** option.
   6. In the **Performance** section, ensure that the *Standard* option is selected.
   7. In the **Access tier** section, ensure that the *Hot* option is selected.
   8. In the **Secure transfer required** section, ensure that the *Enabled* option is selected.
   9. In the **Resource group** section, select the **Use Existing** option.
   10. In the **Resource group** section, locate the dialog box and enter the value **MOD08QBUS**.
   11. In the **Virtual networks** section, ensure that the *Disabled* option is selected.
   12. Click the **Create** button.

**Note**: Wait for Azure to finish creating the storage account prior to moving forward with the lab. You will receive a notification when the *Storage Account* is created.

1. In the navigation pane on the left side of the Azure Portal, click **All services**.
2. In the **All services** blade that displays, click **Storage Accounts**.
3. In the **Storage accounts** blade that displays, select the Storage account instance that has a prefix of **star20532**.
4. In the *Storage account* blade, locate the **Settings** section on the left-side of the blade and click the **Access keys** link.
5. In the **Keys** pane, select any one of the keys and record the value in the **Connection string** field. You will use this value later in this lab.
6. In the navigation pane on the left side, click **All services**.
7. In the **All services** blade that displays, click **SQL databases**.
8. In the **SQL databases** blade that displays, view your list of SQL databases.
9. At the top of the **SQL databases** blade, click the **Add** button.
10. In the *SQL database* blade that displays, locate the **Database name** field and provide the value **db20532**.
11. Locate the **Resource group** section, select the **Use existing** option and then select the option **MOD08QBUS** from the list.
12. In the **Select source** list, select the value **Blank database**.
13. Click the **Server** option.
14. In the **Server** blade that displays, perform the following actions:
    1. Click **Create a new server**.
    2. In the **New Server** blade that displays, locate the **Server Name** field.
    3. In the **Server name** field, type **sv20532[*Your Name Here*]**.
    4. In the **Server admin login** field, type **testuser** as login.
    5. In the **Password** field, type **TestPa$$w0rd** as password.
    6. In the **Confirm password** field, type **TestPa$$w0rd**.
    7. In the **Location** list, select the region that is closest to your location.
    8. Click the **Select** button.
15. Back in the **SQL database** blade, click the **Pricing Tier** option.
16. In the **Resource Configuration & Pricing** blade that displays, select the **Basic** option.
17. Click **Apply** to close the blade.
18. Back in the **SQL database** blade, click the **Create** button to create the SQL database and server.

**Note**: Wait for Azure to finish creating the SQL Database instance prior to moving forward with the lab. You will receive a notification when the *SQL Database* is created.

1. In the navigation pane on the left side of the Azure Portal, click **All services**.
2. In the **All services** blade that displays, click **SQL Database**.
3. In the **SQL Database** blade that displays, select the SQL database instance named **db20532**.
4. In the *SQL database* blade, locate the **Settings** section on the left-side of the blade and click the **Connection strings** link.
5. In the **Connection strings** pane, copy the value of the **ADO.NET** connection string. Be sure to replace the placeholder values for {your\_username} and {your\_password} with the values **testuser** and **TestPa$$w0rd** respectively.

* **Note**: For example, if your copied connection string is Server=tcp:sv20532microsoft.database.windows.net,1433;Initial Catalog=db20532;Persist Security Info=False;User ID={your\_username};Password={your\_password};MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;Connection Timeout=30;, your updated connection string would be Server=tcp:sv20532microsoft.database.windows.net,1433;Initial Catalog=db20532;Persist Security Info=False;User ID=testuser;Password=TestPa$$w0rd;MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;Connection Timeout=30;

1. In the navigation pane on the left side of the Azure Portal, click **All services**.
2. In the **All services** blade that displays, click **SQL servers**.
3. In the **SQL servers** blade that displays, select the SQL database instance that has a prefix of **sv20532**.
4. In the *SQL server* blade, locate the **Security** section on the left-side of the blade and click the **Firewalls and virtual networks** link.
5. In the **Firewalls and virtual networks** pane, click the **Add client IP** button to add your virtual machine's IP Address to the list of allowed IP Address ranges.
6. Click on **Save** at the top of the blade. Once saved, close the confirmation dialog by clicking the **Ok** button.

* **Note**: It might take couple of minutes for the firewall changes to get updated on server.

1. In the navigation pane on the left side of the Azure Portal, click **All services**.
2. In the **All services** blade that displays, click **Azure Cosmos DB**.
3. In the **Azure Cosmos DB** blade that displays, view your list of Storage instances.
4. At the top of the **Azure Cosmos DB** blade, click the **Add** button.
5. In the **New account** blade that displays, perform the following steps:
   1. In the **ID** field, provide the value **nosql20532[your name in lowercase here]**.
   2. In the **API** list, select the **SQL** option.
   3. In the **Resource group** section, select the **Use existing** option.
   4. In the **Resource group** section, select the value **MOD08QBUS** from the list.
   5. In the **Location** list, select the region closest to your current location.
   6. Ensure that the *Enable geo-redundancy* checkbox is un-selected.
   7. In the **Virtual networks** section, ensure that the *Disabled* option is selected.
   8. Click the **Create** button.

**Note**: Wait for Azure to finish creating the Azure Cosmos DB account prior to moving forward with the lab. You will receive a notification when the *Azure Cosmos DB account* is created.

1. In the navigation pane on the left side of the Azure Portal, click **All services**.
2. In the **All services** blade that displays, click **Azure Cosmos DB**.
3. In the **Azure Cosmos DB** blade that displays, select the Azure Cosmos DB account instance that has a prefix of **nosql20532**.
4. In the *Azure Cosmos DB account* blade, locate the **Settings** section on the left-side of the blade and click the **Keys** link.
5. In the **Keys** pane, record the values in the **URI** and **PRIMARY KEY** fields. You will use these values later in this lab.

#### Task 4: Update Settings in Contoso.Events Projects

1. On the Start screen, click the **Desktop** tile.
2. On the taskbar, click the **File Explorer** icon.
3. In the *This PC* window, go to **Allfiles (F):\Mod08\Labfiles\Starter**, and then double-click **Contoso.Events.sln**.
4. On the **Build** menu, click **Build Solution**.
5. In the **Solution Explorer** pane of the Contoso.Events - Microsoft Visual Studio window, expand the **Contoso.Events.Management** project.
6. In the **Solution Explorer** pane, double-click **appsettings.json**.
7. In the JSON object, in line 21, locate the **ConnectionStrings.EventsContextConnectionString** path. Observe that the current value is empty:

* "ConnectionStrings": {  
   "EventsContextConnectionString": ""  
  },

1. Update the value of the **EventsContextConnectionString** property by setting it's value to the **Connection String** of the **SQL Database** you recorded earlier in this lab.
2. In the JSON object, in line 17, locate the **CosmosSettings.EndpointUrl** path. Observe that the current value is empty:

* "CosmosSettings": {  
   "DatabaseId": "EventsDatabase",  
   "CollectionId": "RegistrationCollection",  
   "EndpointUrl": "",  
   "AuthorizationKey": ""  
  },

1. Update the value of the **EndpointUrl** property by setting it's value to the **Endpoint Uri** of the **Azure Cosmos DB account** you recorded earlier in this lab.
2. In the JSON object, in line 18, locate the **CosmosSettings.AuthorizationKey** path. Observe that the current value is empty:

* "CosmosSettings": {  
   "DatabaseId": "EventsDatabase",  
   "CollectionId": "RegistrationCollection",  
   "EndpointUrl": "",  
   "AuthorizationKey": ""  
  },

1. Update the value of the **AuthorizationKey** property by setting it's value to the **Key** of the **Azure Cosmos DB account** you recorded earlier in this lab.
2. In the JSON object, in line 6, locate the **StorageSettings.ConnectionString** path. Observe that the current value is empty:

* "StorageSettings": {  
   "ConnectionString": "",  
   "ContainerName": "signinsheets",  
   "QueueName": "signinqueue"  
  },

1. Update the value of the **ConnectionString** property by setting it's value to the **Connection String** of the **Storage account** you recorded earlier in this lab.
2. In the JSON object, in line 11, locate the **ServiceBusSettings.ConnectionString** path. Observe that the current value is empty:

* "ServiceBusSettings": {  
   "ConnectionString": "",  
   "QueueName": "documentqueue"  
  },

1. Update the value of the **ConnectionString** property by setting it's value to the **Connection String** of the **Service Bus namespace policy** you recorded earlier in this lab.
2. Save the **appsettings.json** file.
3. In the **Solution Explorer** pane of the Contoso.Events - Microsoft Visual Studio window, expand the **Contoso.Events.Worker** project.
4. In the **Solution Explorer** pane, double-click **local.settings.json**.
5. In the JSON object, in line 4, locate the **AzureWebJobsStorage** path. Observe that the current value is empty:

* "AzureWebJobsStorage": "",

1. Update the value of the **AzureWebJobsStorage** property by setting it's value to the **Connection String** of the **Storage account** you recorded earlier in this lab.
2. In the JSON object, in line 5, locate the **AzureWebJobsDashboard** path. Observe that the current value is empty:

* "AzureWebJobsDashboard": "",

1. Update the value of the **AzureWebJobsDashboard** property by setting it's value to the **Connection String** of the **Storage account** you recorded earlier in this lab.
2. In the JSON object, in line 6, locate the **EventsContextConnectionString** path. Observe that the current value is empty:

* "EventsContextConnectionString": "",

1. Update the value of the **EventsContextConnectionString** property by setting it's value to the **Connection String** of the **SQL Database** you recorded earlier in this lab.
2. In the JSON object, in line 7, locate the **ServiceBusConnectionString** path. Observe that the current value is empty:

* "ServiceBusConnectionString": "",

1. Update the value of the **ServiceBusConnectionString** property by setting it's value to the **Connection String** of the **Service Bus namespace policy** you recorded earlier in this lab.
2. In the JSON object, in line 8, locate the **CosmosEndpointUrl** path. Observe that the current value is empty:

* "CosmosEndpointUrl": "",

1. Update the value of the **CosmosEndpointUrl** property by setting it's value to the **Endpoint Uri** of the **Azure Cosmos DB account** you recorded earlier in this lab.
2. In the JSON object, in line 9, locate the **CosmosAuthorizationKey** path. Observe that the current value is empty:

* "CosmosAuthorizationKey": "",

1. Update the value of the **CosmosAuthorizationKey** property by setting it's value to the **Key** of the **Azure Cosmos DB account** you recorded earlier in this lab.
2. Save the **local.settings.json** file.

## Exercise 2: Using Azure Storage Queues for Document Generation

#### Task 1: Implement QueueContext Class using the IQueueContext Interface

1. In the **Solution Explorer** pane of the Contoso.Events - Microsoft Visual Studio window, expand the **Contoso.Events.Data** project and double-click the **QueueContext.cs** file.
2. In the code editor tab for the **QueueContext.cs** file, locate the **QueueContext** class:

* public class QueueContext : IQueueContext

1. Within the **QueueContext** class, locate the **SendQueueMessageAsync** method and delete any existing code within the method:

* public async Task SendQueueMessageAsync(string eventKey)  
  {  
    
  }

1. Within the **SendQueueMessageAsync** method, add a new line of code to create a new **CloudStorageAccount** class instance:

* CloudStorageAccount account = CloudStorageAccount.Parse(StorageSettings.ConnectionString);

1. Add a new line of code to create a new instance of the **CloudQueueClient** class using the **CreateCloudQueueClient** method of the **CloudStorageAccount** class:

* CloudQueueClient queueClient = account.CreateCloudQueueClient();

1. Add a new line of code to get a reference to a new or existing queue using the **GetQueueReference** method of the **CloudQueue** class:

* CloudQueue queue = queueClient.GetQueueReference(StorageSettings.QueueName);

1. Add a new line of code to invoke the **CreateIfNotExistsAsync** method of the **CloudQueue** class that will create the queue if it does not already exist:

* await queue.CreateIfNotExistsAsync();

1. Add a new line of code to create a new **CloduQueueMessage** instance using the value of the **eventKey** parameter:

* CloudQueueMessage message = new CloudQueueMessage(eventKey);

1. Add a new line of code to invoke the **AddMessageAsync** method of the **CloudQueue** class to send the message to the queue.

* await queue.AddMessageAsync(message);

1. Save the **QueueContext.cs** file.

#### Task 2: Register QueueContext Implementation with ASP.NET MVC Dependency Injection

1. In the **Solution Explorer** pane of the Contoso.Events - Microsoft Visual Studio window, expand the **Contoso.Events.Management** project and double-click the **Startup.cs** file.
2. In the code editor tab for the **Startup.cs** file, locate the **Startup** class:

* public class Startup

1. Within the **Startup** class, locate the **ConfigureServices** method:

* public void ConfigureServices(IServiceCollection services)

1. Within the **ConfigureServices** method, add a new line of code at line 35 that registers the **QueueContext** implementation of the **IQueueContext** interface:

* services.AddTransient<IQueueContext, QueueContext>();

1. Save the **Startup.cs** file.

#### Task 3: Implement Storage Queue Trigger for Azure Function

1. In the **Solution Explorer** pane of the Contoso.Events - Microsoft Visual Studio window, expand the **Contoso.Events.Worker** project and double-click the **ProcessDocuments.cs** file.
2. In the code editor tab for the **ProcessDocuments.cs** file, locate the **ProcessDocuments** class:

* public static class ProcessDocuments

1. Within the **ProcessDocuments** class, locate the **Run** method:

* public static async Task Run(string message, [Blob("signinsheets/output.docx", FileAccess.Write)] Stream blob, TraceWriter log)

1. Update the method signature of the **Run** method by adding a **QueueTrigger** parameter attribute to the **message** parameter specifying to use the **signinqueue** queue and the **AzureWebJobsStorage** connection string:

* public static async Task Run([QueueTrigger("signinqueue", Connection = "AzureWebJobsStorage")]string message, [Blob("signinsheets/output.docx", FileAccess.Write)] Stream blob, TraceWriter log)

1. Update the method signature of the **Run** method by modifying the existing **Blob** parameter attribute to use the value from the queue message as the filename of the blob:

* public static async Task Run([QueueTrigger("signinqueue", Connection = "AzureWebJobsStorage")]string message, [Blob("signinsheets/{queueTrigger}.docx", FileAccess.Write)] Stream blob, TraceWriter log)

1. Save the **ProcessDocuments.cs** file.

#### Task 4: Validate Sign-In Sheet Generation

1. In the **Solution Explorer** pane, right-click the **Contoso.Events** solution, and then click **Properties**.
2. Navigate to the **Startup Project** section located under the **Common Properties** header.
3. In the **Startup Project** section, locate and select the **Multiple startup projects** option.
4. Within the **Multiple startup projects** table, perform the following actions:
   1. Locate the **Contoso.Events.Management** entry and change it's *Action* from **None** to **Start**.
   2. Locate the **Contoso.Events.Worker** entry and change it's *Action* from **None** to **Start**.
5. Click the **OK** button to close the *Property* dialog.
6. On the **Debug** menu, click **Start Debugging**.
7. On the home page of the web application, verify that it displays a list of events.
8. Click the **Generate Sign-In Sheet** button for any of the events in the list. Remember the name of the event as you will click the same button again in this lab.
9. You will observe that the Azure Function will immediately detect a message in the **signinqueue** queue and begin processing that message. Once it is done processing, it will create a new document in the **signinsheets** container.

* **Note**: This can take up to a minute to occur.

1. Refresh the browser window that is displaying the website.
2. You should now see a section titled **Sign-In Document Already Exists**. This indicates that the sign-in document was generated successfully.
3. Open the downloaded *docx* file on your local machine. Observe and then close the file.
4. Close the browser window that is displaying the website.
5. Close the console window for the Azure Functions instance.

## Exercise 3: Using Service Bus Queues for Document Generation

#### Task 1: Implement ServiceBusContext Class using the IQueueContext Interface

1. In the **Solution Explorer** pane of the Contoso.Events - Microsoft Visual Studio window, expand the **Contoso.Events.Data** project and double-click the **ServiceBusContext.cs** file.
2. In the code editor tab for the **ServiceBusContext.cs** file, locate the **ServiceBusContext** class:

* public class ServiceBusContext : IQueueContext

1. Within the **ServiceBusContext** class, locate the **SendQueueMessageAsync** method and delete any existing code within the method:

* public async Task SendQueueMessageAsync(string eventKey)  
  {  
    
  }

1. Within the **SendQueueMessageAsync** method, add a new line of code to create a new **QueueClient** class instance:

* QueueClient queueClient = new QueueClient(ServiceBusSettings.ConnectionString, ServiceBusSettings.QueueName);

1. Add a new line of serialize the **eventKey** parameter as a byte array:

* byte[] messageContent = Encoding.UTF8.GetBytes(eventKey);

1. Add a new line of code to create a new instance of the **Message** class and pass in the byte array to the class constructor:

* Message message = new Message(messageContent);

1. Add a new line of code to set the **Label** property of the **Message** class instance to the **eventKey** parameter:

* message.Label = eventKey;

1. Add a new line of code to invoke the **SendAsync** method of the **QueueClient** class to send the message to the queue.

* await queueClient.SendAsync(message);

1. Save the **QueueContext.cs** file.

#### Task 2: Register ServiceBusContext Implementation with ASP.NET MVC Dependency Injection

1. In the **Solution Explorer** pane of the Contoso.Events - Microsoft Visual Studio window, expand the **Contoso.Events.Management** project and double-click the **Startup.cs** file.
2. In the code editor tab for the **Startup.cs** file, locate the **Startup** class:

* public class Startup

1. Within the **Startup** class, locate the **ConfigureServices** method:

* public void ConfigureServices(IServiceCollection services)

1. Within the **ConfigureServices** method, replace the existing line of code at line 35 with a new line of code that registers the **ServiceBusContext** implementation of the **IQueueContext** interface:

* services.AddTransient<IQueueContext, ServiceBusContext>();

1. Save the **Startup.cs** file.

#### Task 3: Implement Service Bus Queue Trigger for Azure Function

1. In the **Solution Explorer** pane of the Contoso.Events - Microsoft Visual Studio window, expand the **Contoso.Events.Worker** project and double-click the **ProcessDocuments.cs** file.
2. In the code editor tab for the **ProcessDocuments.cs** file, locate the **ProcessDocuments** class:

* public static class ProcessDocuments

1. Within the **ProcessDocuments** class, locate the **Run** method:

* public static async Task Run([QueueTrigger("signinqueue", Connection = "AzureWebJobsStorage")]string message, [Blob("signinsheets/{queueTrigger}.docx", FileAccess.Write)] Stream blob, TraceWriter log)

1. Update the method signature of the **Run** method by replacing the **QueueTrigger** parameter attribute with a **ServiceBusTrigger** parameter attribute and specifying to use the **documentqueue** queue and the **ServiceBusConnectionString** connection string:

* public static async Task Run([ServiceBusTrigger("documentqueue", Connection = "ServiceBusConnectionString")]string message, [Blob("signinsheets/{queueTrigger}.docx", FileAccess.Write)] Stream blob, TraceWriter log)

1. Update the method signature of the **Run** method by modifying the existing **Blob** parameter attribute to use the **label** value from the queue message as the filename of the blob:

* public static async Task Run([ServiceBusTrigger("documentqueue", Connection = "ServiceBusConnectionString")]string message, [Blob("signinsheets/{label}.docx", FileAccess.Write)] Stream blob, TraceWriter log)

1. Save the **ProcessDocuments.cs** file.

#### Task 4: Validate Sign-In Sheet Generation

1. In the **Solution Explorer** pane, right-click the **Contoso.Events** solution, and then click **Properties**.
2. Navigate to the **Startup Project** section located under the **Common Properties** header.
3. In the **Startup Project** section, locate and select the **Multiple startup projects** option.
4. Within the **Multiple startup projects** table, perform the following actions:
   1. Locate the **Contoso.Events.Management** entry and change it's *Action* from **None** to **Start**.
   2. Locate the **Contoso.Events.Worker** entry and change it's *Action* from **None** to **Start**.
5. Click the **OK** button to close the *Property* dialog.
6. On the **Debug** menu, click **Start Debugging**.
7. On the home page of the web application, verify that it displays a list of events.
8. Click the **Generate Sign-In Sheet** button for any of the events in the list. Remember the name of the event as you will click the same button again in this lab.
9. You will observe that the Azure Function will immediately detect a message in the **signinqueue** queue and begin processing that message. Once it is done processing, it will create a new document in the **signinsheets** container.

* **Note**: This can take up to a minute to occur.

1. Refresh the browser window that is displaying the website.
2. You should now see a section titled **Sign-In Document Already Exists**. This indicates that the sign-in document was generated successfully.
3. Open the downloaded *docx* file on your local machine. Observe and then close the file.
4. Close the browser window that is displaying the website.
5. Close the console window for the Azure Functions instance.

## Exercise 4: Cleanup Subscription

#### Task 1: Open Cloud Shell

1. At the top of the portal, click the **Cloud Shell** icon to open a new shell instance.
2. In the **Cloud Shell** command prompt at the bottom of the portal, type in the following command and press **Enter** to list all resource groups in the subscription:

* az group list

1. Type in the following command and press **Enter** to view a list of possible CLI commands to *delete a Resource Group*:

* az group delete --help

#### Task 2: Delete Resource Group

1. Type in the following command and press **Enter** to delete the **MOD08QBUS** *Resource Group*:

* az group delete --name MOD08QBUS --no-wait --yes

1. Close the **Cloud Shell** prompt at the bottom of the portal.

#### Task 3: Close Active Applications

1. Close the currently running web browser application.
2. Close the currently running **Visual Studio** application.

**Review**: In this exercise, you "cleaned up your subscription" by removing the **Resource Groups** used in this lab.

©2016 Microsoft Corporation. All rights reserved. The text in this document is available under the [Creative Commons Attribution 3.0 License](https://creativecommons.org/licenses/by/3.0/legalcode), additional terms may apply. All other content contained in this document (including, without limitation, trademarks, logos, images, etc.) are ***not*** included within the Creative Commons license grant. This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.

This document is provided "as-is." Information and views expressed in this document, including URL and other Internet Web site references, may change without notice. You bear the risk of using it. Some examples are for illustration only and are fictitious. No real association is intended or inferred. Microsoft makes no warranties, express or implied, with respect to the information provided here.