Module 11

Controlling Access to ASP.NET MVC 4 Web Applications

Contents:

Module Overview

Authentication is a vital requirement in most web-based applications. Developers usually display only restricted information to all users. Web applications require users to authenticate themselves to view exclusive information. Web applications also display specific information that is relevant to specific user roles. Microsoft ASP.NET 4.5 includes various authentication models, including local authentication providers, claim-based authentication systems, and federated authentication systems. You need to know how to use these authentication models to implement authentication functionality in your web application. You should also know how to authorize users and roles in your application to restrict information access according to user and role membership.

Objectives

After completing this module, you will be able to:

- Implement authentication and authorization systems in your web application.
- Manage users and roles in your web application.
- Authorize users and roles in your web application.

Lesson 1
Implementing Authentication and Authorization

Membership providers, which were introduced in ASP.NET 2.0, help you to create secure authentication and authorizations systems for web applications. You need to know how to use the membership provider model in ASP.NET 4.5, to change authentication methods with minimum changes to the code. You can use claim-based authentication and federated authentication to allow external users to authenticate themselves, and access and use the web application. You should know how to configure restrictions in the web application to ensure that users access only specific information that is permitted and relevant to them.

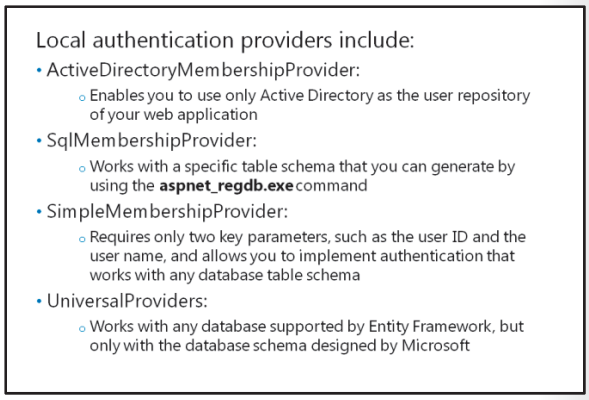
Lesson Objectives

After completing this lesson, you will be able to:

- List the local authentication options available for an ASP.NET 4 web application.
- Describe how client-based authentication works.
- Describe how federated authentication works.
- Describe how to restrict access to resources.
- Authorize access to specific users to MVC 4 web application controller actions.

Local Authentication Providers

Authentication providers include code that runs when ASP.NET needs to authorize a user. The code authenticates users by using the information stored in back-end databases, such as Active Directory or Microsoft SQL Server. The membership system in ASP.NET 4.5 includes the following authentication providers: **ActiveDirectoryMembershipProvider**, **SqlMembershipProvider**, **SimpleMembershipProvider**, and **UniversalProviders**. The **SimpleMembershipProvider** and **UniversalProviders** are new authentication providers that support the OAuth authentication mechanism.



The ASP.NET 4.5 membership system allows application developers to switch amongst authentication providers, without modifying the code.

The following is a description of the authentication providers**—ActiveDirectoryMembershipProvider** and **SqlMembershipProvider**:

- *ActiveDirectoryMembershipProvider*. This provider class is defined in the **System.Web.Security** namespace, and the provider enables you to use Active Directory as the membership and role repository of your web application.
- *SqlMembershipProvider*. This provider class is defined in the **System.Web.Security** namespace. The provider works with a specific table schema that you can generate by using the **aspnet_regdb.exe** command.

ActiveDirectoryMembershipProvider and **SqlMembershipProvider** are configured to work only with a specific table schema or directory. To overcome this restriction, Microsoft developed **SimpleMembershipProvider** and **UniversalProviders** to replace **ActiveDirectoryMembershipProvider** and **SqlMembershipProvider**.

- *SimpleMembershipProvider*. This is a new generation of membership providers that works with the SQL Server, SQL Server Compact Edition, Windows Azure SQL Database, and other versions of SQL Server. **SimpleMembershipProvider** requires only three key parameters—table name, user ID, and user name. You use this provider to implement authentication that works with any SQL Server database table schema.
- *UniversalProviders*. This is a set of membership providers that works with any database that Entity Framework supports. However, these providers work only with the database schema designed by Microsoft. While initializing a universal provider, if the schema does not exist in the database, the provider generates the schema.

Question: What is the benefit of using **SimpleMembershipProvider**?

Claims-Based Authentication

Claims-based authentication is a model that facilitates single sign-on. Single sign-on is a feature that allows you to receive a claim when you log on to a centralized authentication system. The claim is a ticket that authentication systems use to authenticate user logons. The claim contains user identity information that helps authentication systems identify users. With claims-based authentication systems, you can focus efforts on developing business functions, rather than worrying about authenticating users.

Claim-based authentication:

- Facilitates single sign-on
- Helps authenticate users
- Helps store user account information
- Integrates the application with the identity systems of other platforms or companies

Claims-based authentication facilitates:

- Authenticating users to access applications.
- Storing user account information and passwords.
- Checking enterprise directories for user information.
- Integrating the application with the identity systems of other platforms or companies.

To implement claims-based authentication in your web application, you need to use Windows Identity Foundation (WIF) 4.5. WIF 4.5.is a set of .NET Framework classes that helps read information from the claims in a web application.

The following steps describe the functioning of claims-based authentication systems:

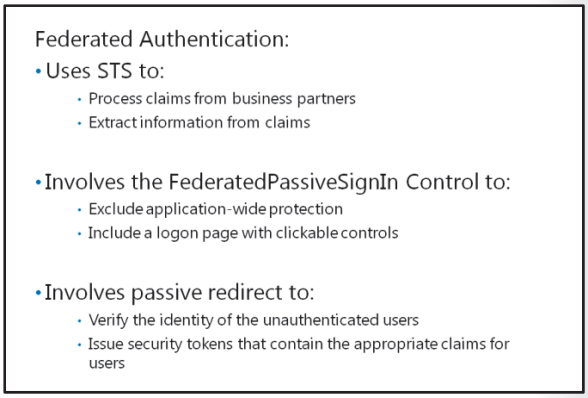
1. When an unauthenticated user requests a webpage, the request is redirected to the Identity Provider (IP) pages.
2. The IP requires you to enter credentials such as the user name and password.
3. After you enter the credentials, the IP issues a token. Then, the token is returned to the web browser.
4. The web browser is redirected to the webpage that you originally requested. WIF determines if the token satisfies the requirements to access the webpage. If the token satisfies all requirements, a

cookie is issued to establish a session. This cookie ensures that the authentication process occurs only once. Then, the authentication control is passed on to the application.

Question: What are the benefits of using claims-based authentication?

Federated Authentication

Federations rely on claims-based authentication to allow external parties such as trusted companies to access their applications. Federations use the WS-Federations claim standard to enable the exchange of claims between two parties in a standardized manner.



WIF provides support for federations by using the **WSFederationAuthenticationModule** HTTP module. **WSFederationAuthenticationModule** enables you to implement support for federations in your ASP.NET 4.5 application, without implementing individual logic.

Federated authentication enables Security Token Service (STS) to:

- Process the claims from business partners.
- Extract user information from the claims.

STS enables you to focus more on writing the business logic. It eliminates the need to manage the authentication information of business partners, in your web application.

Configuring the **WSFederationAuthenticationModule** helps specify the STS to which non-authenticated requests should be redirected. WIF provides two methods of federated authentication—**FederatedPassiveSignIn** and **Passive Redirect**.

FederatedPassiveSignIn Control

Consider cases when unauthenticated users try to access protected resources and you want to redirect these users to a logon page. For such cases, you can embed the **FederatedPassiveSignIn** control in the logon page of your web application, to redirect unauthenticated users to the logon page. The **FederatedPassiveSignIn** control is configured with issuer (STS) information.

You can use the **FederatedPassiveSignIn** control to:

- Exclude application-wide protection.
- Include a logon page with clickable controls.

Passive Redirect

Consider cases when unauthenticated users try to access a protected resource, and you want to redirect these users to an STS without using an application logon page. For such cases, you can use passive redirect.

Passive redirect enables STS to:

- Verify the identity of the unauthenticated users.
- Issue security tokens that contain the appropriate claims for users.

Passive redirect requires you to add **WSFederationAuthenticationModule** in the pipeline of the Hypertext Transfer Protocol (HTTP) modules, to identify unauthenticated user requests and redirect users

to the STS you specify. Adding **WSFederationAuthenticationModule** in the HTTP pipeline processes the claim information before passing the claim to the ASP.NET 4.5 engine.

You can instantiate **WSFederationAuthenticationModule** and use it to trigger the sign-on process. This functionality is similar to that of the **FederatedPassiveSignIn** control; however, passive redirect implements this sign-on functionality with minimum coding effort.

Question: What are the benefits of using federated authentication?

Restricting Access to Resources

You can restrict user access by implementing the **Authorize** attribute in a controller, instead of using the Web.config file as you would use in an ASP.NET WebForms application. The Web.config file requires physical files to exist, for access control to work. You cannot use the Web.config file to restrict user access because MVC applications route requests to actions, not pages.

The following code shows how to add the **Authorize** attribute to your controller class.

The **Authorize** attribute:
- Restricts user access to information
- Mandates that users should be authorized to access information

The **AllowAnonymous** attribute:
- Allows users to access specific portions of information

Authorizing Action Methods

```
 [Authorize()]
public ActionResult GetEmployee()
{
    return View();
}
```

Observe the **Authorize** attribute in the code sample. If you specify this attribute, ASP.NET 4.5 mandates that users should be authorized to access the view returned by the code sample.

If you add the **Authorize** attribute at the class level, the attribute requires users to log on before they can access anything in the controller class. To allow anonymous users to access a specific portion of your class, you can use the **AllowAnonymous** attribute.

The following code shows how to use the **AllowAnonymous** attribute.

Using the AllowAnonymous Attribute

```
 [AllowAnonymous()]
public ActionResult Register()
{
    return View();
}
```

Question: Why should you use the **Authorize** attribute, instead of the Web.config file to control authorization of pages in your ASP.NET MVC application?

Demonstration: How to Authorize Access to Controller Actions

In this demonstration, you will see how to:

- Generate authentication for access to a controller action.

- Handle unauthenticated requests for actions that require authentication by using ASP.NET.

Demonstration Steps

1. On the **DEBUG** menu of the **OperasWebSite - Microsoft Visual Studio** window, click **Start Debugging**.
2. On the Operas I Have Seen page, click **All Operas**.
3. On the Index of Operas page, click the **Create New** link.

Note: The Create an Opera page is displayed without logging on to the application. This enables anonymous users to create new operas.

4. In the Windows Internet Explorer window, click the **Close** button.
5. In the Solution Explorer pane of the **OperasWebSite - Microsoft Visual Studio** window, expand **OperasWebSite**, expand **Controllers**, and then click **OperaController.cs**.
6. In the OperaController.cs code window, locate the following code.

```
public ActionResult Create()
```

7. Place the mouse cursor before the located code, type the following code, and then press Enter.

```
[Authorize]
```

8. In the OperaController.cs code window, locate the following code.

```
[HttpPost]
public ActionResult Create(Opera newOpera)
```

9. Place the mouse cursor before the located code, type the following code, and then press Enter.

```
 [Authorize]
```

10. On the **FILE** menu of the **OperasWebSite - Microsoft Visual Studio** window, click **Save All**.
11. On the **DEBUG** menu of the **OperasWebSite - Microsoft Visual Studio** window, click **Start Debugging**.
12. On the Operas I Have Seen page, click **All Operas**.
13. On the Index of Operas page, click the **Create New** link.

Note: The **Login** view is now displayed and this prevents anonymous users from creating new operas.

14. On the Index of Operas page, click the **Register** link.
15. In the **User name** box of the Register page, type **David Johnson**.
16. In the **Password** box, type **Pa$$w0rd**, in the **Confirm password** box, type **Pa$$w0rd**, and then click **Register**.
17. On the Operas I Have Seen page, click **All Operas**.
18. On the Index of Operas page, click the **Create New** link.

Note: The Add an Opera page is displayed because the request is authenticated.

19. In the Windows Internet Explorer window, click the **Close** button.
20. In the **OperasWebSite - Microsoft Visual Studio** window, click the **Close** button.

Lesson 2
Assigning Roles and Membership

Roles and memberships complement authentication features, to help you control all modes of access in web applications. To define access levels for different types of users, you need to know how to implement roles and users. ASP.NET 4.5 provides role providers and membership providers to help you assign roles and users. ASP.NET 4.5 also enables you to create custom role providers and custom membership providers. These custom providers allow you to store role and user information in data sources not supported by ASP.NET 4.5, such as a FoxPro or Oracle database.

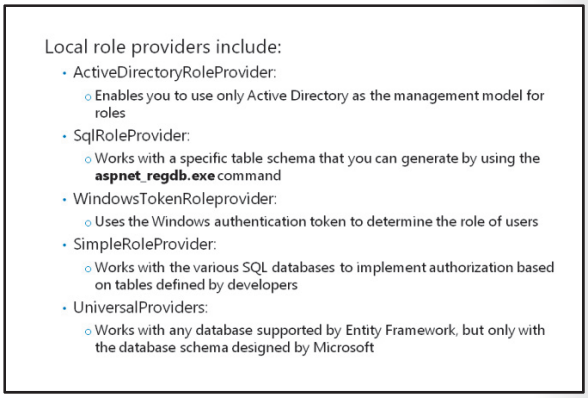
Lesson Objectives

After completing this lesson, you will be able to:

- Describe role providers in ASP.NET 4.5.
- Describe how to add user accounts to roles.
- Describe how to build custom role providers.
- Describe membership services.
- Describe how to build custom membership providers.
- Add code to an MVC 4 web application to help users reset their password.

Role Providers in ASP.NET 4.5

Role providers help develop applications by using a role-based security model. Similar to membership providers, two types of role providers are available in ASP.NET 4.5—**ActiveDirectoryRoleProvider** and **SqlRoleProvider**:



- *ActiveDirectoryRoleProvider*. This provider class is defined in the **System.Web.Security** namespace, and the provider enables you to use Active Directory as the management model for roles.
- *SqlRoleProvider*. This provider class is defined in the **System.Web.Security** namespace. The provider works with only a specific table schema in Microsoft SQL Server. You can generate this schema by using the **aspnet_regdb.exe** command.
- *WindowsTokenRoleProvider*. This provider uses the Windows authentication token to determine the role of users. Then, the provider checks if users belong to any group stored in the Windows authentication token that was generated when they first logged on. The format of the group name is **Domain\Group**.

ActiveDirectoryRoleProvider and **SqlRoleProvider** have restrictions such as lack of support for non-SQL databases or developer-defined schema. Therefore, Microsoft developed **SimpleRoleProvider** and **UniversalProviders** to replace **ActiveDirectoryRoleProvider** and **SqlRoleProvider**. The following list describes the **SimpleRoleProvider** and **UniversalProviders**:

- *SimpleRoleProvider*. This is a new generation of role provider that works with SQL Server, SQL Compact Editions, and others versions of SQL Server. **SimpleRoleProvider** enables you to implement authorization based on the table defined by application developers.
- *UniversalProviders*. This is a database agnostic version that works with any database that Entity Framework supports. However, the database schema is determined by Microsoft. The provider usually generates tables during initialization.

You can choose a role provider based on the membership provider that you select. You can also mix role providers; for example, you can combine **SimpleMembershipProvider** and **UniversalRoleProvider**. However, you should avoid mixing role providers because the manner in which each provider identifies users is different.

Question: What is the difference between **SimpleRoleProvider** and **SqlRoleProvider**?

Adding User Accounts to Roles

When you use a database-based role provider, such as **SimpleRoleProvider** or **UniversalRoleProvider**, you can load the initial data onto the database by directly editing the table. You can also edit the table by using the **AddUsersToRoles** function. To use the **AddUsersToRoles** function, you need to implement the custom management tool in your application. The custom management tool helps call the **AddUsersToRoles** function, to add roles to a database.

The following code shows how to use the **AddUsersToRoles** function.

To add users to roles:

- Implement the custom management tool in your application
- Use the **Authorize** attribute
- Ensure that the required role exists before using the **Authorize** attribute

Using the AddUsersToRoles Function

```
AddUserToRoles("Peter", new string[] {"Admin","Staff"})
```

You can also use the **AddUsersToRoles** function in business applications to add users to a role. However, you must ensure that the role to which you want to add users exists, before using the **AddUsersToRoles** function. ASP.NET 4.0 and all previous versions of ASP.NET include the Website Administration Tool (WSAT). Unfortunately, WSAT does not work with the **SimpleRoleProvider**.

Question: Why should we not use WSAT while using **SimpleRoleProvider**?

Building a Custom Roles Provider

ASP.NET 4.5 allows you to build custom role providers. Custom role providers enable you to implement role management that uses your own database schema and logic. To build a custom role provider, you need to create a class that inherits the **RoleProvider** class.

The following code shows how to build a custom role provider.

To build a custom role provider:

- Create a class that inherits the **RoleProvider** class
- Implement the **GetRolesForUser** function
- Modify the Web.config file

Creating a Custom Role Provider

```
public class CustomRoleProvider :
RoleProvider
{
}
```

In the **RoleProvider** class, you need to implement the **GetRolesForUser** function. The **GetRolesForUser** function takes the user name as the input and returns a list of roles to which the user belongs. You can write your own code to obtain role information from the database or other back-end stores.

The following code shows how to use the **GetRolesForUser** method.

Implementing the GetRolesForUser Function

```
public override string[] GetRolesForUser(string username)
{
          //code to return a list of roles for users
}
```

After implementing the **GetRolesForUser** function, you need to apply the custom role provider to the application by modifying the Web.config file.

The following code shows how to apply the custom role provider to the application.

Configuring a Site to Use a Custom Role Provider

```
<roleManager defaultProvider="CustomRoleProvider" enabled="true"
cacheRolesInCookie="false">
  <providers>
    <clear />
    <add name="CustomRoleProvider" type="CustomRoleProvider"  />
  </providers>
</roleManager>
```

Question: Why should you create a custom role provider?

Providing Membership Services

You need to add the **SimpleMembershipProvider** to the membership section of the Web.config file to use it for membership services.

The following code shows how to add **SimpleMembershipProvider** to the Web.config file.

To implement membership services in your web application:

- Modify the Web.config file
- Modify the AccountModel.cs file for any additional attributes
- Call the **WebSecurity.InitializeDatabaseConnection** function
- Add the **InitializeSimpleMembership** attribute in **AccountController** class

Configuring the Simple Membership Provider

```
<membership
defaultProvider="SimpleMembershipProvider"
>
  <providers>
    <clear/>
    <add name="SimpleMembershipProvider"
type="WebMatrix.WebData.SimpleMembershipProvider, WebMatrix.WebData" />
  </providers>
</membership>
```

Next, you need to create a User table. If you do not have a User table created already, you can modify the AccountModel.cs file, to generate the table. To view the AccountModel.cs file, you should select the Internet Application template, when you create the project.

In the following code sample, the **UserProfile** class from AccountModel.cs has been modified to add an additional property, Country.

Creating a User Table

```
 [Table("UserProfile")]
public class UserProfile
{
    [Key]
    [DatabaseGeneratedAttribute(DatabaseGeneratedOption.Identity)]
    public int UserId { get; set; }
    public string UserName { get; set; }
    public string Country { get; set; }
}
```

After modifying the AccountModel.cs file, you need to call the **WebSecurity.InitializeDatabaseConnection** function in the App.Start.cs file. The **WebSecurity.InitializeDatabaseConnection** function ensures that **SimpleMembershipProvider** is configured with connection strings to the database.

The following code shows how the membership provider connects to the User table, by using the **DBConn** connection string.

Connecting to the User Table

```
WebSecurity.InitializeDatabaseConnection("DBConn","User","id","Username",false)
```

If a specific table does not exist in a database, the **autoCreateTables** parameter helps indicate to the membership provider that it may create the table. To initialize **SimpleMembershipProvider**, you need to add the **InitializeSimpleMembership** attribute to the **AccountController** class.

The following code shows how to add the **InitializeSimpleMembership** attribute to the **AccountController** class.

Adding the InitializeSimpleMembership Attribute

```
 [Authorize]
[InitializeSimpleMembership]
public class AccountController : Controller
{
}
```

Question: Why should we use the **InitializeDatabaseConnection** method?

Building a Custom Membership Provider

If you need to add functionality to a membership provider, such as custom logic for authentication, you need to implement a custom membership provider.

The following code shows how to implement a custom membership provider.

A Custom Membership Provider

```
public class CustomMembershipProvider :
SimpleMembershipProvider
{
}
```

To build custom membership providers:

- Implement the custom membership provider
- Override the **ValidateUser** function
- Override the **Provider** constructor to add additional logic

After implementing the custom membership provider, you override the **ValidateUser** method defined in **SimpleMembershipProvider**. The following code shows how to override the **ValidateUser** method defined in **SimpleMembershipProvider**.

Overriding the ValidateUser Method

```
public override bool ValidateUser(string username, string password)
{
}
```

In the preceding code sample, the **ValidateUser** method validates the user name and password against the membership store.

Perhaps, you want to add logic to get the decryption key for passwords. To add this logic, you can override the **Provider** constructor, when the custom membership provider is initializing. Overriding the **Provider** constructor initializes the encryption engine.

Overriding the Provider Constructor

```
public CustomAdminMembershipProvider(SimpleSecurityContext simpleSecurityContext)
{
}
```

After adding logic to the custom membership provider, you need to modify the Web.config file to apply the custom provider class to the application.

Adding a Custom Provider to Web.config

```
<membership defaultProvider="CustomMemberProvider">
      <providers>
        <clear/>
        <add name="CustomMemberProvider" type=" CustomAdminMembershipProvider " />
      </providers>
```

```
    </membership>
```

Question: Why should you implement a custom membership provider?

Demonstration: How to Reset a Password

In this demonstration, you will see how to write code that uses a Membership Services Provider to reset the user's password. Using this code, you can enable users to control their own passwords and reset the password without involving a site administrator.

Demonstration Steps

1. In the Solution Explorer pane of the **OperasWebSite - Microsoft Visual Studio** window, under **Controllers**, click **AccountController.cs**.
2. In the AccountController.cs code window, locate the following comment.

```
//Add Reset Password Code Here
```

3. In the AccountController.cs code window, replace the located comment with the following code, and then press Enter.

```
try
{
}
catch (Exception)
{
}
```

4. In the **try** code block, type the following code.

```
changePasswordSucceeded = Membership.Provider.ChangePassword(User.Identity.Name,
model.OldPassword, model.NewPassword);
```

5. In the **catch** code block, type the following code.

```
changePasswordSucceeded = false;
```

6. On the **FILE** menu of the **OperasWebSite - Microsoft Visual Studio** window, click **Save All**.
7. On the **DEBUG** menu of the **OperasWebSite - Microsoft Visual Studio** window, click **Start Debugging**.
8. On the Operas I Have Seen page, click the **Log In** link.
9. On the Logon page, in the **User name** box, type **David Johnson**, in the **Password** box, type **Pa$$w0rd**, and then click **Log in**.
10. On the Operas I Have Seen page, click the **Reset Password** link.
11. On the ResetPassword page, in the **Current password** box, type **Pa$$w0rd**, and then in the **New password** box, type **Pa$$w0rd2**.
12. In the **Confirm new password** box, type **Pa$$w0rd2**, and then click **Change Password**.

Note: On the ResetPassword page, the message, "Your password has been changed." is displayed.

13. In the Windows Internet Explorer window, click the **Close** button.
14. In the OperasWebSite – Microsoft Visual Studio window, click the **Close** button.
15. If the Microsoft Visual Studio warning message appears, click **Yes**.

Lab: Controlling Access to ASP.NET MVC 4 Web Applications

Scenario

A large part of the functionality for your proposed Photo Sharing application is in place. However, stakeholders are concerned about security because there are no restrictions on the tasks that users can complete. The following restrictions are required:

- Only site members should be able to add or delete photos.
- Only site members should be able to add or delete comments.

You have been asked to resolve these concerns by creating a membership system for the Photo Sharing application. Visitors should be able to register as users of the web application and create user accounts for themselves. After registration, when the users log on to the application, they will have access to actions such as adding and deleting photos and comments. Anonymous users will not have access to perform these actions. Additionally, registered users should also be able to reset their own password.

Objectives

After completing this lab, you will be able to:

- Configure a web application to use ASP.NET Form Authentication with accounts stored in Windows Azure SQL database.
- Write models, controllers, and views to authenticate users in a web application.
- Provide access to resources in a web application.
- Enable users to reset their own password.

Lab Setup

Estimated Time: 90 minutes

Virtual Machine: **20486B-SEA-DEV11**

User name: **Admin**

Password: **Pa$$w0rd**

Note: In Hyper-V Manager, start the MSL-TMG1 virtual machine if it is not already running.

- Before starting the lab, you need to enable the **Allow NuGet to download missing packages during build** option, by performing the following steps:
  i. On the **TOOLS** menu of the Microsoft Visual Studio window, click **Options**.
  ii. In the navigation pane of the **Options** dialog box, click **Package Manager**.
  iii. Under the Package Restore section, select the **Allow NuGet to download missing packages during build** checkbox, and then click **OK**.
- After completing the lab, take a Snapshot of the **20486B-SEA-DEV11** virtual machine. Ensure that this Snapshot is applied before initiating the lab in Module 13.

Exercise 1: Configuring Authentication and Membership Providers

Scenario

You want to use a Windows Azure SQL database to store user accounts and membership information.

In this exercise, you will:

- Create a Windows Azure SQL database.
- Configure a provider to connect to the database.

The main tasks for this exercise are as follows:

1. Configure a new Windows Azure SQL database.
2. Install universal providers.
3. Configure providers in Web.config.

▶ Task 1: Configure a new Windows Azure SQL database.

1. Start the virtual machine, and log on with the following credentials:
   - Virtual Machine: **20486B-SEA-DEV11**
   - User name: **Admin**
   - Password: **Pa$$w0rd**
2. Log on to the Windows Azure portal by using the following information:
   - Portal address: **http://manage.windowsazure.com**
   - User name: <*username*>
   - Password: <*password*>
3. Create a new database server and a new custom database by using the following information:
   - Database name: **PhotoAppServices**
   - Database server: **New SQL Database Server**
   - Logon name: <*your first name*>
   - Logon password: **Pa$$w0rd**
   - Logon password confirmation: **Pa$$w0rd**
   - Region: <*a region close to you*>
4. In the list of allowed IP addresses for the **PhotoAppServices** database, add the following IP address ranges:
   - Rule name: **First Address Range**
   - Start IP address: <*first address in range*>
   - End IP address: <*last address in range*>

▶ Task 2: Install universal providers.

1. Open the **PhotoSharingApplication.sln** file from the following location:
   - File location: Allfiles (D):\Labfiles\Mod11\Starter \PhotoSharingApplication
2. Install the **Microsoft ASP.NET Universal Providers** package in the **PhotoSharingApplication** project by using the NuGet Package Manager.

▶ Task 3: Configure providers in Web.config.

1. Remove the connection string named **DefaultConnection** from the top-level Web.config file.
2. Obtain the connection string for the PhotoAppServices database and add it to the Web.config file.

3. Configure the web application to use Forms authentication in Web.config, by using the following information:
    - Parent element: **<system.web>**
    - Element: **<authentication>**
    - Mode: **Forms**
4. Configure the logon page for the web application by using the following information:
    - Parent element: **<authentication>**
    - Element: **<forms>**
    - Logon URL: **~/Account/Login**
    - Timeout: **2880**
5. Configure the default profile provider to use the connection string named **PhotoAppServices**.
6. Configure the Default Membership Provider to use the connection string named **PhotoAppServices**.
7. Configure the Default Role Provider to use the connection string named **PhotoAppServices**.
8. Configure the Default Session Provider to use the connection string named **PhotoAppServices**.
9. Save all the changes.

Results: At the end of this exercise, you will have created a Photo Sharing application that is configured to use Windows Azure SQL database for user accounts and membership information. In subsequent exercises, you will add model classes, actions, and views to implement authentication for this database.

Exercise 2: Building the Logon and Register Views

Scenario

You have configured the Photo Sharing application to connect to Windows Azure SQL database for authentication and membership services. However, to use forms authentication in an MVC application, you need to build model classes, controllers, and views that enable users to log on, log off, and register for an account.

In this exercise, you will:

- Add model classes.
- Add controllers.
- Import logon and register views.
- Test the developed components.

The main tasks for this exercise are as follows:

1. Add account model classes.
2. Add an account controller.
3. Import Logon and Register views.
4. Add authentication controls to the Template view.
5. Test registration, log on, and log off.

▶ Task 1: Add account model classes.

1. Add a new **Class** file named **AccountModelClasses.cs** to the **Models** folder.
2. Add references to the following namespaces, to the new class file:
   - System.ComponentModel.DataAnnotations
   - System.Data.Entity
3. Remove the **AccountModelClasses** class and add a new class by using the following information:
   - Scope: **Public**
   - Name: **UsersContext**
   - Inherit: **DbContext**
4. In the **UsersContext** class, create a constructor that passes the **PhotoAppServices** connection string to the base class constructor.
5. In the **AccountModelClasses.cs** code file, add a new public class named **Login**.
6. Add a new property to the **Login** class by using the following information:
   - Scope: **public**
   - Type: **string**
   - Name: **UserName**
   - Access: **Read/Write**
   - Display name: **User name**
   - Use the **Required** annotation.
7. Add a new property to the **Login** class by using the following information:
   - Scope: **public**
   - Type: **string**
   - Name: **Password**
   - Access: **Read/Write**
   - Data type: **Password**
   - Use the **Required** annotation.
8. Add a new property to the **Login** class by using the following information:
   - Scope: **public**
   - Type: **bool**
   - Name: **RememberMe**
   - Access: **Read/Write**
   - Display name: **Remember me?**
9. In the **AccountModelClasses.cs** code file, add a new public class named **Register**.
10. Add a new property to the **Register** class by using the following information:
    - Scope: **public**
    - Type: **string**

- Name: **UserName**
- Access: **Read/Write**
- Display name**: User name**
- Use the **Required** annotation.

11. Add a new property to the **Register** class by using the following information:
    - Scope: **public**
    - Type: **string**
    - Name: **Password**
    - Access: **Read/Write**
    - Data type: **Password**
    - Use the **Required** annotation.
12. Add a new property to the **Register** class by using the following information:
    - Scope: **public**
    - Type: **string**
    - Name: **ConfirmPassword**
    - Access: **Read/Write**
    - Data type: **Password**
    - Display name: **Confirm password**
    - Ensure that this property matches the **Password** property by using the **Compare** annotation.
13. Save all the changes.

▶ Task 2: Add an account controller.

1. Add a new controller named **AccountController** to the MVC web application by using the **Empty MVC controller** template.
2. Delete the default **Index** action from the **AccountController** file and add **using** statement references for the following namespaces:
    - System.Web.Security
    - PhotoSharingApplication.Models
3. Create a new action method in the **AccountController** class by using the following information:
    - Scope: **public**
    - Return type: **ActionResult**
    - Name: **Login**
    - Parameter: a string named **returnUrl**
4. In the **Login** action, store the **returnUrl** value in the **ViewBag** collection, and then return a view named **Login**.
5. Create a new action method in the **AccountController** class by using the following information:
    - HTTP verb: **POST**
    - Scope: **public**

- Return type: **ActionResult**
- Name: **Login**
- Parameters: a Login object named **model** and a string named **returnUrl**

6. Within the **Login** action method code block for the **HTTP POST** verb, check if the **ModelState** is valid.
7. Add an **if…else** statement to check the user's credentials by using the following information:
   - Method: **Membership.ValidateUser**
   - User name: **model.UserName**
   - Password: **model.Password**
8. If the user's credentials are correct, authenticate the user by using the following information:
   - Method: **FormsAuthentication.SetAuthCookie**
   - User name: **model.UserName**
   - Create persistent cookie: **model.RememberMe**
9. If **returnUrl** is a local URL, redirect the user to the **returnUrl**. Otherwise, redirect the user to the **Index** action of the **Home** controller.
10. If the user's credentials are incorrect, add a model error to the **ModelState** object by using the following information:
    - Key: An empty string
    - Error message: **The username or password is incorrect**
11. If the **ModelState** is not valid, return the current view and pass the **model** object so that the user can correct errors.
12. Create a new action method in the **AccountController** class by using the following information:
    - Scope: **public**
    - Return type: **ActionResult**
    - Name: **LogOff**
    - Parameters: None
13. In the **LogOff** action, log off the user, and then redirect to the **Index** action of the **Home** controller by using the **FormsAuthentication.SignOut()** method.
14. Create a new action method in the **AccountController** class by using the following information:
    - Scope: **public**
    - Return type: **ActionResult**
    - Name: **Register**
    - Parameters: None
15. In the **Register** action, return the **Register** view.
16. Create a new action method in the **AccountController** class by using the following information:
    - HTTP verb: **POST**
    - Scope: **public**

- Return type: **ActionResult**
- Name: **Register**
- Parameter: a **Register** object named **model**.

17. Within the **Register** action method code block for the **HTTP POST** verb, check if the **ModelState** is valid.
18. If the **ModelState** is valid, create a **try…catch** block that catches exceptions of the type **MembershipCreateUserException e**.
19. In the **try** block, create a new user with the right user name and password by using the **Membership.CreateUser** method. Store the result in a **MembershipUser** object named **NewUser**.
20. Authenticate the new user and redirect the browser to the **Index** action of the **Home** controller.
21. In the **catch** block, add a model error to the **ModelState** object by using the following information:
    - Key: **Registration Error**
    - Error message: Report the error status code as a string
22. If the **ModelState** is not valid, return the current view and pass the model object so that the user can correct errors.
23. Save all the changes.

▶ Task 3: Import Logon and Register views.

1. Add a new folder named **Account** to the **Views** folder.
2. Add the **Login.cshtml** file to the **Account** folder from the following location:
    - File location: **Allfiles (D):\Labfiles\Mod11\Account Views**
3. Add the **Register.cshtml** file to the **Account** folder from the following location:
    - File location: **Allfiles (D):\Labfiles\Mod11\Account Views**

▶ Task 4: Add authentication controls to the Template view.

1. Open the **_MainLayout.cshtml** page for editing.
2. Immediately before the **DIV** element with **clear-floats** class, insert a **DIV** element with **login-controls** class.
3. In the new **DIV** element, write a Razor **if… else** code block that checks whether the current request is from an authenticated user.
4. If the request is from an authenticated user, render a greeting message that includes the authenticated user's name.
5. After the greeting message, render a link to the **LogOff** action by using the following information:
    - Helper: **Html.ActionLink()**
    - Link text: **Log Off**
    - Action name: **LogOff**
    - Controller name: **Account**
6. If the request is from an anonymous user, render a link to the **Login** action by using the following Information:
    - Helper: **Html.ActionLink()**

- Link text: **Log In**
- Action name: **Login**
- Controller name: **Account**

7. After the **Log In** link, render a link to the **Register** action by using the following Information:
   - Helper: **Html.ActionLink()**
   - Link text: **Register**
   - Action name: **Register**
   - Controller name: **Account**
8. Save all the changes.

▶ Task 5: Test registration, log on, and log off.

1. Start the web application in debugging mode and register a user account by using the following information:
   - User name: **David Johnson**
   - Password: **Pa$$w0rd**
2. Log off and then log on with the credentials you just created.
3. Stop debugging.

Results: At the end of this exercise, you will create a Photo Sharing application in which users can register for an account, log on, and log off.

Exercise 3: Authorizing Access to Resources

Scenario

Now that you have enabled and tested authentication, you can authorize access to resources for both anonymous and authenticated users.

You should ensure that:

- Only site members can add or delete photos.
- Only site members can add or delete comments.
- The account controller actions are authorized properly.
- Only authenticated users see the_**Create** view for comments in the **Display** view.

The main tasks for this exercise are as follows:

1. Restrict access to Photo actions.
2. Restrict access to the Comment actions.
3. Restrict access to the Account actions.
4. Check authentication status in a view.
5. Test authorization.

▶ Task 1: Restrict access to Photo actions.

1. In the **PhotoController.cs** file, add the **[Authorize]** annotation to ensure that only authenticated users can access the **Create** action for the GET requests.
2. Add the **[Authorize]** annotation to ensure that only authenticated users can access the **Create** action for the **HTTP POST** verb.
3. Add the **[Authorize]** annotation to ensure that only authenticated users can access the **Delete** action.
4. Add the **[Authorize]** annotation to ensure that only authenticated users can access the **DeleteConfirmed** action for the **HTTP POST** verb.
5. Save all the changes.

▶ Task 2: Restrict access to the Comment actions.

1. In the **CommentController.cs** file, add the **[Authorize]** annotation to ensure that only authenticated users can access the **_Create** action.
2. Add the **[Authorize]** annotation to ensure that only authenticated users can access the **Delete** action.
3. Add the **[Authorize]** annotation to ensure that only authenticated users can access the **DeleteConfirmed** action for the **HTTP POST** verb.
4. Save all the changes.

▶ Task 3: Restrict access to the Account actions.

1. In the **AccountController.cs** file, add the **[Authorize]** annotation to ensure that only authenticated users can access all actions by default.
2. Add the **[AllowAnonymous]** annotation to ensure that anonymous users can access the **Login** action.
3. Add the **[AllowAnonymous]** annotation to ensure that anonymous users can access the **Login** action for the **HTTP POST** verb.
4. Add the **[AllowAnonymous]** annotation to ensure that anonymous users can access the **Register** action.
5. Add the **[AllowAnonymous]** annotation to ensure that anonymous users can access the **Register** action for the **HTTP POST** verb.
6. Save all the changes.

▶ Task 4: Check authentication status in a view.

1. Open the **_CommentsForPhoto.cshtml** partial view.
2. In the **_CommentsForPhoto.cshtml** partial view, add an **if** statement to ensure that the **_Create** partial view is only displayed if the request is authenticated.
3. If the request is not authenticated, render a link to the **Login** action of the **Account** controller to display the text **To comment you must log in**.
4. Save all the changes.

▶ Task 5: Test authorization.

1. Start the web application in debugging mode and then attempt to add a new photo to the web application, without logging on to the application.

2. Without logging on to the application, view any photo in the application and attempt to add a comment.
3. Log on to the web application by using the following credentials:
   - User name: **David Johnson**
   - Password: **Pa$$w0rd**
4. Add a comment of your choice to the photo by using the following information:
   - Subject: **Authenticated Test Comment**
5. Stop debugging.

Results: At the end of this exercise, you will have authorized anonymous and authenticated users to access resources in your web application.

Exercise 4: Optional—Building a Password Reset View

Scenario

Site visitors can now register as users of the Photo Sharing application and log on to the site so that they can add photos and comments. However, they do not have the facility to change their password. In this exercise, you will create a password reset page by using the membership services provider.

Complete this exercise if time permits.

The main tasks for this exercise are as follows:

1. Add a local password model class.
2. Add reset password actions.
3. Import the reset password view.
4. Add a link to the reset password view.
5. Test password reset.

▶ Task 1: Add a local password model class.

1. Add a new class to the **AccountModelClasses.cs** file by using the following information:
   - Scope: **public**
   - Name of the class: **LocalPassword**
2. Add a new property to the **LocalPassword** class by using the following information:
   - Scope: **public**
   - Type: **string**
   - Name: **OldPassword**
   - Access: **Read/Write**
   - Data type: **Password**
   - Annotation**: [Required]**
   - Display name: **Current password**
3. Add a new property to the **LocalPassword** class by using the following information:

- Scope: **public**
- Type: **string**
- Name: **NewPassword**
- Access: **Read/Write**
- Data type: **Password**
- Annotation**: [Required]**
- Display name: **New password**

4. Add a new property to the **LocalPassword** class by using the following information:
   - Scope: **public**
   - Type: **string**
   - Name: **ConfirmPassword**
   - Access: **Read/Write**
   - Data type: **Password**
   - Display name: **Confirm new password**
   - Use the **Compare** annotation to ensure this property matches the **NewPassword** property.
5. Save all the changes.

▶ Task 2: Add reset password actions.

1. In the **AccountController** class, add a new enumeration by using the following information:
   - Scope: **public**
   - Name: **ManageMessageId**
   - Values: **ChangePasswordSuccess**, **SetPasswordSuccess**
2. Add a new action to the **AccountController** class by using the following information:
   - Scope: **public**
   - Return type: **ActionResult**
   - Name: **ResetPassword**
   - Parameters: an optional **ManageMessageId** object named **message**
3. If the **message** parameter is not null, set the **ViewBag.StatusMessage** property to **Your password has been changed**.
4. Set the **ViewBag.ReturnUrl** property to the URL of the **ResetPassword** action.
5. Return a view named **ResetPassword**.
6. Add a new action to the **AccountController** class by using the following information:
   - Annotation: **HttpPost**
   - Scope: **public**
   - Return type: **ActionResult**
   - Name: **ResetPassword**
   - Parameter: a **LocalPassword** object named **model**

7. In the new **ResetPassword** action for the HTTP POST verb, set the **ViewBag.ReturnUrl** property to the URL of the **ResetPassword** action.
8. Include an **if** statement to check whether the **ModelState** is valid.
9. If the **ModelState** is valid, create a new Boolean variable named **changePasswordSucceeded**, and then add a **try… catch** block that catches all exceptions.
10. In the **try** block, change the user's password by using the following information:
    - Method: **Membership.Provider.ChangePassword**
    - User name: **User.Identity.Name**
    - Old password: **model.OldPassword**
    - New password: **model.NewPassword**
    - Store the result in **changePasswordSucceeded**
11. In the **catch** block, set the **changePasswordSucceeded** variable to **false**.
12. After the **try…catch** code block, if **changePasswordSucceeded** is true, redirect to the **ResetPassword** action and pass the **ManageMessageId.ChangePasswordSucccess** value to the **message** parameter.
13. If **changePasswordSucceeded** is false, add a model error to the **ModelState** object by using the following information:
    - Key: An empty string
    - Message: **The current password is incorrect or the new password is invalid**
14. If the **ModelState** is not valid, return the current view and pass the **model** object so that the errors can be corrected.
15. Save all the changes.

▶ Task 3: Import the reset password view.

1. Add the **ResetPassword.cshtml** file to the **Views/Account** folder from the following location:
    - **Allfiles (D):\Labfiles\Mod11\Account Views**

▶ Task 4: Add a link to the reset password view.

1. Add a new link to the **_MainLayout.cshmtl** template view by using the following information:
    - Position: After the link to the **LogOff** action
    - Text: **Reset**
    - Action: **ResetPassword**
    - Controller: **Account**
2. Save all the changes.

▶ Task 5: Test password reset.

1. Start the web application in debugging mode, and then log on with the following credentials:
    - User name: **David Johnson**
    - Password: **Pa$$w0rd**
2. Change the password from **Pa$$w0rd** to **Pa$$w0rd2**.

3. Stop debugging and close the open windows.

Results: At the end of this exercise, you will build a Photo Sharing application in which registered users can reset their own password.

Question: In Exercise 3, when you tried to add a photo before logging on to the application, why did ASP.NET display the **Login** view?

Question: How can you ensure that only Adventure Works employees are granted access to the **Delete** action of the **Photo** controller?

Module Review and Takeaways

In this module, you discussed the various membership and role providers in ASP.NET 4.5. You compared the benefits of using **SimpleProviders** and **UniversalProviders** with the benefits of using other providers, for database-centric applications. However, for applications that need to use Windows for authentication and authorization, you can use the existing role and membership providers. You also viewed how to implement custom providers, to add functionalities such as password encryption to your web application.

Review Question(s)

Question: What is the key difference between implementing authentication and authorization in ASP.NET 4.5 from implementing the same in the previous versions of ASP.NET?

Real-world Issues and Scenarios

When you create web applications, you may need to create custom providers because you do not want to use the schema provided by Microsoft. However, you can use **SimpleProviders** to remove the need to develop custom providers and reduce the effort required for building applications.

Module 12

Building a Resilient ASP.NET MVC 4 Web Application

Contents:

Module Overview

Security is always the top priority for web applications, because publicly accessible web applications are usually the targets of different types of web attacks. Hackers use web attacks to access sensitive information. To help prevent these attacks, you need to know how to use AntiXSS and request validation for your web application. Using various security mechanisms, you can build a secure web application. Another problem that developers usually face is the need to retain information amongst multiple postbacks. You need to know how to use state management techniques to retain information for multiple HTTP requests. Using state management, you can help reduce the need for users to re-enter information every time they need to place a request.

Objectives

After completing this module, you will be able to:

- Develop secure sites.
- Implement state management.

Lesson 1
Developing Secure Sites

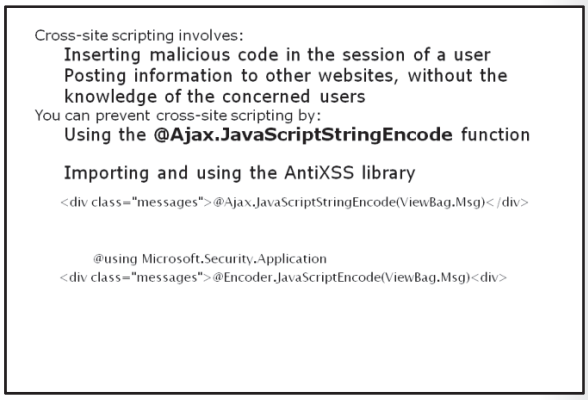
Web applications are often subject to security attacks. These attacks prevent the applications from functioning properly, and the attackers try to access sensitive information stored in the underlying data store. ASP.NET provides built-in protection mechanisms that help prevent such attacks. However, such mechanisms also tend to affect the operation of your applications. You should know when to enable or disable the protection mechanisms to avoid any impact on the functionalities of your application. You also need to know how to use Secure Sockets Layer (SSL) to prevent unauthorized access to information during information transmission.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe how to avoid malicious attacks that use cross-site scripting.
- Describe the common attack techniques used by malicious users.
- Describe how to disable attack protection in your MVC application.
- Decide whether to disable attack protection in various scenarios.
- Describe the role of Secure Sockets Layer in helping secure web applications.

Cross-Site Scripting



Cross-site scripting involves the malicious insertion of scripts in a user session. Cross-site scripting poses information to other websites, by using the authentication information of users, without their knowledge. For example, consider the code inserted into a web application to maliciously post messages on social networking websites, without the knowledge of the users. When a script is inserted into a web application, the script has full access to the Document Object Model (DOM) of the HTML. This access allows the attacker to create fake input boxes in the application, create fake users, and post fake information on the web application.

The cross-site scripting attack usually takes input from improperly escaped output. These scripting attacks usually impact query strings. For example, consider the following URL:

http://localhost/Default1/?msg=Hello

The following code shows how to access the msg querystring parameter from a controller action.

Accessing a Query String Parameter

```
public ActionResult Index(string msg)
{
    ViewBag.Msg = msg;
    return View();
}
```

The following code shows how to display the value of the querystring parameter in a view.

Displaying the Query String Parameter

```
<div class="messages">@ViewBag.Msg<div>
```

After running the preceding code samples, the application should display the resultant word, **Hello**. Now, consider a scenario where the query string parameter **msg** is changed to a less benign value resulting in the following URL:

http://localhost/Default1/?msg=<script>alert('pwnd')</script>

As a result, the script block included in the query string is displayed to users. In such cases, attackers can inject malicious code into your app by using the value of a query string parameter.

ASP.NET includes request validation, to help protect the input values that are subject to cross-site scripting attacks. However, attackers can bypass this mechanism by using encoding to subvert common cross-site scripting filters. For example, here is the same query string, this time encoded:

http://localhost/Default1/?msg=Jon\x3cscript\x3e%20alert(\x27pwnd\x27)%20\x3c/script\x3e

You can modify the view class to use the **@Ajax.JavaScriptStringEncode** function to help prevent cross-site scripting attacks, instead of directly using the input from query strings. The following line of code shows how to use the **@Ajax.JavaScriptStringEncode** function.

Using the JavaScriptStringEncode Function

```
<div class="messages">@Ajax.JavaScriptStringEncode(ViewBag.Msg)<div>
```

You can also import the AntiXSS library to check the query string content for possible attacks. The AntiXSS library is part of the Web Protection Library, which was developed by Microsoft to detect more complex web attacks than those that the request validation of ASP.NET can detect.

 **Additional Reading:** To view more information on the AntiXSS library, go to http://go.microsoft.com/fwlink/?LinkID=293690&clcid=0x409

After importing the AntiXSS library in your MVC application, you can use the library to encode any output content in HTML.

The following code shows how to use the AntiXSS library.

Using the AntiXSS Library

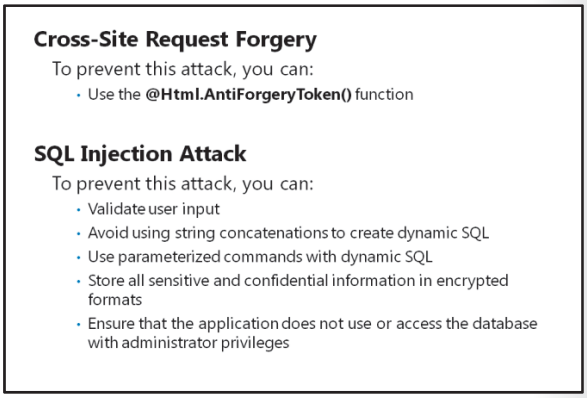
```
@using Microsoft.Security.Application
<div class="messages">@Encoder.JavaScriptEncode(ViewBag.Msg)<div>
```

The code in the preceding sample illustrates how to encode input values by using the JavaScriptEncode method of the AntiXSS library, when displaying output in HTML. This practice ensures that the input values are safe for display.

Question: What causes cross-site scripting attacks?

Other Attack Techniques

In addition to cross-site scripting attacks, hackers can use other types of attacks, including cross-site request forgery and SQL injection attacks to subvert web applications.



Cross-Site Request Forgery

Cross-site request forgery (CSRF) is an attack that occurs when you open a URL in a web browser, by using your user context, without knowing that you are allowing attackers to make changes to your system. For example, consider that your application uses query strings to pass information to other applications. You receive an email message with a link such as the following:

```
<a href="http://localhost/Default1/?id=100">Click Me</a>
```

When you click the link, the action associated with the view runs on your web browser. Because, you are an authenticated user in the application, the attacker can now access your system.

You can prevent CSRF by using the following rules:

1. Ensure that a GET request does not replay by clicking a link. The HTTP specifications for GET requests imply that GET requests should be used only for retrieval and not for state modifications.
2. Ensure that a request does not replay if an attacker uses JavaScript to simulate a form POST request.
3. Prevent any data modifications that use the GET request. These modifications should require some user interaction. This practice of introducing user interaction does not help prevent form-based attacks. However, user interaction limits several types of easier attacks, such as malicious links embedded in XSS-compromised sites.

The **@Html.AntiForgeryToken()** function helps protect your system from CSRF by using unique tokens that are passed to the application along with requests. The **@Html.AntiForgeryToken()** function uses not only a hidden form field but also a cookie value, making it more difficult to forge a request.

The following code shows how to use the **@Html.AntiForgeryToken()** function in a view.

Using an Anti-Forgery Token

```
@using (Html.BeginForm())
{
  @Html.AntiForgeryToken();
  @Html.EditorForModel();
  <input type="submit" value="Submit" />
}
```

The following code shows how to force Anti-Forgery token checking in a controller by using the ValidateAntiForgeryToken attribute.

Forcing Anti-Forgery Token Checking

```
 [HttpPost]
[ValidateAntiForgeryToken()]
public ActionResult Index(User user)
{
   Return View();
}
```

Note the **ValidateAntiForgeryToken()** attribute in the preceding code sample. This attribute enables the controller to check if the user input from the HTML form includes the token generated by the **@Html.AntiForgeryToken()**, before accepting a request.

SQL Injection Attack

SQL injection attacks are similar to cross-site scripting attacks. However, the difference is that the user input is used to create dynamic SQL, instead of HTML. Observe the following line of code.

```
cmd.CommandText = "select userID from Users where userID = '" +
Request.QueryString["user"] + "'";
```

Consider a scenario where an attacker modifies the query string as displayed in the following line of code.

```
user=1' AND 1=1 GO SELECT * from Users WHERE NOT 0='
```

The following line of code denotes the resultant SQL.

```
select userID from Users where userID = '1' AND 1=1 GO SELECT * from Users WHERE NOT
0=''
```

In this case, SQL returns everything from the **Users** table.

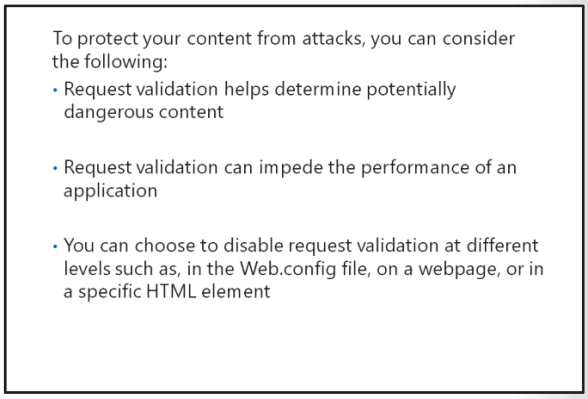
To prevent SQL injection attacks, you should:

1. Validate user input.
2. Avoid using string concatenations to create dynamic SQL.
3. Use parameterized commands with dynamic SQL.
4. Store all sensitive and confidential information in encrypted formats.
5. Ensure that the application does not use or access the database with administrator privileges.

Question: What is the best way to prevent SQL injection attacks if you cannot avoid dynamic SQL?

Disabling Attack Protection

Request validation is an ASP.NET feature that examines an HTTP request and determines if it contains potentially dangerous content. Potentially dangerous content can include any HTML markup or JavaScript code in the body, header, query string, or cookies of the request.



However, request validation can also cause the application to function improperly by preventing some input from entering the application page for processing. Consider a situation in which your application uses an HTML editor to generate HTML code for user input, before saving the input on the database. In this case, you may want to disable the request validation to allow the HTML editor to function properly.

You can choose to disable request validation at different levels, such as, in the Web.config file, on a webpage, or in a specific HTML element.

The following code shows how to disable request validation for your application in the Web.config file.

Disabling Request Validation

```
<system.web>
  <httpRuntime requestValidationMode="2.0" />
</system.web>
```

The following code shows how to add the **ValidateInput(false)** attribute to the controller, to disable request validation for an action in a controller.

Using the ValidateInput Attribute

```
 [HttpPost]
[ValidateInput(false)]
public ActionResult Edit(string comment)
{
    return View(comment);
}
```

The following code shows how to add the **AllowHtml** attribute to the **Prop1** property of the model, to disable request validation for this property

Using the AllowHtml Attribute

```
 [AllowHtml]
public string Prop1 { get;  set; }
```

You should consider using attack protection techniques that:

- Have a minimum impact on the application.
- Involve minimum fields to accept HTML elements in the request validations.

Question: Describe a scenario when you would want to disable request validation?

Secure Sockets Layer

Secure Sockets Layer (SSL) is an application layer protocol that helps:

- Encrypt content by using the public key infrastructure (PKI) keys.
- Protect the content that is transmitted between the server and client.
- Prevent unauthorized access of content during transmission.
- Reassure users that a site is genuine and certified.

SSL:
- Encrypts content by using the public key infrastructure (PKI) keys
- Protects the content that is transmitted between the server and client
- Prevents unauthorized access of content during transmission
- Involves using the **RequireHttps** attribute to redirect users to the SSL link

You can use SSL on views that accept user input if the input includes sensitive information, such as credit card information and passwords. Using SSL on such crucial views ensures the confidentiality of the content and the authenticity of the sender. However, you may not be able to analyze your code and easily detect if a user accesses the web application by using SSL.

ASP.NET MVC 4 includes the **RequireHttps** attribute that enables you to use SSL on the views that involve sensitive information. The **RequireHttps** attribute redirects users to the SSL link, if they request a view by using normal HTTP.

The following code shows how to add the **RequireHttps** attribute in your controller class and action methods to require a user to use the SSL protocol.

Using the RequireHttps Attribute

```
 [RequireHttps]
public class Controller1
{
    [RequireHttps]
    public ActionResult Edit()
    {
    }
}
```

You can use the **RequireHttps** attribute at the controller level or action level. This flexibility allows you to choose SSL when required in your web application.

Note that web servers require you to configure the PKI certificate so that the server accepts SSL connections. SSL certificates need to be purchased from a certificate authority (CA). During application development, you can use the self-sign certificate to simplify the configuration process.

Question: What action is required to be performed on the web server, before implementing SSL?

Lesson 2
State Management

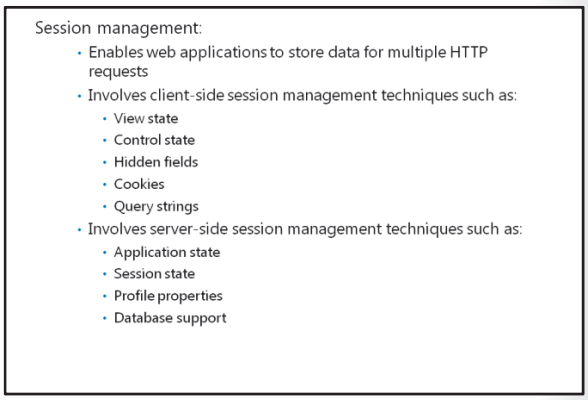
While developing applications, you may want to create functions that require information to be retained across requests. For example, consider an application in which you need to first select a customer and then work on the order relevant to the customer. HTTP is a stateless protocol. Therefore, ASP.NET includes different state management techniques to help store information for multiple HTTP requests. You need to know how to configure and scale state storage mechanisms to support web server farms.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe the benefits of storing state information.
- List the options for storing state information.
- Configure state storage.
- Describe the scalability of state storage mechanisms.
- Store and retrieve state information in an MVC 4 web application.

Why Store State Information?



HTTP is a stateless protocol that does not relate submitted requests to each other. HTTP treats each request as a new request. However, this independence among the requests affects business applications that have information dependencies amongst functions. For example, consider that you can access the Update Address function only after you provide the user logon information. The page that handles the User Logon function is different from the page that handles the Update Address function. Therefore, these functions require two separate HTTP requests. With the default HTTP implementation, the server may lose the result of the logon request, when a user accesses the Update Address page. ASP.NET implements the session management technology to retain dependent information for multiple HTTP requests.

Session management enables web applications to store values for multiple HTTP requests, regardless of whether the request is directed to the same page or multiple pages. The session management process involves two techniques—server-side techniques and client-side techniques, to retain user information pertinent to multiple HTTP requests.

Client-Side Session Management

Client-side session management technologies include:

- Hidden fields
- Cookies
- Query strings

Client-side session management technologies help retain dependent information when a user triggers the HTTP POST action by clicking a button on a webpage. Then, when a page is rendered, the server generates the dependent information.

Note: If you are familiar with ASP.NET Web Forms, you may have used View State and Control State as client-side state storage. These techniques are not available in MVC because they rely on Web Forms server controls.

Server-Side Session Management

Server-side options provide higher security for storing page information than the client-side options because information retained by using client-side options is not visible to the client system. However, server-side options require more web server resources such as memory for storing data. This additional usage of resources can lead to scalability issues if the size of the information to be stored is large.

ASP.NET provides the following server-side session management options:

- Application state
- Session state
- TempData
- Profile properties
- Database support

Additional Reading: For more information about ASP.NET state management recommendations, go to http://go.microsoft.com/fwlink/?LinkID=288986&clcid=0x415

Question: Why do you need to implement session management?

State Storage Options

You can store values for requests, and the subsequent requests related to the previous requests, by using the **TempData** object. By default, the **TempData** object uses the **Session** variable to store information relevant to requests. You can use the **TempData** object to access session information, instead of using the **Session** variable. The **TempData** object includes logic to handle redirection of information to different views.

- You can use the **TempData** object to store information relevant to requests
- Some commonly used state storage options include:
  - The InProc mode
  - The StateServer mode
  - The SQLServer mode
  - The Custom mode
  - The Off mode

You can use the **sessionState** element in the Web.config file, to define the repository that should retain the session state information. You can also use the **sessionState** element to turn off session-state support. The following list describes some of the commonly used session state storage options:

- *The InProc mode*. Stores the session in the ASP.NET process.
- *The StateServer mode*. Stores session information in dedicated services running on the web server or on a dedicated server. It does not store information on Internet Information Services (IIS).

- *The SQLServer mode*. Stores the session information in a SQL Server database.
- *The Off mode*. Deactivates the session state support in your application.

Question: Consider that you have a web server that contains multiple web servers. You want to allow users to access the same state information for all the web servers they access. In this case, which state storage mode should you used?

Configuring State Storage

You can configure state storage to use the StateServer and SQLServer modes.

The StateServer Mode:

1. Run the ASP.NET state service on the server used for storing session information
2. Configure the ASP.NET application for using the StateServer mode
3. Configure StateServer support in your application

The SQLServer Mode:

1. Install the ASP.NET session state database on the SQL Server by using the aspnet_regsql.exe
2. Configure the ASP.NET application for using the SQLServer mode
3. Create the session state database on the SQL Server

The StateServer Mode

You need to run the ASP.NET state service on the server used for storing session information, before using the StateServer mode. The ASP.NET state service is installed as a service when ASP.NET and the .NET Framework are installed as part of Microsoft .NET Framework. The ASP.NET service is installed at the following location: *systemroot*\Microsoft.NET\Framework*versionNumber*\aspnet_state.exe

To configure an ASP.NET application for using the StateServer mode, you should perform the following steps:

- In the Web.config file of the application, set the **mode** attribute of the **sessionState** element to **StateServer**.
- In the Web.config file of the application, set the **stateConnectionString** attribute to **tcpip=<***serverName***>:42424**.

To configure StateServer support in your application, you should perform the following steps:

1. Run the services.msc file.
2. Set the ASP.NET state service **Startup Type** attribute to **automatic**, and right-click the service.
3. Open the Web.config file, identify the **sessionState** element, and then update the code.

Consider that a session state is stored on the **SampleStateServer** remote computer. The following code shows how to configure the **StateServer** mode.

Configuring the State Server Mode

```
<configuration>
  <system.web>
    <sessionState mode="StateServer"
      stateConnectionString="tcpip=SampleStateServer:42424"
      cookieless="false"
      timeout="20"/>
  </system.web>
</configuration>
```

Note: The objects stored in session state must be serializable, if the mode is set to StateServer or SQLServer.

The SQLServer Mode

Before using the **SQLServer** mode, you need to install the ASP.NET session state database on the SQL Server.. You can install the ASP.NET session state database by using the aspnet_regsql.exe tool. To configure an ASP.NET application for using the SQLServer mode, you should perform the following steps:

- In the Web.config file of the application, set the **mode** attribute of the **sessionState** element to **SQLServer**.
- In the Web.config file of the application, set the **sqlConnectionString** attribute to a connection string for your SQL Server database.

To create the session state database on SQL Server, you should perform the following steps:

1. Open the command prompt with administrative privileges.
2. From the <*systemroot**Microsoft.NET**Framework**versionNumber*> folder on your web server, run the Aspnet_regsql.exe tool.
3. Open the Web.config file, identify the **sessionState** element, and then update the code.

Consider that a session state is stored on the **SampleSqlServer** SQL Server. The following code shows how to configure the **SQLServer** mode.

Configuring SQL Server Session State Storage

```
<configuration>
  <system.web>
    <sessionState mode="SQLServer"
      sqlConnectionString="Integrated Security=SSPI;data
        source=SampleSqlServer;" />
  </system.web>
</configuration>
```

You can customize the behavior of the aspnet_regsql.exe tool by using switches such as **-E**, **-S**, and **-d**. You can use these switches to change the name of the default ASPState database.

Additional Reading: For more information about the aspnet_regsql.exe tool, go to: http://go.microsoft.com/fwlink/?LinkID=293693&clcid=0x409

Question: Which tool should you install before using the SQLServer mode?

Scaling State Storage Mechanisms

The StateServer and SQLServer modes help access the state storage on a web server. These modes allow multiple servers to access state storage information. This process distributes the web workload among multiple web servers.

You can implement partitioning of session state to allow multiple state servers or multiple SQL Server databases to handle session state information. You can apply the logic for partitioning, by implementing the **IPartitionResolver** interface in your application. You need to configure the Web.config file to control the session

Partitioning the session state:

- Enables multiple state servers to handle state information
- Enables multiple SQL Server databases to handle state information
- Involves configuring the session management engine
- Involves implementing the **IPartitionResolver** interface

management engine. Configuring the session management engine helps you use the logic to identify the appropriate web server on which you can store information.

The following code shows how to configure the Web.config file, to enable the **IPartitionResolver** interface.

Using the IPartitionResolver Interface

```
<configuration>
    <system.web>
        <sessionState
            mode="SQLServer"
            partitionResolverType=
                "PartitionResolverClass" />
    </system.web>
</configuration>
```

You can use the **IPartitionResolver** interface only when the session state uses the SQLServer or StateServer modes. In the preceding code sample, note that no connection string is specified by using the **sqlConnectionString** or **stateConnectionString** attributes.

Additional Reading: For more details about implementing state partitioning, go to: http://go.microsoft.com/fwlink/?LinkID=288988&clcid=0x417

Question: When should you implement the **IPartitionResolver** interface?

Demonstration: How to Store and Retrieve State Information

In this demonstration, you will see how to:

- Store and retrieve user preferences from session state in controller actions.

Demonstration Steps

1. In the Solution Explorer pane of the **OperasWebSite – Microsoft Visual Studio** window, under **OperasWebSite**, expand **Controllers**, and then click **HomeController.cs**.
2. In the HomeController.cs code window, locate the following code.

```
public ActionResult About()
{
   return View();
}
```

3. Place the mouse cursor at the end of the located code, press Enter twice, and then type the following code.

```
public ContentResult GetBackground()
{
}
```

4. Place the mouse cursor within the **GetBackground** action code block, and then type the following code.

```
string style;
if (Session["BackgroundColor"] != null)
{
}
else
{
}
```

5. Place the mouse cursor within the **if** statement code block you just added, and then type the following code.

```
style = String.Format("background-color: {0};", Session["BackgroundColor"])
```

6. Place the mouse cursor within the **else** statement code block you just added, and then type the following code.

```
style = "background-color: #dc9797;";
```

7. Place the mouse cursor at the end of the **GetBackground** action code block and outside the **if...else** statements, press Enter, and then type the following code.

```
return Content(style);
```

8. Place the mouse cursor outside any action code block but inside the **HomeController** class, and then type the following code.

```
public ActionResult SetBackground(string color)
{
}
```

9. Place the cursor within the **SetBackground** action code block, and then type the following code.

```
Session["BackgroundColor"] = color;
return View("Index");
```

10. In the Solution Explorer pane, expand **Views**, expand **Home**, and then click **Index.cshtml**.
11. In the Index.cshtml code window, place the mouse cursor at the end of the **P** element, press Enter twice, and then type the following code.

```
<p>
   Choose a background color:
</p>
```

12. Place the mouse cursor at the end of the text in the **P** element, press Enter, and then type the following code.

```
@Html.ActionLink("Pink", "SetBackground", new { color = "#dc9797"})
```

13. Place the mouse cursor at the end of the link you just created, press Enter, and then type the following code.

```
@Html.ActionLink("Blue", "SetBackground", new { color = "#82bbf2"})
```

14. In the Solution Explorer pane, expand **Shared**, and then click **_SiteTemplate.cshtml**.
15. In the _SiteTemplate.cshtml code window, locate the following code.

```
<body>
```

16. Replace the located code with the following code.

```
<body style="@Html.Action("GetBackground", "Home")">
```

17. On the **DEBUG** menu of the **OperasWebSite – Microsoft Visual Studio** window, click **Start Debugging**.

18. On the Operas I Have Seen page, click the **Blue** link, and then note that the blue background color has been applied to the home page.
19. On the Operas I Have Seen page, click the **All Operas** link.

Note: If the background color of the page is blue, click the **Refresh** button.

20. On the Index of Operas page, click the **Details** link corresponding to the title, **Cosi Fan Tutte**.
21. On the Cosi Fan Tutte page, note that the blue background color has been applied to the page.

Note: The blue background preference is applied to all pages of the application.

22. On the Opera: Cosi Fan Tutte page, click **Home**.
23. On the Operas I Have Seen page, click the **Pink** link, and then note that the pink background color has been applied to the home page.

Note: If the background color of the page is blue, click the **Refresh** button.

24. On the Operas I Have Seen page, click **All Operas**.
25. On the Index of Operas page, click the **Details** link corresponding to the title, **Cosi Fan Tutte**.
26. On the Cosi Fan Tutte page, note that the pink background color has been applied to the page.

Note: The pink background preference is applied to all pages of the application.

27. In the Windows Internet Explorer window, click the **Close** button.
28. In the **OperasWebSite – Microsoft Visual Studio** window, click the **Close** button.
29. If the message **Do you want to stop debugging?** is displayed, click **Yes**.

Lab: Building a Resilient ASP.NET MVC 4 Web Application

Scenario

The senior developer has asked you to implement the following functionality in your Photo Sharing web application.

- Any visitor of the application, including anonymous users, should be able to mark a photograph as a favorite.
- If a user has marked a favorite, a link should be available to display the favorite photo.
- Favorite photos should be displayed in the slideshow view.

Objectives

After completing this lab, you will be able to:

- Store a setting for an anonymous or authenticated user in session state.
- Check a user preference when rendering an action link.
- Render a webpage by checking state values in the application.

Lab Setup

Estimated Time: 45 minutes

Virtual Machine: **20486B-SEA-DEV11**

User name: **Admin**

Password: **Pa$$w0rd**

Note: In Hyper-V Manager, start the **MSL-TMG1** virtual machine if it is not already running.

Before starting the lab, you need to perform the following step:

- Download **Manage Nuget packages** from the Project menu and note the version number for the jquery.ui package.
- Enable the **Allow NuGet to download missing packages during build** option, by performing the following steps:
  i. On the **TOOLS** menu of the Microsoft Visual Studio window, click **Options**.
  ii. In the navigation pane of the **Options** dialog box, click **Package Manager**.
  iii. Under the Package Restore section, select the **Allow NuGet to download missing packages during build** checkbox, and then click **OK**.

Exercise 1: Creating Favorites Controller Actions

Scenario

You have been asked to build functionality that stores the favorite photos of the visitors in the session state of the web application. After users add photos to their favorites, they will be able to view a slideshow of all the photos they selected as favorites.

In this exercise, you will:

- Create the Favorites Slideshow action.
- Create the Add Favorite action.

The main tasks for this exercise are as follows:

1. Create the Favorites Slideshow action.
2. Create the Add Favorite action.

▶ Task 1: Create the Favorites Slideshow action.

1. Start the virtual machine, and log on with the following credentials:
   - Virtual machine: **20486B-SEA-DEV11**
   - User name: **Admin**
   - Password: **Pa$$w0rd**
2. Open the **PhotoSharingApplication.sln** file from the following location:
   - File location: Allfiles (D:)\Labfiles\Mod12\Starter \PhotoSharingApplication
3. In the PhotoController.cs code file, add a new action by using the following information:
   - Scope: **public**
   - Return type: **ActionResult**
   - Name: **FavoritesSlideShow**
4. In the **FavoritesSlideShow** action, create and instantiate a new enumerable list of **Photo** objects named **favPhotos**.
5. Create a new enumerable list of integers named **favoriteIds**. Set this list to be equal to the **Session["Favorites"]** variable by casting this variable as a list of integers.
6. If **favoriteIds** is null, set **favoriteIds** to be a new enumerable list of integers.
7. Create a new **Photo** object named **currentPhoto**. Do not instantiate the new object.
8. Create a new **foreach** code block that loops through all the integers in the **favoriteIds** list.
9. For each integer in the **favoriteIds** list, obtain the **Photo** object with the right ID value by using the **context.FindPhotoById()** method. Store the object in the **currentPhoto** variable.
10. If the **currentPhoto** variable is not equal to **null**, then add **currentPhoto** to the **favPhotos** list.
11. At the end of the **FavoritesSlideShow** action, return the **SlideShow** view and pass the **favPhotos** list as a model class.
12. Save all the changes.

▶ Task 2: Create the Add Favorite action.

1. Add a new action to the **PhotoController.cs** file by using the following information:
   - Scope: **public**
   - Return type: **ContentResult**
   - Name: **AddFavorite**
   - Parameter: an integer named **PhotoId**
2. Create a new enumerable list of integers named **favoriteIds**. Set this list to be equal to the **Session["Favorites"]** variable, by casting the variable as a list of integers.
3. If **favoriteIds** is null, set **favoriteIds** to be a new enumerable list of integers.
4. Add the **PhotoId** value to the **favoriteIds** list of integers.
5. Set the **Session["Favorites"]** variable to equal the **favoriteIds** variable.

6. Return HTML content by using the following information:
   - Method: **Content()**
   - Content: **The picture has been added to your favorites**
   - Content type: **text/plain**
   - Encoding: **System.Text.Encoding.Default**
7. Save all the changes.

Results: After completing this exercise, you will be able to create controller actions that store values in the session state of the web application, and retrieve values from the same session state.

Exercise 2: Implementing Favorites in Views

Scenario

You have created the necessary controller actions to implement favorite photos. Now, you should implement the user interface components to display a control for adding a favorite. If a user has favorites, you should display a link to the **FavoritesSlideShow** action.

In this exercise, you will:

- Add an AJAX action link in the Photo Display view.
- Add a link and update the site map.

The main tasks for this exercise are as follows:

1. Add an AJAX action link in the Photo Display view.
2. Add a link and update the site map.
3. Test favorites.

▶ Task 1: Add an AJAX action link in the Photo Display view.

1. Open the **Display.cshtml** view for the Photo controller.
2. At the end of the ** ** element, insert a new ** ** element with the ID, **addtofavorites**.
3. In the **DIV** element, include the **Ajax.ActionLink()** helper to render a link to the **AddFavorite** action by using the following information:
   - Text: **Add this photo to your favorites**
   - Action: **AddFavorite**
   - Controller: **Photo**
   - Route values: Pass the **Model.PhotoID** value to the **PhotoId** action parameter.
   - Pass a new **AjaxOptions** object.
   - Update target ID: **addtofavorites**
   - HTTP method: **GET**
   - Insertion mode: **Replace**
4. Save all the changes.

▶ Task 2: Add a link and update the site map.

1. Open the **_MainLayout.cshtml** view for editing.
2. After the breadcrumb trail, add a Razor **@if** statement to ensure that the **Session["Favorites"]** variable is not null.
3. If the **Session["Favorites"]** variable is not null, render a ** ** element with the ID, **favorites-link**.
4. In the **DIV** element, render a link to the **FavoritesSlideShow** action by using the following information:
   - Helper: **Html.ActionLink()**
   - Link text: **Favorite Photos**
   - Action: **FavoritesSlideShow**
   - Controller: **Photo**
5. Save all the changes.
6. Open the **Mvc.sitemap** file.
7. Add a new site map node to the **Mvc.sitemap** file by using the following information:
   - Title: **Favorites**
   - Visibility: **SiteMapPathHelper,!***
   - Controller: **Photo**
   - Action: **FavoritesSlideShow**
8. Save all the changes.

▶ Task 3: Test favorites.

1. Start the web application in debugging mode.
2. Add three photos of your choice to your favorite photos.
3. Browse to the home page and click the **Favorite Photos** link.
4. Stop debugging and close Visual Studio.

Results: After completing this exercise, you will be able to :

Create the user interface components for the favorite photos functionality.

Test the functionality of the user interface components.

Question: In this lab, you stored the list of favorite photos in the session state. While testing, your manager notices that authenticated users lose their favorite photos list whenever they close their browser. Where would you store a list of favorites for each authenticated user so that the list is preserved whenever a user logs on to the web application?

Question: How would you create a view of favorite photos with the card-style presentation users see on the **All Photos** page?

Module Review and Takeaways

Web applications are usually subject to different kinds of attacks. These attacks enable attackers to access sensitive information stored in the database and to perform malicious actions. You can use AntiXSS and request validation, to protect the applications from web attacks. You can also use state management techniques to store information for multiple HTTP requests to help avoid users from typing the same information more than once.

Review Question(s)

Question: Recently, an error occurred in one of applications that you had developed for your company. After performing few tests, you realize that the issue was due to an HTML code that was passed from the user to the server. You want to prevent such issues in the future. What would you consider to do?

Real-world Issues and Scenarios

While implementing web applications, you may want to use a rich format input editor, to enable users to format the input within text boxes. Therefore, you may need to disable request validation, to enable ASP.NET to capture and process user input.

Complex business functions usually involve multiple views. Such functions can pose problems because information must be shared across multiple views. Session state management helps resolve these problems, because it enables retaining information pertinent to multiple views.

Module 13

Using Windows Azure Web Services in ASP.NET MVC 4 Web Applications

Contents:

Module Overview

Developers often place all application logic in the application code. While this practice works for small-scale applications, it may not work for applications with a large number of users. An alternate approach is to make the code modular, by modifying the applications to use web services.

Windows Azure is a cloud-based platform that enables you to host web applications and services. You can use Windows Azure to make your web application more scalable and manageable. To effectively use Windows Azure, you need to know how to write a web service. You also need to know how to call a web service from a web application or from other applications, such as a mobile device application.

Objectives

After completing this module, you will be able to:

- Describe Windows Azure.
- Design Windows Azure services.
- Call Windows Azure services from a web application.

Lesson 1
Introducing Windows Azure

Windows Azure is a Microsoft platform that helps meet specific needs of hosting web applications, such as creating scalable and highly available web applications. Windows Azure enables you to host services, applications, and databases. Windows Azure also helps you create cost-effective and secure web offerings that suit businesses of various types and sizes. You need to know how to host applications, services, and databases on Windows Azure to increase the scalability, flexibility, and availability of your web offerings.

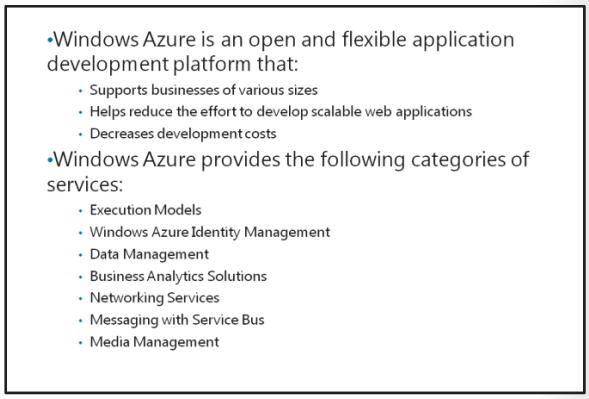
Lesson Objectives

After completing this lesson, you will be able to:

- Describe Windows Azure.
- Describe the hosting services available in Windows Azure.
- Explain how to host web applications in Windows Azure.
- Describe the data storage options offered by Azure Storage and Windows Azure SQL Database.
- Determine when to use Windows Azure.

What Is Windows Azure?

The Windows Azure application platform provides comprehensive services to host applications, files, media, and databases. Windows Azure is a flexible platform that supports businesses of various sizes, helps reduce the effort to develop scalable web applications, and decreases development costs. Microsoft hosts Windows Azure in multiple data centers, across the United States, Europe, and Asia.



Windows Azure provides the following categories of services:

- *Execution Models*. Provides three different execution models—virtual machines (VMs), web applications, and cloud services, for running applications
- *Windows Azure Identity Management*. Includes Windows Azure Active Directory and Access Control services to help manage identities, authentication, and authorization in a web application.
- *Data Management*. Includes services to host and manage relational database and Blob content
- *Business Analytics Solutions*. Includes services that helps perform business information analysis by using SQL Server Reporting Services and Apache Hadoop. Business analytics solutions support all the data stores distributed across multiple Windows Azure data centers.
- *Networking Services*. Includes services that provide networking capabilities to securely connect multiple services in Microsoft data centers.
- *Messaging with Service Bus*. Includes services that enable applications to communicate on Windows Azure.

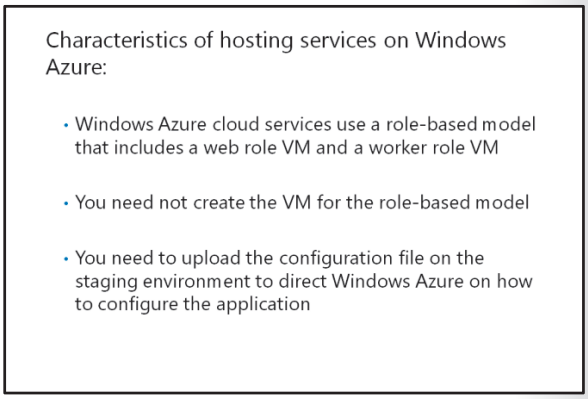
- *Media Management*. Includes services that bring content geographically closer to users and help store large volumes of content outside the web application. Storing content outside the application helps improve the performance of the application.

Question: What are the benefits of hosting applications on Windows Azure?

Benefits of Hosting Services in Windows Azure

Windows Azure enables you to configure application logic to run as services. Services denote a portion of the application code, which performs specific business logic based on user inputs, such as retrieving product pricing information. These services usually do not require:

- Administrator access to Windows.
- Complete access to the user interface of Windows.
- Access to other functionalities that allow interacting with users, which VMs facilitate.



Windows Azure cloud services do not use a dedicated VM. However, Windows Azure cloud services use a role-based model that relies on the VM to host applications. The role-based model includes a web role and a worker role. Each role is a VM that is preconfigured to support automated deployment and monitoring of applications.

While deploying these cloud services, you need not create the VM. You only need to upload a configuration file to direct Windows Azure how to configure the application. You need not upload this configuration file on the VM; you can upload the file on to the staging environment. Then, the Windows Azure management agent copies this configuration information to the production servers.

Windows Azure cloud services do not retain state data in a role, because the VM has no permanent storage. You can either use other services to retain state data or design the application in a stateless manner.

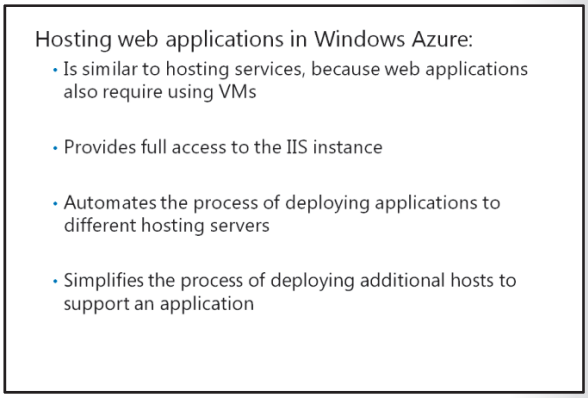
Question: What are the benefits of using Windows Azure cloud services?

Benefits of Hosting Web Applications in Windows Azure

Hosting web applications on Windows Azure is similar to hosting services, because both cases use virtual machines. However, unlike services, web applications provide full access to the IIS instance assigned to your application. You can use a web application to host ASP.NET websites, static content, open source applications, and other file-based web applications.

Hosting web applications in Windows Azure provides the following benefits:

- It automates the process of deploying



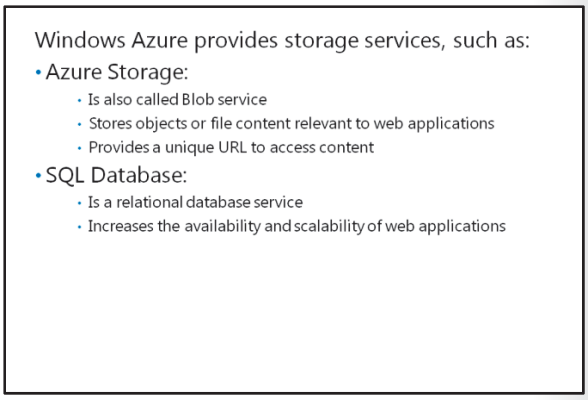
applications to different hosting servers.

- It simplifies the process of scaling up and down as your application usage grows or contracts.

Question: What are the key benefits of using Windows Azure web applications?

Windows Azure Storage Services

Windows Azure includes the following storage services:

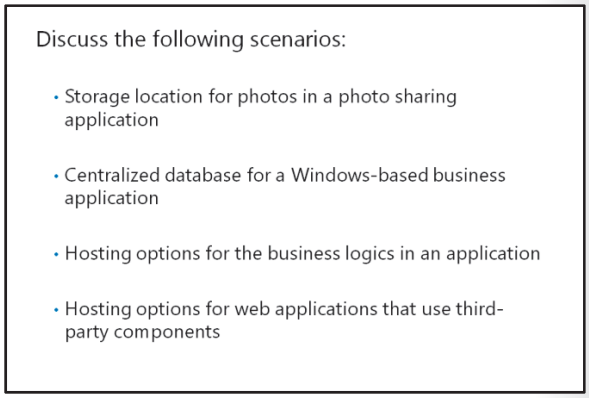


- *Azure Storage*. This is also known as BLOB (Binary Large Object) service. You can use Azure storage to store objects or file content relevant to your web applications. Each Blob includes a unique URL that allows applications to directly access content in the Azure Storage.
- *SQL Database*. This is a relational database service on the Windows Azure platform. You can use SQL Database as the database engine for storing application data in a web application hosted in Azure or on any other Internet server. Other applications, such as desktop packages and mobile apps can connect to the same database over the Internet. When you use SQL Database in Windows Azure, you:
  - Increase the availability and scalability of the data store and any applications built on that data store.
  - Reduce investments in hardware.

Question: What are the benefits of using Windows Azure SQL Database?

Discussion: Windows Azure Scenarios

Consider the following scenarios. In each case, discuss which Azure service will best suit the scenario.



1. You are creating a photo sharing application, and you want to enable each user to discuss the photos with their friends. You need to ensure that the photos in your application do not occupy large disk storage in your web farm. Therefore, you do not want to store the photos, which the users upload, in your application database or server.
2. You are creating a Windows-based business application. This application requires a centralized database that users can access by using the Internet.

3. You are writing business logic for your company, and both public and internal applications want to use this business logic.
4. You are creating an application that uses a third-party component. The third-party component requires full access to the Windows console.

Lesson 2
Designing and Writing Windows Azure Services

Windows Azure cloud services provide a scalable hosting platform for web applications. You can use Windows Azure cloud services to host Windows Communication Foundation (WCF) services. Hosting such services helps transfer key business logic from a web application onto a cloud platform. You need to know about the life cycle of the Windows Azure services to add and modify code during the different stages of application development. You also need to know how to use diagnostic tools in Windows Azure, to identify the problems that occur in your application

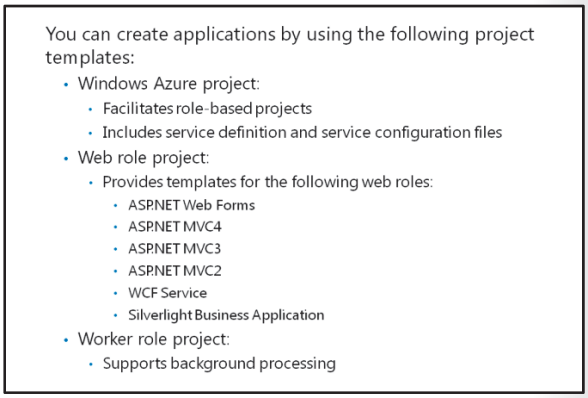
Lesson Objectives

After completing this lesson, you will be able to:

- Describe Windows Azure Visual Studio project templates.
- Write code to create a web service.
- Describe the life cycle of a web service.
- Deploy a web service.
- Debug a Windows Azure web service.

Windows Azure Visual Studio Project Templates

Microsoft Visual Studio provides project templates such as Windows Azure project, web role project, and worker role project, which help create applications and host them on Windows Azure.



Windows Azure Project

The Windows Azure project template facilitates role-based projects. This template includes service definition and service configuration files.

Web Role Project

The web role project provides templates for the following web roles:

- ASP.NET Web Forms
- ASP.NET MVC4
- ASP.NET MVC3
- ASP.NET MVC2
- WCF Service
- Silverlight Business Application

Worker Role Project

A worker role performs background processing by communicating with storage services and other Internet-based services, without interacting with end users. Worker role projects are ideal for long running services and other modular business logic.

Question: Which project template should you use to create Windows Communication Foundation (WCF) services?

Coding a Web Service

Modular programming plays an important role in developing applications, because it enables you to re-use application code. Re-using services enables cross-application sharing of business logic.

To create code for a web service, you need to perform the following steps:

To code a web service:

1. Create a WCF Service Web Role project
2. Rename the interface for the service
3. Define properties and methods in the interface
4. Rename the service class
5. Implement the service logic
6. Test the service

1. Open Microsoft Visual Studio and create a Windows Azure WCF Service web role project.
2. Rename the IService1 interface that the template creates by default. You can use the Visual Studio Refactor menu to rename the interface and update references to it throughout the solution.
3. Modify the interface code to define the interface-level properties and methods. The following code shows a simple service interface that defines a single method. Note the [ServiceContract] and [OperationContract] annotations.

```
    [ServiceContract]
    public interface IMyServiceInterface
    {
        [OperationContract]
        string GetHelloWorld();
    }
```

4. Rename the Service1 class that the template creates by default. Again, you can use the Refactor menu to help with this task.
5. Modify the service code to implement the methods required in the service interface. The following code shows a simple service class that implements the previous simple interface.

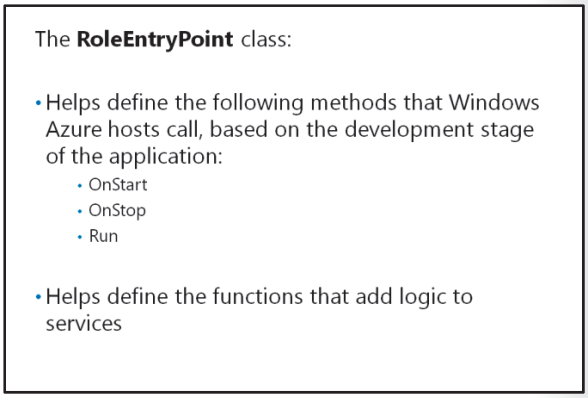
```
public class MyService : IMyServiceInterface
    {
        public string GetHelloWorld()
        {
            return "Hello Azure
        }
    }
```

6. Press F5 to test the services locally by using the Windows Azure emulator. This practice ensures that the services function appropriately, before deploying the services on Windows Azure.

Question: Why should you test the service code before deploying the service on Windows Azure?

The Life Cycle of a Service

The **RoleEntryPoint** class defines the method that Windows Azure hosts call, based on the development stage of the application. The **RoleEntryPoint** class also helps define the functions that add logic to services. WCF services may sometimes inherit the **RoleEntryPoint** class; however, worker roles services mandatorily inherit the **RoleEntryPoint** class.



The **RoleEntryPoint** class helps define the following three life-cycle methods:

- *OnStart*. Windows Azure triggers this method when the application starts.
- *OnStop*. Windows Azure triggers this method when the application stops.
- *Run*. Windows Azure triggers this method to start long running processes.

The **OnStart** and **OnStop** methods return a Boolean value to the Windows Azure host, when a function completes running. If the **OnStart** and **OnStop** methods returns **false**, Windows Azure stops the application from running. You should ensure that the **OnStart** and **OnStop** methods always return **true**, to ensure that your application continues running.

Windows Azure does not trigger the **OnStop** method if a process terminates because of exceptions. Windows Azure treats all uncaught exceptions as unhandled exceptions that cause processes to terminate.

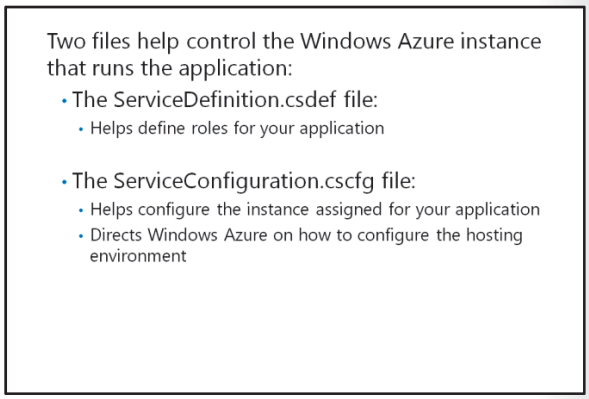
While defining the **RoleEntryPoint** class, you must:

- Call the **Application_Start** method after the **RoleEntryPoint.OnStart** method finishes.
- Call the **Application_End** method before calling the **RoleEntryPoint.OnStop** method.

Question: When should you use the **RoleEntryPoint** class?

Deploying a Web Service

When you create an application project, the system creates two files—ServiceDefinition.csdef and ServiceConfiguration.cscfg. These files control the Windows Azure instance that helps run the application.



The following list describes the ServiceDefinition.csdef and ServiceConfiguration.cscfg files:

- *ServiceDefinition.csdef*. It helps define roles for your application.
- *ServiceConfiguration.cscfg*. It is a hosted file that helps configure the instance assigned for your application. For example, you can use ServiceConfiguration.cscfg to configure the number of instances for your application.

Two copies of ServiceConfiguration.cscfg are available, a cloud version and a local version, which you can use to locally test the services before deploying on Windows Azure. You can manually update the details in the ServiceConfiguration.cscfg file by using the Windows Azure property window.

Additional Reading: For more information about configuring a Windows Azure project, go to: http://go.microsoft.com/fwlink/?LinkID=288989&clcid=0x418

You can use the Package tool in Microsoft Visual Studio to generate the service package:

1. In the Solution Explorer pane, right-click the web service project, and then click **Package**.
2. In the **Service Configuration** box of the **Package Windows Azure Application** dialog box, ensure that **Cloud** is selected, and then, in the **Build Configuration** box, click **Release**.
3. Click **Package**. Visual Studio creates the **.cspkg** file and the **.cscfg** file for the service and displays them in the File Explorer window

Then, you need to upload the services package to the staging environment of the Windows Azure platform by using the management portal. Package files have the .cspkg extension. When you upload the package, you also need to upload the ServiceConfiguration.cscfg file. The ServiceConfiguration.cscfg file directs Windows Azure on how to configure the hosting environment.

After uploading and deploying the package on the Windows Azure platform, you can access the service by using a URL, for example:

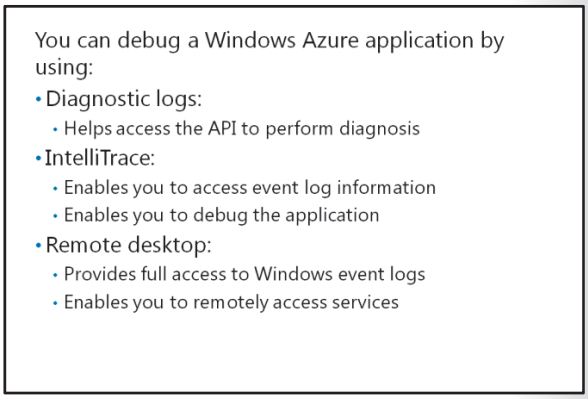
http:// <guid>.cloudapp.net/<service file name>

Question: When should you use the ServiceConfiguration.cscfg file?

Debugging a Windows Azure Web Service

You can debug a Windows Azure application by using the following tools:

- Diagnostic log
- IntelliTrace
- Remote Desktop



Diagnostic **Logs**

To debug Windows Azure web services, you need to include the Diagnostic module in the ServiceDefinition.csdef file. The Diagnostics module helps access the Application Programming Interface (API), to perform diagnosis.

The following code shows how to include the Diagnostic module in the ServiceDefinition.csdef file.

Configuring the Diagnostic Modulec

```
<?xml version="1.0" encoding="utf-8"?>
<ServiceDefinition name="MyHostedService"
xmlns="http://schemas.microsoft.com/ServiceHosting/2008/10/ServiceDefinition">
  <WebRole name="WebRole1">
    <Imports>
      <Import moduleName="Diagnostics" />
    </Imports>
```

```
  </WebRole>
</ServiceDefinition>
```

Running the code in the preceding sample displays diagnostic data.

Additional Reading: For more information about the data types that you can configure your application to collect, go to: http://go.microsoft.com/fwlink/?LinkID=288990&clcid=0x419

To enable your application to monitor additional information, you need to add the following lines of code in the **OnStart** method of your application file.

Add Data Sources

```
var config = DiagnosticMonitor.GetDefaultInitialConfiguration();
config.WindowsEventLog.DataSources.Add("System!*");
DiagnosticMonitor.Start("Microsoft.WindowsAzure.Plugins.Diagnostics.ConnectionString",
config);
```

The lines of code in the preceding sample help:

- Configure the Windows Azure service to capture the Windows Event log. The Windows Azure engine generates this log when the service runs.
- Save the Windows Event log to the log files that are stored in LocalStorage. You can access the log files by using the Windows Azure storage explorer and download the files.

IntelliTrace

IntelliTrace is a feature in Windows Azure that enables you to:

- Access event log information.
- Debug the application.

Note: The virtual machines in Windows Azure always run 64-bit operating systems. If you have Visual Studio 2010 Ultimate installed on a machine that runs a 32-bit operating system, you must install this Intellitrace hotfix. This Intellitrace hotfix helps install the 64-bit IntelliTrace components.

Additional Reading: To install the Intellitrace hotfix, go to: http://go.microsoft.com/fwlink/?LinkID=288991&clcid=0x420

You can enable IntelliTrace, while you deploy or package services. You can also configure services to enable IntelliTrace for specific events. This configuration helps capture detailed information such as Windows log and system configurations.

The following image displays the IntelliTrace Settings dialog box.

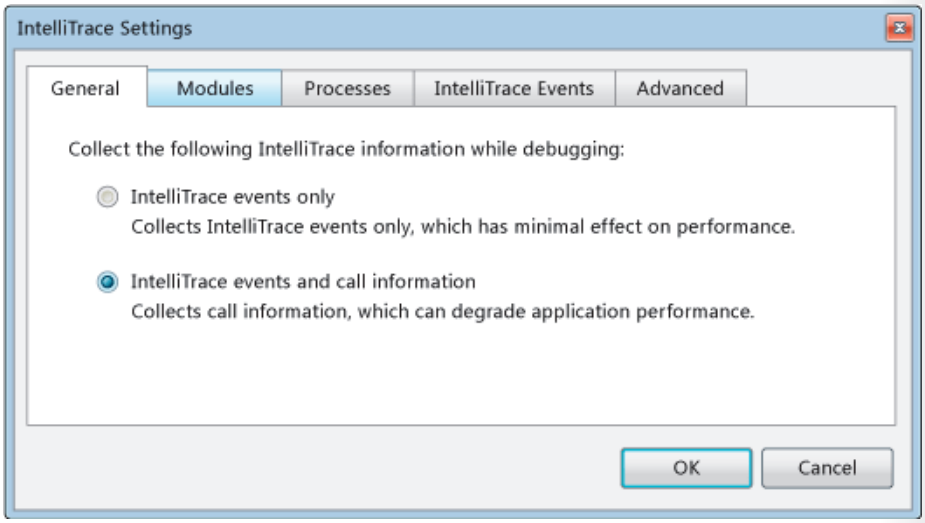


FIGURE 13.1:THE INTELLITRACE SETTINGS DIALOG BOX

IntelliTrace also helps in problem diagnosis. You can simulate some problems that you may encounter while running web services. Then, you can download the IntelliTrace logs from Windows Azure by using the Server Explorer. These logs can help you to access and understand the problems.

The following image displays the IntelliTrace tool in the Server Explorer.

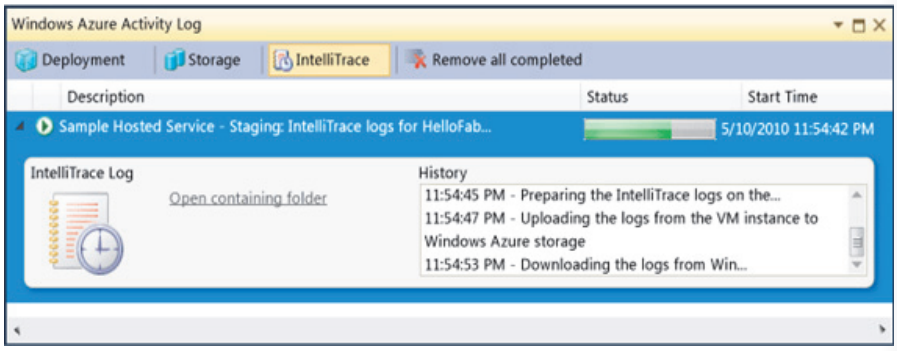


FIGURE 13.2:INTELLITRACE LOGS

Remote Desktop

If you use Windows Azure Cloud Services, you can configure the application to provide remote desktop access to an instance. This configuration provides full access to Windows event logs.

You can enable remote desktop access when you publish your cloud service. To do this, in the Visual Studio Publish wizard, on the **Settings** page, select **Enable Remote Desktop for all roles**. Then, you specify the credentials that a user should use to access the remote desktop console. All instances of your application that run on Windows Azure will use these credentials.

After publishing the application on Windows Azure, you can access the remote desktop by using the management portal.

Question: What is the difference between using diagnostic logs and IntelliTrace?

Lesson 3
Consuming Windows Azure Services in a Web Application

After completing the setup and deployment of the Windows Azure Services, you need to add logic to your application to use the services from ASP.NET MVC application and HTML code. You can call a Windows Azure web service using server-side code or by using client-side code.

Lesson Objectives

After completing this lesson, you will be able to:

- Call a Windows Azure web service by using server-side code.
- Explain how to call a Windows Azure web service by using jQuery and JSON.
- Call a Windows Azure web service by using jQuery.

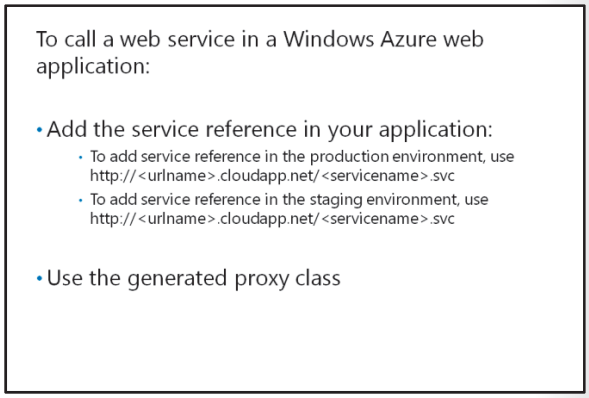
Calling a Windows Azure Service by Using Server-Side Code

You can call web services hosted on Windows Azure in the same manner with which you call standard web services or WCF services. To call a web service, you need to:

1. Add the service reference in your application.
2. Use the generated proxy class.

To add a service reference in an application, you need to use the URL of the service. You can deploy a service in the production environment, by using the following URL format:

http://<urlname>.cloudapp.net/<servicename>.svc



You can deploy a service in the staging environment, by using the following URL format:

http://<guid>.cloudapp.net/<servicename>.svc

To use the WCF service hosted on Windows Azure, from your MVC application, you need perform the following steps:

1. In the **Solution Explorer** pane, right-click your project name, and then click **Add Service Reference**.
2. Enter the URL of the WCF service, and then click **Go**.
3. Define the local namespace of the service, and then click **Add Service Reference**.
4. Add the following code to an MVC controller class to use the generated proxy class and call the web service created earlier in this module.

```
public ActionResult Index()
{
Service1Client client = new Service1Client();
GetHelloWorldRequest request = new GetHelloWorldRequest();
GetHelloWorldResponse response = client.GetHello(request);
ViewBag.Response = response.GetHelloWorldResult;
return View();
}
```

Question: What is the key difference between deploying a service in the production environment and deploying a service in the staging environment?

Calling a Windows Azure Service by Using jQuery

You can enable the jQuery **ajax** function to call WCF services, by configuring the services to accept POST requests in the JavaScript Object Notation (JSON) data format. JSON is a data format that facilitates requests and responses between client and server systems.

The following code shows how to configure services to accept POST requests in the JSON data format.

To enable the jQuery **ajax** function to call WCF services:

- Configure the services to accept POST requests in the JSON data format
- Use the **ajax** function to submit requests to WCF services:
  - Specify parameters such as **type**, **url**, **contentType**, and **dataType**
  - Specify the **ServiceSucceeded** callback function as a parameter

Accepting POST Requests

```
 [ServiceContract]
public interface IService
{
    [OperationContract]
    [WebInvoke(Method = "POST",
     BodyStyle = WebMessageBodyStyle.Wrapped,
     ResponseFormat = WebMessageFormat.Json)]
    string GetHelloWorld();
}
```

You can use the **ajax** function in jQuery to submit requests to WCF services. While adding the **ajax** function, you need to specify parameters such as **type**, **url**, **contentType**, and **dataType**. The following code shows how to use the **ajax** function in jQuery, to call an Azure-hosted WCF service.

Sending Requests to a WCF Service

```
// Function to call WCF  Service
function CallService() {
    $.ajax({
        type: 'POST',
        url: 'http://<guid>.cloudapp.net/service1.svc/GetHelloWorld',
        data: '{}',
        contentType: 'application/json; charset=utf-8',
        dataType: “json”,
        processdata: true,
        success: function(msg) {
            ServiceSucceeded(msg);
        },
        error: ServiceFailed// When Service call fails
    });
}
function ServiceFailed(result) {
}
function ServiceSucceeded(result) {
   result=result.GetHelloWorldResult;
}
```

After successfully calling the WCF service, the code in the preceding sample triggers the **ServiceSucceeded** callback function.

Demonstration: How to Call a Windows Azure Service by Using jQuery

In this demonstration, you will see how to call a web service by using jQuery.

Demonstration Steps

1. In the Solution Explorer pane of the **OperasWebSite – Microsoft Visual Studio** window, expand **OperasWebSite**, expand **Views**, expand **Home**, and then double-click **Index.cshtml**.
2. In the Index.cshtml code window, locate the following code.

```
@Html.ActionLink("operas I've seen.", "Index", "Opera")
</p>
```

3. Place the mouse cursor after the located code, press Enter twice, and then type the following code.

```
<form>
</form>
```

4. Place the mouse cursor in the **FORM** element code block you just created, and then type the following code.

```
<input type="button" value="Get Latest Quote" name="GetLatestQuote"
onclick="callWebService();" />
```

5. Place the mouse cursor at the end of the **INPUT** element, press Enter, and then type the following code.

```
<p id="quote-display"></p>
```

6. In the Index.cshtml code window, locate the following code.

```
</form>
```

7. Place the mouse cursor at the end of the located code, press Enter twice, and then type the following code.

```
<script type="text/javascript">
</script>
```

8. Place the mouse cursor in the **SCRIPT** element code block you just created, and then type the following code.

```
function callWebService() {
}
```

9. Place the mouse cursor in the **callWebService** function code block, and then type the following code.

```
var serviceUrl = '@Url.Content("~/WebServices/QuotesService.asmx")';
```

10. In the **callWebService** function code block, place the mouse cursor at the end of the variable you just created, press Enter, and then type the following code.

```
$.ajax({
   type: "POST",
   url: serviceUrl + "/LatestQuote",
   data: {},
   contentType: "application/json; charset=utf-8",
   dataType: "json",
   success: OnSuccess,
   error: OnError
```

```
});
```

11. Place the mouse cursor at the end of the **callWebService** function code block, but within the **SCRIPT** element, press Enter twice, and then type the following code.

```
function OnSuccess(response) {
}
```

12. Place the mouse cursor in the **OnSuccess** function code block, and then type the following code.

```
$('#quote-display').html(response.d);
```

Note: response.d is the property you use to access JSON data from the server.

13. Place the mouse cursor at the end of the **OnSuccess** function code block, but within the **SCRIPT** element, press Enter twice, and then type the following code.

```
function OnError(response) {
}
```

14. Place the mouse cursor within the **OnError** function, and then type the following code.

```
$('#quote-display').html("Could not obtain the latest quote");
```

15. On the **DEBUG** menu of the **OperasWebSite – Microsoft Visual Studio** window, click **Start Debugging**.
16. On the Operas I Have Seen page, click **Get Latest Quote**.

Note: jQuery calls the web service and displays a quote on the home page. Note that you need not reload the page to display the quote.

17. In the Windows Internet Explorer window, click the **Close** button.
18. In the **OperasWebSite – Microsoft Visual Studio** window, click the **Close** button.

Lab: Using Windows Azure Web Services in ASP.NET MVC 4 Web Applications

Scenario

In the Photo Sharing application, the users have the option to add location information of a photo when they upload it. The senior developer recommends that you should store the location as a longitude and latitude, and an address so that other applications can use the data in mash-ups. You have been asked to create a service, hosted in Windows Azure, which will perform this conversion. You have to call this service from the Photo Upload page in the Photo Sharing application.

Objectives

After completing this lab, you will be able to:

- Install the Windows Azure software development kit (SDK).
- Create a Bing Maps developer account and trial key.
- Write a web service in Visual Studio that is hosted in Windows Azure.
- Call a Windows Azure web service from server-side code in a web application.

Lab Setup

Estimated Time: 75 minutes

Virtual Machine: **20486B-SEA-DEV11**

User name: **Admin**

Password: **Pa$$w0rd**

Note: In Hyper-V Manager, start the **MSL-TMG1** virtual machine if it is not already running.

- Before initiating this lab, you need to perform the following steps:
  - Apply the Snapshot of the virtual machine, **20486B-SEA-DEV11**, that was taken after completing the lab in module 11.
  - Navigate to **Allfiles (D):\Labfiles\Mod 11\Starter\PhotoSharingApplication\PhotoSharingApplication**, and then copy the **web.config** file.
  - Navigate to **Allfiles (D):\Labfiles\Mod 13\Starter\PhotoSharingApplication\PhotoSharingApplication**, and then paste the **web.config** file.
  - Enable the **Allow NuGet to download missing packages during build** option, by performing the following steps:

    i. On the **TOOLS** menu of the Microsoft Visual Studio window, click **Options**.

    ii. In the navigation pane of the **Options** dialog box, click **Package Manager**.

    iii. Under the Package Restore section, select the **Allow NuGet to download missing packages during build** checkbox, and then click **OK**.
- After completing this lab, you need to take the Snapshot of the virtual machine. You need to apply this Snapshot before initiating the labs in module 14, 15 and 16.

Exercise 1: Accessing Windows Azure and Bing Maps

Scenario

To develop a Windows Communication Foundation (WCF) service that is hosted in Windows Azure, you must install the Windows Azure SDK. To resolve address details to latitude and longitude data, you will use the Bing Maps Location API. You will call this API from the WCF service hosted in Windows Azure so that you can re-use your web service from other Adventure Works websites and applications.

In this exercise, you will:

- Install the Windows Azure SDK.
- Create a Bing Maps developer account.
- Create a Bing Maps Key.

The main tasks for this exercise are as follows:

1. Install the Windows Azure SDK.
2. Create a Bing Maps developer account.
3. Create a Bing Maps key.

▶ Task 1: Install the Windows Azure SDK.

1. Start the virtual machine, and log on with the following credentials:
   - Virtual Machine: **20486B-SEA-DEV11**
   - User name: **Admin**
   - Password: **Pa$$w0rd**
2. Navigate to the following webpage:
   - **http://www.microsoft.com/web/downloads/platform.aspx**
3. Download and run the Web Platform Installer.
4. Use the Web Platform Installer to download and install the **Windows Azure SDK for .NET (VS 2012)**.

▶ Task 2: Create a Bing Maps developer account.

1. Navigate to the following webpage:
   - **https://www.bingmapsportal.com**
2. Log on to the Bing Maps Account Center web application by using following the Windows Live account credentials:
   - User name: *<Your Windows Live account name>*
   - Password: *<Your Windows Live account password>*
3. Register a new Bing Maps developer account by using the following information:
   - Account Name: *<Your account name>*
   - Contact Name: *<Your name>*
   - Email Address: *<Your Windows Live account name>*

▶ Task 3: Create a Bing Maps key.

1. Create a new Bing Maps key in the Photo Sharing application by using the following information:

- Application name: **Photo Sharing Application**
- Key type: **Trial**
- Application type: **Public website**

Results: After completing this exercise, you will be able to register an application to access the Bing Maps APIs.

Exercise 2: Creating a WCF Service for Windows Azure

Scenario

You want to create a WCF service to resolve a locality string to a latitude and longitude by looking up the information in the Bing Maps Geocode SOAP service. After creating the WCF service, you need to host the service in Windows Azure by using the Windows Azure Cloud Service project template from the Windows Azure SDK.

In this exercise, you will:

- Add a new Windows Azure Cloud Service project to the web application.
- Create the Location Checker Service interface.
- Add a service reference to the Bing Maps Geocode service.
- Write the Location Checker service.
- Publish the Service in Windows Azure.

The main tasks for this exercise are as follows:

1. Add a new Windows Azure Cloud Service project to the web application.
2. Create the Location Checker Service interface.
3. Add a service reference to the Bing Maps Geocode service.
4. Write the Location Checker service.
5. Publish the Service in Windows Azure.

▶ Task 1: Add a new Windows Azure Cloud Service project to the web application.

1. Open the **PhotoSharingApplication.sln** file from the following location:
   - File location: **Allfiles (D):\Labfiles\Mod13\Starter\PhotoSharingApplication**
2. Add a new project to the **PhotoSharingApplication** web application by using the following information:
   - Template: **Windows Azure Cloud Service**
   - Name: **LocationChecker**
   - .NET Framework 4.5 role: **WCF Service Web Role**
3. Rename the **WCFServiceWebRole1** project as **LocationCheckerWebRole**.
4. Rename the **IService1.cs** file as **ILocationCheckerService.cs**.
5. Rename the **Service1.svc** file as **LocationCheckerService.svc**.
6. Rename the **WCFServiceWebRole1** namespace as **LocationCheckerWebRole** throughout the LocationCheckerService.svc.cs file.

7. Rename the **Service1** class as **LocationCheckerService**.
8. In the **ServiceDefinition.csdef** file, set the value of **vmsize** for the **LocationCheckerWebRole** to **ExtraSmall**.
9. Save all the changes.

▶ Task 2: Create the Location Checker Service interface.

1. Remove all the method declarations from the **ILocationCheckerService** interface.
2. Add a new method declaration to the **ILocationCheckerService** interface by using the following information:
   - Annotation: **[OperationContract]**
   - Return type: **string**
   - Name: **GetLocation**
   - Parameter: a string named **address**
3. Save all the changes.

▶ Task 3: Add a service reference to the Bing Maps Geocode service.

1. Add a service reference to the **LocationCheckerWebRole** project by using the following information:
   - Address: **http://dev.virtualearth.net/webservices/v1/geocodeservice/geocodeservice.svc?wsdl**.
   - Namespace: **GeocodeService**
2. In the **LocationCheckerService.svc** code file, add a **using** statement for the following namespace:
   - **LocationCheckerWebRole.GeocodeService**
3. Save all the changes.

▶ Task 4: Write the Location Checker service.

1. Remove the following public methods from the **LocationCheckerService** class:
   - **GetData**
   - **CompositeType**
2. Add a new method to the **LocationCheckerService** class by using the following information:
   - Scope: **public**
   - Return type: **string**
   - Name: **GetLocation**
   - Parameter: a **string** named **address**
3. In the **GetLocation** method, create a string variable, **results**, and initialize the value of the **results** variable to an empty string.
4. From the Bing Maps Account Center page, copy the key you created for the Photo Sharing Application.
5. In the **GetLocation** method, create a string variable, **key**, and paste the key from the Bing Maps Account Center page as a value of the **key** variable.
6. Open the **Geocoding Code.txt** from the following location:

- **Allfiles (D):\Labfiles\Mod13\Service Code**

7. Copy all the text from the **Geocoding Code.txt** file and paste the text into the **GetLocation** method, immediately after the **key** variable initialization.
8. At the end of the **GetLocation** method, return the string variable **results**.
9. Save all the changes.

▶ Task 5: Publish the Service in Windows Azure.

1. Package the **LocationChecker** project by using the following information:
   - Service configuration: **Cloud**
   - Build configuration: **Debug**
2. In Windows Internet Explorer, log on to the Windows Azure Portal.
3. Create a new cloud service by using the following information:
   - URL: <*Your Windows Live Account Name*>**LocationService**
   - Region: <*Choose a location near you*>

Note: If your Windows Live Account Name includes dots or @ symbols, replace these characters with dashes.

4. Upload a new staging deployment to the new cloud service by using the following information:
   - Name: **LocationChecker**
   - Package location: **Allfiles (D):\Labfiles\Mod13\Starter\PhotoSharingApplication\LocationChecker\bin\Debug\app.publish**
   - Package: **LocationChecker.cspkg**
   - Configuration: **ServiceConfiguration.Cloud.cscfg**
   - Deploy even if one or more roles contain a single instance.

Results: After completing this exercise, you will be able to create a WCF cloud service. You will also be able to publish the WCF cloud service in Windows Azure.

Exercise 3: Calling a Web Service from Controller Action

Scenario

Now that you have created and deployed the Location Checker WCF service in Windows Azure, you can call the service from the Photo Sharing ASP.NET MVC web application. You can also call the service from other .NET code, such as desktop applications, if necessary.

In this exercise, you will use the Location Checker service to add latitude and longitude data to new photos as they are added to the Photo Sharing application.

The main tasks for this exercise are as follows:

1. Add a service reference to the Photo Sharing application.
2. Call the WCF service from the photo create action.
3. Configure the Services Database connection string.

4. Test the Location Checker service.

▶ Task 1: Add a service reference to the Photo Sharing application.

1. Add a new **Service Reference** to the **PhotoSharingApplication** project by using the following information:
   - Address: Copy the **URL** from the CLOUD SERVICE page in the Windows Azure Portal
   - Namespace: **GeocodeService**
2. Add a **using** statement for the following namespace to the **PhotoController.cs** code file:
   - **PhotoSharingApplication.GeocodeService**
3. Save all the changes.

▶ Task 2: Call the WCF service from the photo create action.

1. Add a new method to the **PhotoController** class by using the following information:
   - Scope: **private**
   - Return type: **string**
   - Name: **CheckLocation**
   - Parameter: a string named **location**
2. In the **CheckLocation** method, create a new variable by using the following information:
   - Type: **LocationCheckerServiceClient**
   - Name of the variable: **client**
   - Value of the variable: **Null**
3. In the **CheckLocation** method, create a new string variable, **response**, and initialize the value of the response variable to **Null**.
4. Add a new **try…catch** statement that catches all errors in a variable named **e**.
5. In the **try** block, set the **client** variable to be a **new LocationCheckerServiceClient**.
6. Pass the **location** parameter to the **client.GetLocation** method and store the result in the **response** variable.
7. In the **catch** block, store the **e.Message** property in the **response** variable.
8. At the end of the **CheckLocation** method, return the **response** string.
9. At the start of the **Create** action method for the HTTP POST verb, add an **if** statement that checks that **photo.Location** is not an empty string.
10. In the **if** statement, pass the **photo.Location** property to the **CheckLocation** method and store the result in a new **string** variable named **stringLongLat**.
11. Add an **if** statement to check whether the **stringLongLat** variable starts with the string, **Success**.
12. In the **if** statement, create a new array of characters named **splitChars** and add the single character ":" to the array.
13. Pass the **splitChars** array to the **Split** method of **stringLongLat**, and store the result in a new array of strings named **coordinates**.
14. Set the **photo.Latitude** property to the second string in the **coordinates** array and set the **photo.Longitude** property to the third string in the **coordinates** array.

15. Save all the changes.

▶ Task 3: Configure the Services Database connection string.

1. In the Windows Azure Portal, copy the ADO.NET connection string for the **PhotoSharingAppServices** database to the clipboard.
2. Paste the connection string into the **connectionString** value for the **AzureAppServices** connection string.
3. Save all the changes.

▶ Task 4: Test the Location Checker service.

1. Set **PhotoSharingApplication** as the startup project for the solution.
2. Run the web application in debugging mode, and then log on to the web application by using the following credentials:
   - User name: **David Johnson**
   - Password: **Pa$$w0rd2**
3. Add a new photo to the application by using the following information:
   - Title: **Testing Locations**
   - Photo: **Allfiles (D):\Labfiles\Mod13\Sample Photos\Beach.jpg**
   - Location: **Florence, OR**
4. Browse to the **Home** page and display your new photo to check the Latitude and Longitude properties.
5. Stop debugging and close Visual Studio.

Results: After completing this exercise, you will be able to add a service reference for a Windows Azure WCF service to a .NET Framework application. You will also be able to call a WCF service from an MVC controller action.

Question: What is the advantage of calling the Bing Maps Geocoding service from a WCF service in Windows Azure, instead of calling the Geocoding service directly from the Create action in the MVC web application?

Question: Why is latitude and longitude data useful for photos in the Photo Sharing application?

Module Review and Takeaways

You can upload applications and services on Windows Azure to enable better scalability and re-usability of the application logic. Windows Azure includes different roles that support application needs such as hosting and storage. You can also use cloud services to host WCF services, and use the hosting services available in MVC and HTML applications.

Review Question(s)

Question: Your teammate enquired whether you would be using Windows Azure or self host the server for the application. What should you recommend to your teammate based on the fact that the application loading would be seasonal?

Module 14

Implementing Web APIs in ASP.NET MVC 4 Web Applications

Contents:

Module Overview

Most web applications require integration with external systems such as mobile applications. You need to know how to use Web APIs to promote application interaction with external systems. You can use the Web API to implement Representational State Transfer (REST) services in your application. REST services help reduce application overhead and limit the data that is transmitted between client and server systems. You need to know how to call Web API services by using server-side code, jQuery code, and JSON.NET library to effectively implement REST-style Web APIs in your application.

Objectives

After completing this module, you will be able to:

- Develop a Web API.
- Call a Web API from mobile and web applications.

Lesson 1
Developing a Web API

You need to know how to develop Web API for applications, because Web API facilitates creating APIs for mobile applications, desktop applications, web services, web applications, and other applications. By creating a Web API, you make the information in your web application available for other developers to use in their systems. Each web application has a different functional methodology; this difference can cause interoperability issues in applications. REST services have a lightweight design, and Web API helps implement REST services to solve the interoperability issues. You need to know how to use the different routing methods that ASP.NET provides to implement REST services.

Lesson Objectives

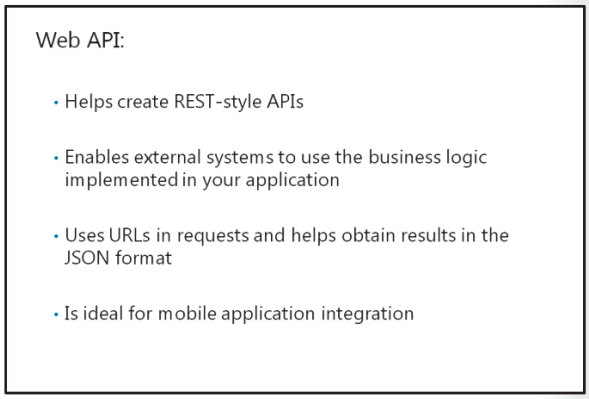
After completing this lesson, you will be able to:

- Describe Web APIs.
- Create a Web API for an MVC 4 web application.
- Describe REST services.
- Describe data return formats.
- Explain how to use routes and controllers to implement REST in Web APIs.
- Explore a Web API by using Internet Explorer as a client.

What Is a Web API?

Web API is a framework that is part of ASP.NET MVC 4 that enables you to build Representational State Transfer (REST)-enabled APIs. REST-enabled APIs help external systems use the business logic implemented in your application to increase the reusability of the application logic. Web API facilitates two-way communication between the client system and the server through tasks such as:

- Instructing an application to perform a specific task
- Reading data values
- Updating data values



Web API enables developers to obtain business information by using REST, without creating complicated XML requests such as Simple Object Access Protocol (SOAP). Web APIs use URLs in requests, thereby eliminating the need for complicated requests. For example, the following URL obtains information for a customer entity with the ID **1**:

http://api.contoso.com/api/customers/1

Web API uses such URLs in requests and obtains results in the JSON format. The following code shows a Web API request response in the JSON format.

A Web API JSON Response

```
 [{"Id":1,"Name":"Tomato soup","Category":"Groceries","Price":1.0},{"Id":2,"Name":
"Yo-yo","Category":"Toys","Price":3.75},{"Id":3,"Name":"Hammer","Category":
```

```
"Hardware","Price":16.99}]
```

REST and Web API enable all kinds of different applications, including mobile device applications, to interact with services. In particular, REST and Web API provide the following benefits for mobile applications:

- They reduce the processing power needed to create complex request messages for data retrieval.
- They enhance the performance of the application by reducing the amount of data exchange between client and server.

Question: What is the key benefit of using REST with Web APIs?

Routing in Web API

When you create a new project by using the Web API template in ASP.NET MVC 4, it includes a default routing rule. This routing rule helps map HTTP requests to the Web API controllers and actions by using HTTP verbs and the request URL. You can make use of a naming convention to map requests to actions, or you can control the behavior of the mapping by using annotations on action methods.

Characteristics of routing in Web API:

- You can use API controller names and a naming convention for actions to route Web API requests
- Alternatively, you can use the following attributes to control the mapping of HTTP requests (HTTP verb+URL) to actions in the controller:
  - The **HttpGet**, **HttpPut**, **HttpPost**, or **HttpDelete** attributes
  - The **AcceptVerbs** attribute
  - The **ActionName** attribute

The Default API Route

Like standard MVC web applications, MVC Web API uses routes to map requests to the right API controller and action. In the Visual Studio project templates, the default API route is defined in the WebApiConfig.cs file in the App_Start folder.

The following code shows the default route.

The Default API Route

```
routes.MapHttpRoute(
    name: "API Default",
    routeTemplate: "api/{controller}/{id}",
    defaults: new { id = RouteParameter.Optional }
);
```

In the preceding code sample, observe that the default route includes the literal path segment **api**. This segment ensures that Web API requests are clearly separate from MVC controller routes, because Web API requests must start with **api**.

The first placeholder variable, *{controller}* helps identify the API controller to forward the request to. As for MVC controllers, Web API appends the word, Controller, to this value to locate the right API controller class. For example, Web API routes a request to the URI, **api/products**, to the controller called, **ProductsController**. As for MVC controllers, the optional placeholder variable, *{id}*, is sent to the action as a parameter.

You can also define your own API routes in the same manner as you do for MVC routes. Observe, however, that Web API routes can handle requests from many types of client systems, including mobile device applications, desktop applications, web applications, and web services. MVC routes only handle web browser requests.

Using the Action Naming Convention

The default Web API route does not include a placeholder variable for the action name. This is because Web API uses the HTTP verb and a naming convention to route requests to the right action within a given controller.

Clients can make HTTP requests with one of four standard verbs: **GET**, **POST**, **PUT**, and **DELETE**. Other verbs are possible. Web API looks for an action whose name begins with the requested HTTP verb. For example, if the client sends a **DELETE** request to the URI **api/products/23**, Web API looks for a controller called **ProductsController**. Within this controller, it locates an action whose name begins with **Delete**. According to the default route, the segment **23** is the **{id}** parameter. If there is more than one action whose name begins with **Delete**, Web API chooses the action that accepts a parameter called **id**.

The HttpGet, HttpPut, HttpPost, and HttpDelete Attributes

You can use the **HttpGet**, **HttpPut**, **HttpPost**, or **HttpDelete** attributes in your controller action to override the action naming convention. You can also use these verbs to specify that a function is mapped to a specific HTTP verb. The following table describes how the HTTP attributes map to the HTTP verbs.

Attribute	HTTP Verb
HttpGet	GET
HttpPut	PUT
HttpPost	POST
HttpDelete	DELETE

The following code illustrates the use of the HttpGet attribute on the FindProduct action.

Specifying the HTTP Verb

```
public class ProductsController : ApiController
{
    [HttpGet]
    public Product FindProduct(id) {}
}
```

In the preceding code sample, observe that the HTTP attributes only allow mapping of one HTTP verb to an action in the controller.

The AcceptVerbs Attribute

The use of the **AcceptVerbs** attribute allows you to specify custom HTTP Verbs or multiple HTTP verbs to the same actions in the controller.

The following code shows the use of the **AcceptVerbs** attribute to map specific HTTP verbs to the action.

The AcceptVerbs Attribute

```
public class ProductsController : ApiController
{
    [AcceptVerbs("GET", "HEAD")]
    public Product FindProduct(id) { }
    [AcceptVerbs("MKCOL")]
    public void MakeCollection() { }
}
```

The ActionName attribute

By default, the action name is the name of the action you specify in the controller. You can use the **ActionName** attribute to specify the action name to be used in the routing.

The following code shows how to map an action to an HTTP request by using a custom action name.

Mapping an Action

```
public class ProductsController : ApiController
{
    [HttpGet]
    [ActionName("Thumbnail")]
    public HttpResponseMessage GetThumbnailImage(int id);
    [HttpPost]
    [ActionName("Thumbnail")]
    public void AddThumbnailImage(int id);
}
```

Question: What is the purpose of using the HTTP attributes?

Creating a Web API for an MVC 4 Web Application

MVC4 provides a Web API project template that helps implement Web API in a project.

The following image shows the list of project templates available in Microsoft Visual Studio.

To create a Web API for an MVC4 application:

1. Implement a Web API template in your project:
   1. In the **New Project** dialog box, click **ASP.NET MVC 4 Web Application**
   2. In the **Select a Template** box of the **New ASP.NET MVC 4 Project** dialog box, click **Web API**
2. Add an MVC API controller class to the project:
   - Hosts application code for handling requests
   - Derives from the ApiController base class
3. Add action methods to the controller class

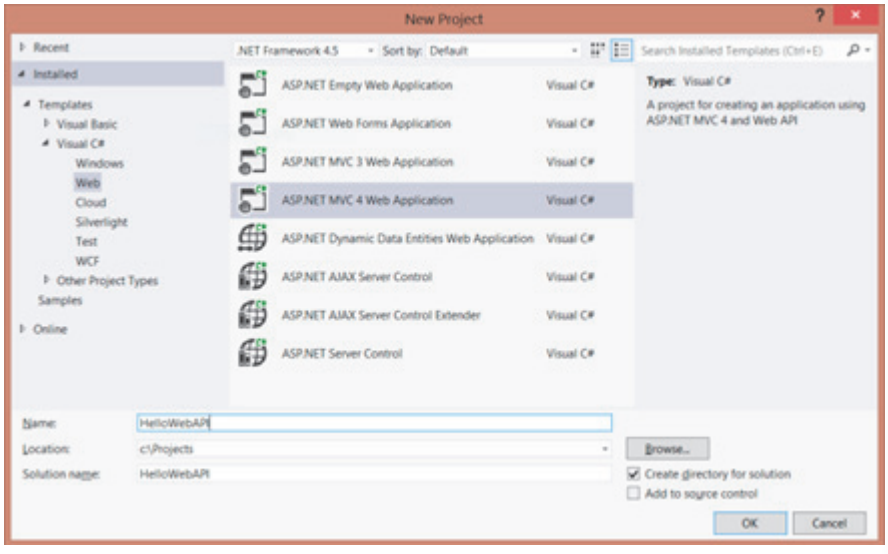


FIGURE 14.1:THE NEW PROJECT DIALOG BOX

To implement a Web API template in your project, you need to perform the following steps:

1. In the **New Project** dialog box, click **ASP.NET MVC 4 Web Application**.
2. In the **Select a Template** box of the **New ASP.NET MVC 4 Project** dialog box, click **Web API**.

The following image shows the New ASP.NET MVC 4 Project dialog box.

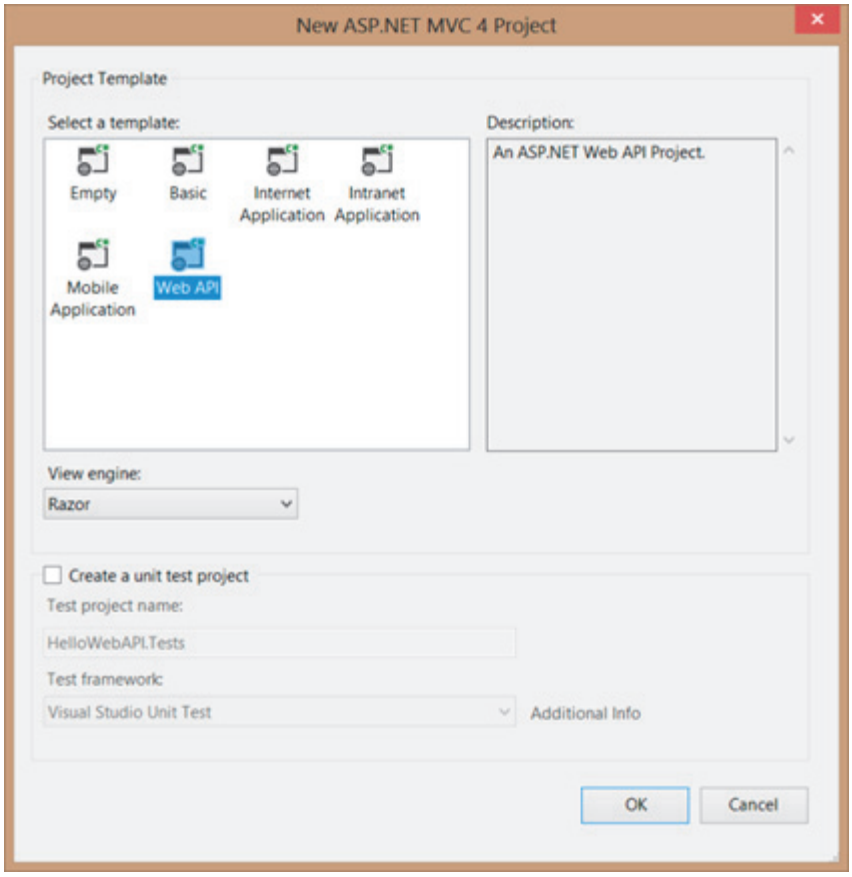


FIGURE 14.2:THE NEW ASP.NET MVC 4 PROJECT DIALOG BOX

After selecting the Web API template, you need to add a new MVC controller class that derives from **ApiController**. The API controller class hosts application code for handling Web API requests. ASP.NET MVC4 engine maps the URL together with the HTTP verb and the controller or the action function of a controller, in the following format.

```
<http verb> http://<hostname>/api/<entity name>/<parameters>
```

The HTTP verb communicates to the Web API about the operations that it should perform; whereas, the rest is to communicate which entity and operations to perform on. Therefore, HTTP plays an important role. For example, consider the following API controller class.

```
public class ProductsController : ApiController
{
        public IEnumerable<Product> GetAllProducts()
        {
        }
        public Product GetProductById(int id)
        {
        }
}
```

In the preceding code sample, note the controller classes **GetAllProducts** and **GetProductById**. The **GetAllProducts** controller class helps obtain a full list of products from the database. You can map the following URL and HTTP verb with the **GetAllProducts** controller class.

```
GET /api/products
```

The **GetProductById** controller class helps obtain a specific product by using the ID detail. You can map the following URL with the **GetProductById** controller class.

```
GET /api/products/id
```

Question: What is the syntax that the ASP.NET MVC engine uses for mapping controllers and action functions?

RESTful Services

REST uses URLs and HTTP verbs to uniquely identify the entity that it operates on and the action that it performs. REST helps retrieve business information from the server. However, in addition to data retrieval, business applications perform more tasks such as creating, updating, and deleting information on the database. Web API and REST facilitate handling such additional tasks. They use the HTTP method to identify the operation that the application needs to perform.

The following table provides information on some HTTP methods that Web API and REST use.

Characteristics of a REST Service:

- Can be called to retrieve business information from the server
- Can create, update, and delete information in a database through HTTP operations
- Uses URLs to uniquely identify the entity that it operates on
- Uses HTTP verbs to identify the operation that the application needs to perform. The HTTP verbs include:
  - **GET**
  - **POST**
  - **PUT**
  - **DELETE**

HTTP Verb	Description
GET	Use this method with the following URL to obtain a list of all customers. `/api/customers`
GET	Use this method with the following URL to obtain a customer by using the ID detail. `/api/customers/id`
GET	Use this method with the following URL to obtain customers by using the category detail. `/api/customers?country=country`
POST	Use this method with the following URL to create a customer record. `/api/customers`

HTTP Verb	Description
PUT	Use this method with the following URL to update a customer record. `/api/customers/id`
DELETE	Use this method with the following URL to delete a customer record. `/api/customers/id`

Web API allows developers to use a strong typed model for developers to manipulate HTTP request. The following code shows how to use the POST, PUT, and DELETE methods for the create, update, and delete requests to handle the creation, retrieval, updation, and deletion (CRUD) of the customer records.

CRUD Operations

```
public HttpResponseMessage PostCustomer(Customer item)
{
}
public void PutCustomer(int id, Customer item)
{
}
public void DeleteProduct(int id)
{
}
```

Question: What is the mandatory requirement of create and update requests?

Data Return Formats

When a client makes a request to a Web API controller, the controller action often returns some data. For GET requests, for example, this data might be all the properties of a specific product or all the properties of all the products. Web API can return this data in one of two formats: JavaScript Object Notation (JSON) or XML.

- Web API can return data in JSON or XML formats
- Web API uses the media formatter to:
  - Format or serialize the information that a Web API REST service returns
  - Control the media type in the HTTP header
  - Format all content that the server renders to client systems
- Media formatter classes inherit from the **MediaTypeFormatter** class and the **BufferedMediaTypeFormatter** class

JSON and XML Data Formats

Both JSON and XML are text formats that represent information as strings. You can also use JSON outside the JavaScript code.

The following code is a simple example of JSON.

A JSON Response

```
{"Name":"Albert","Age":29,"Height":145,"Skills":["Programming", "Technical Writing"]}
```

You can represent the same data in the previous code sample, by using XML as shown in the following code.

An XML Response

```
<Employee Name="Albert" Age="29" Height="145">
   <Skills>
      <Skill Name="Programming" />
      <Skill Name="Technical Writing" />
   </Skills>
</Employee>
```

When a client makes a request, the client can specify the data format for the response. If the data format is not specified, Web API formats data as JSON by default.

Media Formatters

Web API uses a media formatter to format or serialize the information that a Web API REST service returns. Web applications usually use the JSON format to format the data that functions return. However, you can alternatively use the XML media formatter or add a custom media formatter to control the data returned. Media formatters format the content that the server renders to the client systems.

For example, consider that you want Web API to return CSV files. In this case, you need to create a custom media formatter, to create as output data files in the CSV format, instead of XML files in the JSON format.

The following code shows to add a new class in your project, to create a custom media formatter.

A Custom Media Formatter

```
public class CsvFormatter : BufferedMediaTypeFormatter
{
}
```

Media formatter classes inherit from the following classes:

- *MediaTypeFormatter*. This is the fundamental class for all formatter classes.
- *BufferedMediaTypeFormatter*. This is an extended class of formatter classes, which provides support for buffering and asynchronous operations.

Additional Reading: For more information on creating media formatters, go to: http://go.microsoft.com/fwlink/?LinkID=288992&clcid=0x421

Question: Why should you use a media formatter for Web API REST services?

Using Routes and Controllers in Web APIs

ASP.NET MVC4 uses a route table to map a URL and a controller. When you create a project, ASP.NET adds a default route by using the Web API template. This default route helps support the operations of the REST-style Web APIs.

The following code shows the default route.

The Default API Route

```
routes.MapHttpRoute(
    name: "API Default",
    routeTemplate:
"api/{controller}/{id}",
    defaults: new { id =
```

Routing in ASP.NET MVC4 applications involves the following:

- ASP.NET adds a default route to:
  - Map a URL and a controller
  - Support the operations of the REST-style Web APIs
- You can modify the default route to include multiple actions in the same HTTP method
- You can use the **WebApiConfig** class to:
  - Modify the routing
  - Enable multiple versions of API to coexist in the same project

```
RouteParameter.Optional }
);
```

Consider that you want to include multiple actions, such as creating customers with XML and JSON, in the same HTTP method. In this case, you cannot use the default route because it requires a new request for each HTTP method and URL combination. Therefore, you need to update the routing by modifying the Route Map in the **WebApiConfig** class.

The following code shows how to update the routing, to support multiple operations in the same HTTP method.

Supporting Multiple Operations

```
routes.MapHttpRoute(
    name: "ActionApi",
    routeTemplate: "api/{controller}/{action}/{id}",
    defaults: new { id = RouteParameter.Optional }
);
```

You can use the **WebApiConfig** class to enable multiple versions of API to coexist in the same project. For example, you can include **/api/v1/{controller}** as Version One of your API and include **/api/v2/{controller}** as a new version or Version Two of the API.

You may want to include supporting functions in the controller class, and hide the supporting functions from the REST functions. You can eliminate the exposure of the function to the REST interface by adding the **NoAction** attribute to the action function.

The following code shows how to add the **NoAction** attribute to the action function.

Using NoAction

```
 [NonAction]
public string GetPrivateData()
{
}
```

By default, Web API exposes all public methods as REST services. You can prevent this by making the function **private**, but this action prevents application code in the same project from accessing the function. Therefore, you can use the **NonAction** attribute for functions that need to be in public, but do not need to be exposed in REST.

Question: What is the key benefit of using the routing map?

Demonstration: How to Explore a Web API by Using Internet Explorer

In this demonstration, you will see how to:

- Create a simple Web API for an existing ASP.NET MVC 4 web application.
- Download and examine the JSON files that the Web API generates by using Internet Explorer.

Demonstration Steps

1. In the Solution Explorer pane, expand **OperasWebSite**.
2. In the Solution Explorer pane, under OperasWebSite, right-click **Controllers**, point to **Add**, and then click **Controller**.
3. In the **Controller name** box of the **Add Controller** dialog box, type **OperasApiController**, in the **Template** box, click **Empty API Controller**, and then click **Add**

4. In the OperasApiController.cs code window, locate the following code.

```
using System.Web.Http;
```

5. Place the mouse cursor at the end of the located code, press Enter, and then type the following code.

```
using OperasWebSite.Models;
```

6. Place the mouse cursor in the **OperasApiController** class code block, press Enter, and then type the following code.

```
private OperasDB contextDB = new OperasDB();
```

7. Place the mouse cursor at the end of the code you just typed, press Enter twice, and then type the following code.

```
public IEnumerable<Opera> GetOperas()
{
}
```

8. Place the mouse cursor in the **GetOperas** action code block, and then type the following code.

```
return contextDB.Operas.AsEnumerable();
```

9. Place the mouse cursor at the end of the **GetOperas** action code block, press Enter twice, and then type the following code.

```
public Opera GetOperas(int id)
{
}
```

10. Place the mouse cursor in the **GetOperas** action code block you just created, and then type the following code.

```
Opera opera = contextDB.Operas.Find(id);
```

11. Place the mouse cursor at the end of the code you just entered, press Enter, and then type the following code.

```
if (opera == null)
{
   throw new HttpResponseException(HttpStatusCode.NotFound);
}
```

12. Place the mouse cursor at the end of the code you just entered, press Enter, and then type the following code.

```
return opera;
```

13. On the **FILE** menu of the **OperasWebSite – Microsoft Visual Studio** window, click **Save All**.
14. On the **DEBUG** menu of the **OperasWebSite – Microsoft Visual Studio** window, click **Start Debugging**.
15. In the Address bar of the Windows Internet Explorer window, type **http://localhost:<***yourPortNumber*>**/api/OperasApi**, and then click **Go to**.
16. In the Navigation bar, click **Open**.

17. If the "How do you want to open this type of file (.json)?" message is displayed, click **More options**, and then click **Microsoft Visual Studio Version Selector**.
18. On the **EIDT** menu of the **OperasApi.json – Microsoft Visual Studio** window, point to **Find and Replace**, and then click **Quick Find**.
19. In the **Search Item** box of the **Quick Find** dialog box, type **Rigoletto**, and then click **Find Next**.
20. In the **Microsoft Visual Studio** dialog box, click **OK**.
21. In the Quick Find dialog box, click the **Close** button.

Note: Visual Studio finds the JSON data for the **Rigoletto** opera. Note that this is just one entry in the JSON data, which includes all operas in the web application.

22. In the **OperasApi.json – Microsoft Visual Studio** window, click the **Close** button
23. In the Address bar of the Windows Internet Explorer window, type **http://localhost:<***yourPortNumber***>/api/OperasApi/3**, and then click **Go to**.
24. In the Navigation bar, click **Open**.
25. If the "How do you want to open this type of file (.json)?" message is displayed, click **More options**, and then click **Microsoft Visual Studio Version Selector**.
26. In the **3.json - Microsoft Visual Studio** window, note that only the information relating to the **Nixon in China** opera is displayed.

Note: The value for the **OperasID** parameter corresponding to the **Nixon in China** opera is **3**.

27. In the **3.json - Microsoft Visual Studio** window, click the **Close** button.
28. In the Windows Internet Explorer window, click the **Close** button.
29. In the **OperasWebSite – Microsoft Visual Studio** window, click the **Close** button.

Lesson 2

Calling a Web API from Mobile and Web Applications

After you complete the development of the Web API services, you need to create the client applications to call these services. Calling Web API services is different from calling WCF services. However, the methods that you need to use to call these services are similar, regardless of the platform. You need to know how to call Web API services by using server-side code, jQuery code and JSON.NET library, to effectively implement Web API services in most application platforms.

Lesson Objectives

After completing this lesson, you will be able to:

- Call Web APIs by using server-side code.
- Call Web APIs by using jQuery.
- Call Web APIs by using Windows Phone applications.

Calling Web APIs by Using Server-Side Code

You can call REST-style services by using ASP.NET server-side code. You can use the **HttpWebRequest** class to create a manual HTTP request to the REST services. ASP.NET provides a .NET library that you can use in web applications to call REST-enabled Web API services from the .NET server. To use the .NET library, you need to install the **Microsoft.AspNet.WebApi.Client** NuGet package. This NuGet package provides access to the **HttpClient** class. The **HttpClient** class simplifies interacting with Web APIs, because it reduces coding efforts.

To call Web APIs by using server-side code:

- Install the **Microsoft.AspNet.WebApi.Client** NuGet package
- Add code to initialize the **HttpClient** class
- Add code to create requests by using **GetAsync** and **ReadAsAsync**

After installing the NuGet package, you need to initialize the **HttpClient** class. The following code illustrates how to initialize the **HttpClient** class.

Initializing the HttpClient Class

```
HttpClient client = new HttpClient();
client.BaseAddress = new Uri("http://localhost/");
client.DefaultRequestHeaders.Accept.Add(new
MediaTypeWithQualityHeaderValue("application/json"));
```

The last line of code in the preceding code sample informs the client system about the media type that the client system should use. The default media type that applications use is **application/json**. However, applications can use any other media type, based on the media type that the REST-style services support.

The following code shows how to call Web API REST services by using server-side code.

Calling the API from Server-Side Code

```
HttpResponseMessage response = client.GetAsync("api/customers").Result;
if (response.IsSuccessStatusCode)
{
    var products = response.Content.ReadAsAsync<IEnumerable<Customer>>().Result;
}
```

```
else
{
    Console.WriteLine("{0} ({1})", (int)response.StatusCode, response.ReasonPhrase);
}
```

After running the code in the preceding code sample, you need to define a data model that aligns itself with the one used by the Web API service to enable the .NET library to:

- Process the results of the server-side code.
- Return results as .NET objects for the application to use.

Then, you can use the **GetAsync** and **ReadAsAsync** methods to:

- Create requests to Web API REST services.
- Parse the content into .NET objects.

The **PostAsJsonAsync** function uses the HTTP POST method to call Web API services that support the POST method.

Question: What is the benefit of using the **Microsoft.AspNet.WebApi.Client** NuGet package?

Calling Web APIs by Using jQuery Code

You can call Web API services in the same manner as you call other services that use technologies such as WCF. You can also call Web API services by using the jQuery **ajax** function.

The following code shows how to call a Web API service by using the jQuery **ajax** function.

Using jQuery to call Web API services provides you the following options:

- You can use the jQuery **ajax** function to call Web API services
- You can set the **dataType** parameter of the **ajax** function to **json**
- You can use **JSON.stringify()** in the **data** parameter of the **ajax** function to serialize the JavaScript objects into JSON objects

Using the jQuery ajax Function

```
$.ajax({
            url:
'http://localhost/api/customers/,
            type: 'GET',
            dataType: 'json',
            success: function (data) {
            },
            error: function (e) {
            }
        });
```

In the preceding code sample, observe the **dataType** parameter of the **ajax** function. You should set this parameter to **json** or another data type that the Web API service supports. Most applications use JSON because it is light weight. The **ajax** function has built-in functionalities that parse JSON results for the ease of developers.

You can use **JSON.stringify()** in the **data** parameter of the **ajax** function to serialize the JavaScript objects into JSON objects for sending to the Web API method. The following code shows how to use **JSON.stringify()** in the **data** parameter of the **ajax** function.

Using the stringify function

```
var customer = {
            ID:'1',
            CustName: 'customer 1'
        };
```

```
        $.ajax({
            url: 'http://localhost/api/customer',
            type: 'POST',
            data:JSON.stringify(customer),
            contentType: "application/json;charset=utf-8",
            success: function (data) {
            },
            error: function (x) {
            }
        });
```

Question: What is the benefit of using **JSON.stringify()** in the **ajax** function?

Calling Web APIs Using Windows Phone Applications

You can use NuGet packages to call Web API REST services in .NET applications. However, in Windows Phone web applications, you need to use the JSON.NET library to call Web API REST services. The JSON.NET library is more light weight and it enhances the performance of Windows Phone applications. You can download the library from http://go.microsoft.com/fwlink/?LinkID=288993&clcid=0x422

To call Web APIs by using the JSON.NET library:

- Download the JSON.NET library from http://json.codeplex.com
- Add code to create requests by using the **WebClient** class
- Add the **DownloadStringCompleted** event handler to deserialize the results into .NET objects

The JSON.NET library contains a set of .NET classes that help create JSON requests to any REST-style APIs. To use the JSON.NET library, you need to download it and place the JSON.NET assembly in the /WindowsPhone folder of the /bin folder.

The following code shows how to call an HTTP request by using the **WebClient** class.

Using the WebClient Class

```
WebClient webClient = new WebClient();
Uri uri = new Uri("http://localhost/api/customer/");
webClient.DownloadStringCompleted += new
    DownloadStringCompletedEventHandler(webClient_DownloadStringCompleted);
webClient.DownloadStringAsync(uri);
```

When you make a request by using the **WebClient** class, the JSON.NET library processes the request and produces results from the WebClient class to return .NET objects. When you call Web APIs by using the library, you need to add the **DownloadStringCompleted** event handler. This event handler helps place logic to handle the data that Web APIs return.

In the **DownloadStringCompleted** event handler, you should add the code similar to the following.

Coding the Event Handler

```
using Newtonsoft.Json;
/// Lines skipped
List<Customer> customers = JsonConvert.DeserializeObject<List<Customer>>(e.Result);
      foreach (Customer em in customers)
      {
      }
```

The code in the preceding sample helps enable the JSON.NET library to deserialize the results into .NET objects.

Question: What is the key benefit of using the JSON.NET library?

Lab: Implementing APIs in ASP.NET MVC 4 Web Applications

Scenario

Your manager wants to ensure that the photos and information stored in the Photo Sharing application can be integrated with other data in web mash-ups, mobile applications, and other locations. To re-use such data, while maintaining security, you need to implement a RESTful Web API for the application. You will use this Web API to display the locations of photos on a Bing Maps page.

Objectives

After completing this lab, you will be able to:

- Create a Web API by using the new features of ASP.NET MVC 4.
- Add routes and controllers to an application to handle REST requests.
- Call a REST Web API from jQuery client-side code.

Lab Setup

Estimated Time: 60 minutes

Virtual Machine: **20486B-SEA-DEV11**

User name: **Admin**

Password: **Pa$$w0rd**

Note: In Hyper-V Manager, start the **MSL-TMG1** virtual machine if it is not already running.

Before initiating this lab, perform the following steps:

a. Apply the Snapshot of the virtual machine, **20486B-SEA-DEV11**, that was taken after completing the lab in module 13.

b. Navigate to **Allfiles (D):\Labfiles\Mod 13\Starter\PhotoSharingApplication\PhotoSharingApplication**, and then copy the **web.config** file.

c. Navigate to **Allfiles (D):\Labfiles\Mod 14\Starter\PhotoSharingApplication\PhotoSharingApplication**, and then paste the **web.config** file.

d. Enable the **Allow NuGet to download missing packages during build** option, by performing the following steps:

   i. On the **TOOLS** menu of the Microsoft Visual Studio window, click **Options**.

   ii. In the navigation pane of the **Options** dialog box, click **Package Manager**.

   iii. Under the Package Restore section, select the **Allow NuGet to download missing packages during build** checkbox, and then click **OK**.

Exercise 1: Adding a Web API to the Photo Sharing Application

Scenario

You have been asked to implement a Web API for the Photo Sharing application to ensure that photos can be used in third-party websites, mobile device applications, and other applications.

In this exercise, you will

- Add a Web API controller for the Photo model class.

- Configure formatters and routes to support the Web API.
- Test the API by using Internet Explorer.

The main tasks for this exercise are as follows:

1. Add a Photo API controller.
2. Configure API routes.
3. Configure media-type formatters.
4. Test the Web API with Internet Explorer.

▶ Task 1: Add a Photo API controller.

1. Start the virtual machine and log on with the following credentials:
   - Virtual machine: **20486B-SEA-DEV11**
   - User name: **Admin**
   - Password: **Pa$$w0rd**
2. Open the **PhotoSharingApplication** solution from the following location:
   - File location: **Allfiles (D):\Labfiles\Mod14\Starter\PhotoSharingApplication**
3. Add a new API controller to the **PhotoSharingApplication** project by using the following information:
   - Name: **PhotoApiController**
   - Template: **Empty API controller**
4. Add a **using** statement for the following namespace to **PhotoApiController.cs**:
   - **PhotoSharingApplication.Models**
5. Add a new variable to the **PhotoApiController** class by using the following information:
   - Scope: **private**
   - Type: **IPhotoSharingContext**
   - Name: **context**
   - Initial value: a new **PhotoSharingContext** object
6. Add a new action to the **PhotoApiController** by using the following information:
   - Scope: **public**
   - Return type: **IEnumerable<Photo>**
   - Name: **GetAllPhotos**
   - Parameters: none
7. In the **GetAllPhotos** action, return the **context.Photos** collection as an enumerable object.
8. Add a new action to the **PhotoApiController** by using the following information:
   - Scope: **public**
   - Return type: **Photo**
   - Name: **GetPhotoById**
   - Parameters: an integer named **id**

9. In the **GetPhotoById** action, pass the **id** parameter to the **context.FindPhotoById()** method. Store the returned **Photo** object in a variable named **photo**.
10. If the **photo** variable is **null**, throw a new **HttpResponseException** and pass the **HttpStatusCode.NotFound** value.
11. At the end of the **GetPhotoById** action, return the **photo** object.
12. Add a new action to the **PhotoApiController** by using the following information:
    - Scope: **public**
    - Return type: **Photo**
    - Name: **GetPhotoByTitle**
    - Parameters: a string named **title**
13. In the **GetPhotoByTitle** action, pass the **title** parameter to the **context.FindPhotoByTitle()** method. Store the returned **Photo** object in a variable named **photo**.
14. If the **photo** variable is **null**, throw a new **HttpResponseException** and pass the **HttpStatusCode.NotFound** value.
15. At the end of the **GetPhotoByTitle** action, return the **photo** object.
16. Save all the changes.

▶ Task 2: Configure API routes.

1. In the WebApiConfig.cs code file, in the **Register** method, remove all existing route registrations.
2. Add a new route to the **Register** method by using the following information:
    - Name: **PhotoApi**
    - Route template: **api/photos/{id}**
    - Default controller: **PhotoApi**
    - Default action: **GetPhotoById**
    - Constraint: **id= "[0-9]+"**
3. After the **PhotoApi** route, add a new route to the **Register** method by using the following information:
    - Name: **PhotoTitleApi**
    - Route template: **api/photos/{title}**
    - Default controller: **PhotoApi**
    - Default action: **GetPhotoByTitle**
4. After the **PhotoTitleApi** route, add a new route to the **Register** method by using the following information:
    - Name: **PhotosApi**
    - Route template: **api/photos**
    - Default controller: **PhotoApi**
    - Default action: **GetAllPhotos**
5. Save all the changes.

▶ Task 3: Configure media-type formatters.

1. In the **WebApiConfig.cs** code file, at the end of the **Register** method, create a new variable named **json**. Set this variable to **config.Formatters.JsonFormatter**.
2. Set the **json.SerializerSettings.PreserveReferencesHandling** property to **Newtonsoft.Json.PreserveReferencesHandling.Objects**.
3. Remove the **XmlFormatter** object from the **config.Formatters** collection.
4. Save all the changes.

▶ Task 4: Test the Web API with Internet Explorer.

1. Start that web application in debugging mode.
2. Request the photo with ID 4 by using the Web API. Display the returned JSON file by using Visual Studio and check that the **Title** property is **Sample Photo 4**.
3. Request the photo with title **Sample Photo 5** by using the Web API. Display the returned JSON file by using Visual Studio and check that the **Title** property is **Sample Photo 5**.
4. Request all photos by using the Web API. Display the returned JSON file by using Visual Studio, and check that both **Sample Photo 9** and **Sample Photo 13** are present.
5. Close Visual Studio and stop debugging.

Results: At the end of this exercise, you will be able to create a simple Web API for an ASP.NET MVC 4 web application.

Exercise 2: Using the Web API for a Bing Maps Display

Scenario

You need to use the new Web API to obtain the photos in the client-side jQuery code. You will use latitude and longitude properties to display these photos as pins on a Bing API map.

To create the map display in the Photo Sharing application, you must add a new view and action for the photo controller. You must also add a new template view because the Bing Maps AJAX control requires a different **<!DOCTYPE>** directive to the one in use elsewhere in the Photo Sharing application. You will import a JavaScript file with basic Bing Maps code in place. To this JavaScript file, you will add code to call the Web API, obtain photo details, and display them on the map.

In this exercise, you will:

- Create a new template view.
- Create a map action, view, and script file.
- Obtain and display photos.
- Test the Bing Maps control.

The main tasks for this exercise are as follows:

1. Create a new template view.
2. Create a map action, view, and script file.
3. Obtain and display photos.
4. Test the Bing Maps control.

▶ Task 1: Create a new template view.

1. In the **Views/Shared** folder, create a copy of the **_MainLayout.cshtml** view file and name the copy as **_MapLayout.cshtml**.
2. In the **_MapLayout.cshtml** file, replace the **<!DOCTYPE html>** declaration with **<!DOCTYPE html PUBLIC "-//W3C/DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">**.
3. In the **onload** event of the **BODY** element, call the **GetMap()** JavaScript function.
4. Remove the links to jQueryUI and unobtrusive AJAX.
5. Add a new link to the Bing Maps AJAX control in the **HEAD** element by using the following information:
   - Charset: **UTF-8**
   - Type: **text/javascript**
   - SRC: **http://ecn.dev.virtualearth.net/mapcontrol/mapcontrol.ashx?v=7.0**
6. Save all the changes.

▶ Task 2: Create a map action, view, and script file.

1. Add a new action to the **PhotoController** class by using the following information:
   - Scope: **public**
   - Return type: **ViewResult**
   - Name: **Map**
   - Parameters: none
2. In the new **Map** action, return a **View** named **Map**.
3. Add a new view for the **Map** action by using the following information:
   - View name: **Map**
   - Type: Do not create a strongly-typed view
   - Layout: **_MapLayout.cshtml**
4. Remove the **H2** element from the **Map.cshmtl** view.
5. Add a new JavaScript block to the **Map.cshtml** view.
6. In the new JavaScript block, create a new variable named **webApiUrl** by using the **Url.Content()** helper to set the **webApiUrl** variable to **~/api/photos**.
7. Create a new variable named **pictureUrl** by using the **Url.Action()** helper to set the **pictureUrl** variable to the URL of the **GetImage** action of the **Photo** controller. Include a forward slash at the end of the URL.
8. Create a new variable named **displayUrl** by using the **Url.Action()** helper to set the **displayUrl** variable to the URL of the **Display** action of the **Photo** controller. Include a forward slash at the end of the URL.
9. Add a new JavaScript script, with an empty **src** attribute, to the **Map.cshtml** view.
10. Use the **Url.Content()** helper to set the **src** attribute in the new JavaScript code to **~/Scripts/MapDisplay.js**.
11. Create a new **DIV** element by using the following information:

- ID: **mapDiv**
- Style position: **absolute**
- Style width: **650px**
- Style height: **400px**

12. Add the **MapDisplay.js** JavaScript file to the **Scripts** folder from the following location:
    - **Allfiles (D):\Labfiles\Mod14\Bing Maps Script**
13. Add a new node to the site map by using the following information:
    - Title: **Map**
    - Visibility: *****
    - Controller: **Photo**
    - Action: **Map**
14. Save all the changes.
15. Start the web application in debugging mode and browse to the **Map** page.
16. Stop debugging.

▶ Task 3: Obtain and display photos.

1. In the **MapDisplay.js** script file, add a new function by using the following information:
    - Name: **GetPhotos**
    - Parameter: **serviceUrl**
2. Set the **$.support.cors** value to **true**.
3. Use the **$.ajax()** jQuery function to call the **GetPhotos** Web API by using the following information:
    - URL: **serviceUrl**
    - Type: **GET**
    - Data type: **json**
    - Success: **DisplayPics**
    - Error: **OnError**
4. Add a new function by using the following information:
    - Name: **DisplayPics**
    - Parameter: **response**
5. In the **DisplayPics** function, create two variables named **location** and **pin**.
6. Use the jQuery **$.each** function to loop through all the **photo** objects in the **response** collection.
7. For each **photo** object in the **response** collection, set the **location** variable to a new location by using the following information:
    - Object: **Microsoft.Maps.Location**
    - Latitude: **photo.Latitude**
    - Longitude: **photo.Longitude**
8. Set the **pin** variable to a new push pin by using the following information:

- Object: **Microsoft.Maps.Pushpin**
- Location: **location**

9. Set the **pin.Title** property to **photo.Title** and the **pin.ID** property to **photo.PhotoID**.
10. Ensure that the **DisplayInfoBox** method handles the **click** event for pushpins by using the following information:
    - Method: **Microsoft.Maps.Events.addHandler**
    - Object: **pin**
    - Event: **'click'**
    - Handler method: **DisplayInfoBox**
11. Add the **pin** object to the **dataLayer** object by using the **dataLayer.push** function.
12. Add a new function by using the following information:
    - Name: **OnError**
    - Parameter: **response**
13. In the **OnError** function, use the **alert** function to inform the user that the picture coordinates could not be obtained.
14. At the end of the **GetMap** function, call the **GetPhotos** function and pass the **webApiUrl** variable.
15. Save all the changes.

▶ Task 4: Test the Bing Maps control.

1. Start the web application in debugging mode and browse to the **Map** page to check the map control.
2. Click a pin of your choice.
3. Click the thumbnail.
4. Stop debugging and close Visual Studio.

Results: After completing this exercise, you will be able to create a template view to display a Bing Map AJAX control, and create a view and script file to display a Bing Map. You will also use jQuery to call a Web API and obtain details of photos. You will then mash up the data from a web API with Bing Maps data.

Question: How do the API actions you added to the **PhotoApiController** controller in Exercise 1 differ from other actions in MVC controllers?

Module Review and Takeaways

You can use the Web API framework to facilitate creating REST-style Web API calls in applications. REST-style Web API is recommended for mobile applications because of the light weight design of REST services.

REST services use HTTP methods such as **GET**, **POST**, and **PUT** to notify the API of the action that it needs to perform. Web APIs use the media formatter and the JSON.NET library to serialize and deserialize information, respectively. You can call Web API services by using server-side code, jQuery code, and the JSON.NET library.

Real-world Issues and Scenarios

Consider that you develop a mobile application by using Web APIs and the application needs to use currency rate services. For this application, you cannot use WCF, because WCF can impede the performance of the application by using XML for data exchanges. Therefore, you should use REST and JSON in the application to reduce the data that is transmitted between the client system and the server.

Review Question(s)

Question: We are developing a mobile application, which requires to access business data via internet. You are proposing to use Web API but your colleague is proposing to use WCF. What would be the key point for using Web API in this scenario?

Module 15

Handling Requests in ASP.NET MVC 4 Web Applications

Contents:

Module Overview

ASP.NET MVC provides functionalities that help develop web applications. However, ASP.NET MVC does not include functionalities that help change the encoding of the output. In such cases, you need to know how to use an HTTP module or HTTP handler, to facilitate such specific requests. Most web applications require two-way communication between the client and server systems. The web sockets protocol facilitates two-way communications in a robust manner.

Objectives

After completing this module, you will be able to:

- Use HTTP modules and HTTP handlers.
- Use web sockets.

Lesson 1
Using HTTP Modules and HTTP Handlers

Applications perform actions on a request in the HTTP pipeline, before rendering a webpage. You need to know how to use an HTTP module to implement custom authentication mechanism for a webpage, before rendering the webpage. You can also use an HTTP module to create code that renders content in a non-default encoding format that fits application needs. Sometimes, you may need the application to handle requests by using application logic that differs from the built-in ASP.NET page rendering logic. In such cases, you should know how to use the HTTP handler to process such specific requests.

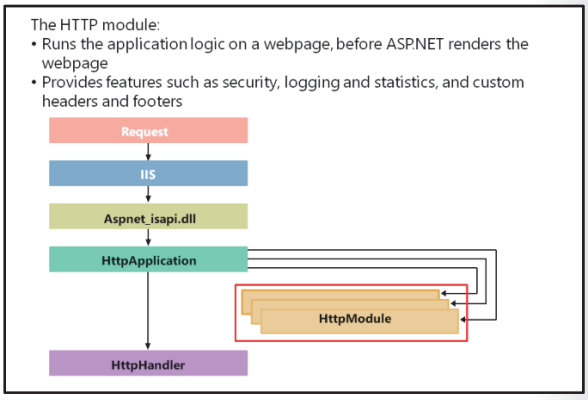
Lesson Objectives

After completing this lesson, you will be able to:

- Describe how HTTP modules intercept web requests.
- Create HTTP modules.
- Describe how HTTP handlers intercept web requests.
- Create HTTP handlers.
- Determine when to use HTTP modules and HTTP handlers.

What Is an HTTP Module?

An HTTP module is a program that runs application logic on a webpage, before ASP.NET renders the webpage. To understand the relevance of an HTTP module, you need to know about the HTTP request pipeline. The IIS HTTP request pipeline is a sequence of programs that run on requests, to help the application perform tasks. When the application receives a request, it passes the request to the **isapi_aspnet.dll** ISAPI library and starts the ASP.NET execution pipeline, to process the request.



Then, the request passes through the **HttpApplication** instance of the application and the **HttpModule** library.

HTTP modules provide the following features to an application:

- *Security*. Provide support for the custom authentication and authorization logic of your application
- *Logging and statistics*. Gather low-level information about the web application execution cycle, for monitoring and logging purposes
- *Custom headers and footers*. Allow insertion of custom header information in the response of each request

The following is a list of HTTP modules in ASP.NET:

- OutputCache
- Session

- WindowsAuthentication
- FormsAuthentication
- PassportAuthentication
- UrlAuthorization
- FileAuthorization
- DefaultAuthentication

If you configure HTTP modules in an application, HTTP modules apply to all the HTTP requests that the application receives. You cannot configure an HTTP module for a specific page.

Question: What are the benefits of using HTTP modules in web applications?

Creating HTTP Modules

You can create an HTTP module by adding the **CustomModule** class to your project; this class implements the **IHttpModule** interface in your application.

The following code shows how to create the **CustomModule** class in a project.

To create HTTP modules:

- Add the **CustomModule** class to implement the **IHttpModule** interface
- In the **CustomModule** class, implement the **ModuleName** property and the **Init** function
- Modify the Web.config file to register the HTTP module

Coding a Custom Module

```
public class CustomModule : IHttpModule
{
    public CustomModule ()
    {
    }
    public String ModuleName
    {
        get { return "CustomModule"; }
    }
    public void Init(HttpApplication application)
    {
context.Response.Write("<h1>Demo<h1>");
    }
}
```

In the **CustomModule** class, you need to implement the following two methods:

- *ModuleName property*. Provides a display name that enables other application code to identify the HTTP module.
- *Init function*. Provides the location to implement all logic for an HTTP module. The **HttpApplication** class triggers this function when the application receives a request.

After you complete the development of an HTTP module, you need to register the HTTP module in the Web.config file. The following code shows how to register an HTTP module for an application running in IIS 6.0 or later versions, configured in the Classic mode.

Registering an HTTP Module for IIS 6.0

```
<configuration>
  <system.web>
    <httpModules>
      <add name="CustomModule" type="CustomModule "/>
     </httpModules>
```

```
  </system.web>
</configuration>
```

The following code shows how to register an HTTP module for an application running in IIS 7.0 or later versions, configured in the Integrated mode.

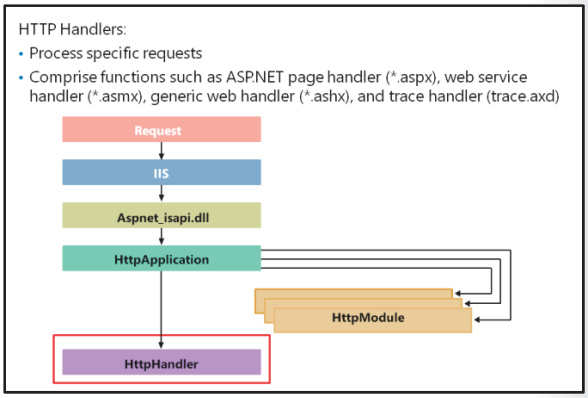
Registering an HTTP Module for IIS 7.0 or Later

```
<configuration>
  <system.webServer>
    <modules>
      <add name="CustomModule" type="CustomModule"/>
    </modules>
  </system.webServer>
</configuration>
```

Question: You have developed a custom HttpModule. What must you do to ensure that your ASP.NET application uses the custom module developed?

What Is an HTTP Handler?

HTTP modules help apply logic to all HTTP requests that an application receives. The **HttpHandler** function is a processing engine that processes specific HTTP requests. For example, you can configure an **HttpHandler** function to handle *.ashx files. You can map an **HttpHandler** function to a URL based on the file extension of the requested server-side page, such as *.aspx. The following list describes some common **HttpHandler** functions:



- *ASP.NET page handler (*.aspx)*. This is the default HTTP handler for ASP.NET pages.
- *Web service handler (*.asmx)*. This is the default HTTP handler for web service pages.
- *Generic web handler (*.ashx)*. This is the default HTTP handler for all web handlers that do not have a user interface and include the **@WebHandler** directive.
- *Trace handler (trace.axd)*. This is a handler that helps display page trace information.

You can use the *.ashx file extension with the Http handler because:

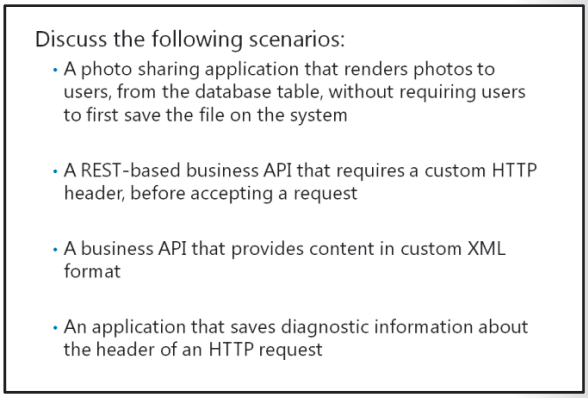
- The extension does not include any page-rendering logic.
- The extension allows developers to write logic to send responses to the client systems.

You can use a custom HTTP handler with the ***.rss** extension, to generate Real Simple Syndication (RSS) feed content for user requests. You can also use a custom handler to request images from the database and send the images to the client systems.

Question: What is the primary use of the HTTP generic handler (*.ashx)?

Discussion: Scenarios for HTTP Modules and Handlers

Consider the following scenarios. In each case, discuss with the rest of the class to determine whether an HttpModule or HttpHandler best suits the case.



- You are creating a photo sharing application, and you want to enable each user to discuss photos and cycling trips with their friends. You need to ensure that the application renders photos to users, from the database table, without requiring users to first save the file on their system.
- You are creating a Representational State Transfer (REST)-based business application programming interface (API) that requires a custom HTTP header, before accepting a request.
- You are creating a business API to provide content in custom XML format.
- You developing an application that requires saving diagnostic information about the header of an HTTP request.

Lesson 2
Using Web Sockets

AJAX technologies are constantly evolving. The HTTP protocol does not cater to crucial requirements such as real-time information updates from the server. You need to know how to use web sockets to create two-way communication between the client and server systems. You also need to know when to use the traditional HTTP model and the web sockets protocol, based on the needs of your application.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe the WebSocket protocol.
- Describe how to create web sockets connections.
- Determine when to use web sockets.
- Add a web sockets-based application to an ASP.NET MVC 4 web application.

What Is the WebSocket Protocol?

The WebSocket protocol is a standard from the World Wide Web Consortium (W3C). The HTML 5 specification includes support for web sockets.

Additional Reading: For more information about the WebSocket protocol, go to: http://go.microsoft.com/fwlink/?LinkID=288994&clcid=0x423

Characteristics of web sockets:

- W3C provides web sockets protocol to ensure that browsers support web sockets as part of the HTML5 implementation
- Web sockets facilitate two-way communication between client and server systems
- Web sockets eliminate the need to re-create requests multiple times
- Microsoft Internet Explorer 10 and Windows 8 applications are compatible with web sockets
- Web sockets function in a similar manner as traditional network sockets

The protocol facilitates two-way communication between web browsers and web servers. The protocol helps overcome the issue of HTTP not providing two-way communication.

Developers often build two-way communication applications by using inefficient techniques such as long running loops and polling. These techniques consume large amounts of memory and other resources of both the server and client systems. The WebSocket protocol resolves the problems that arise with using such techniques by creating a constant socket connection between the client and server. All communication between the client and server systems then occurs through the socket. This socket eliminates the need to re-create a request multiple times, during real-time communications between the client and server systems. Remember that all browsers may not support web sockets. Microsoft Internet Explorer 10 and Windows 8 applications, which use JScript and HTML5, do support web sockets. Latest versions of Chrome, Safari, and Firefox also support web sockets.

The web sockets work in a similar manner as traditional network sockets. The only difference is that during the initial handshake, web sockets use the Upgrade HTTP request that includes the **Upgrade** HTTP header. If the server accepts web sockets, the server returns a response with the status code, 101. Then, the client and server systems send the payload by using the socket. When the client system no longer requires communication with the server, the application sends a close connection payload, to notify the server to close the web socket.

Question: What is the key difference between traditional HTTP and web sockets?

Coding Web Sockets Connections

To use web sockets, you need to implement server-side services that can handle the two-way communication between the server and the client systems. To simplify the programming work, Microsoft provides the **Microsoft.WebSockets** NuGet package. This package includes a library that helps handle web socket operations on the server. While coding web sockets, you can:

- Use the Web API framework to create a REST service that handles the initial handshake.
- Use the WebSocket library to promote communication between the client and server systems.

Creating web sockets connection:

- Install the **Microsoft.WebSockets** NuGet package
- Create a REST service and use the WebSockets library to handle the operations of web sockets in the server
- Add code to handle the communications when the client system sends messages to the service application hosted on the web server
- Use the **WebSocket** object to establish two-way communication between the client and server systems
- Add JavaScript functions to respond to events
- Use the **WebSocket.send** function to send messages to the server

The following code shows how to create a REST service and use the WebSocket library, to handle the operations of web sockets on the server.

Using the Web Sockets Library

```
public   class ChatController : ApiController
{
   public HttpResponseMessage Get(string username)
   {
      HttpContext.Current.AcceptWebSocketRequest(new ChatWebSocketHandler());
      return Request.CreateResponse(HttpStatusCode.SwitchingProtocols);
   }
   class ChatWebSocketHandler : WebSocketHandler
   {
      public ChatWebSocketHandler()
      {
      }
      public override void OnOpen()
      {
      }
      public override void OnMessage(string message)
      {
      }
   }
}
```

Observe the **Get** function in the preceding code sample. The ASP.NET engine maps this function to a URL and the controller with a GET request. After the initial handshake:

- The WebSocket library establishes the web socket at the client and server end.
- The application passes the handling of communications to the **ChatWebSocketHandler** class inherited from the **WebSocketHandler** class.

Whenever the client system sends a message to the service application hosted on the web server, the message triggers the **OnMessage** function. The **OnMessage** function enables you to respond to the message.

To ensure that a message triggers the **OnMessage** function, you need to add the following code to the Web.config file.

Configuring Web Sockets

```
<configuration>
```

```
   <appSettings>
      <add key="aspnet:UseTaskFriendlySynchronizationContext" value="true" />
   </appSettings>
</configuration>
```

On the client side, you can use the **WebSocket** object to establish two-way communication between the client and server systems. The following code shows how to use the **WebSocket** object in JavaScript.

Calling Web Sockets from the Browser

```
$(document).ready(function () {
   websocket = new WebSocket('ws://localhost/api/service1');
   websocket.onopen = function () {
   };
   websocket.onerror = function (event) {
   };
   websocket.onmessage = function (event) {
   };
   websocket.send('Hello');
   websocket.close();
});
```

The preceding code sample shows how to add JavaScript functions to respond to events. The following list describes some common JavaScript functions.

- *onmessage*. You can use this function when the client system receives a message from the server.
- *onopen*. You can use this function when the WebSocket library establishes the web socket.
- *onerror*. You can use this function when an error occurs.

To send a message in JavaScript, you can use the **WebSocket.send()** function. After all communications between the client and server systems end and you no longer need the web socket connection, you can use the **WebSocket.close()** function. This function helps close the web socket communication channel between the client and server systems.

Question: What is the purpose of using Web APIs to create a service for handling web sockets?

What Is SignalR?

SignalR is a set of components that simplifies the development of bidirectional real-time web applications, such as chat rooms in websites. SignalR uses WebSockets wherever possible to connect. This means that it supports all HTML5 compatible browsers. A key advantage of using SignalR is that it supports remote procedure calls (RPCs) with HTML or even .NET applications and can automatically switch to periodic polling for older browsers.

- SignalR is a library developed to simplify development for bidirectional real-time communications in web applications
- SignalR includes a .NET library for handling server-side communications
- SignalR includes a JavaScript library for handling client-side communications

To install the package to your application, you can use NuGet to find the package, **Microsoft.AspNet.SignalR**.

After installation, you can implement the server-side code for the server logic.

To use SignalR, you need to implement a class that inherits from SignalR library. The following example broadcasts a message to all clients when a message is received from one client.

Coding SignalR Connections

```
using System.Threading.Tasks;
using Microsoft.AspNet.SignalR;
public class DemoConnection : PersistentConnection
{
    protected override Task OnReceived(IRequest request, string connectionId, string
data)
    {
        return Connection.Broadcast(data);
    }
}
```

After the server code is implemented, you need to add ASP.NET routing to map the URL back to the SignalR library. The following code maps the /demo URL to the DemoConnection library for handling the message communication by using SignalR.

Mapping SignalR URLs

```
public class Global : System.Web.HttpApplication
{
    protected void Application_Start(object sender, EventArgs e)
    {
        RouteTable.Routes.MapConnection<DemoConnection>("demo", "/demo");
    }
}
```

SignalR also includes a JavaScript library that can be used by clients to help perform communications between client and server. The SignalR library depends on the jQuery library.

The following JavaScript code sends a message to the server, receives a message back, and displays the received message.

Client-Side SignalR Code

```
<script src="http://code.jquery.com/jquery-1.7.min.js" type="text/javascript"></script>
<script src="Scripts/jquery.signalR-1.0.1.min.js" type="text/javascript"></script>
<script type="text/javascript">
   $(function () {
      var connection = $.connection('/demo);
      connection.received(function (data) {
         alert(data);
      });
      connection.start().done(function() {
         $("#broadcast").click(function () {
            connection.send('Hello');
         });
      });
  });
</script>
```

Question: What is the key benefit of using SignalR, instead of WebSockets directly?

Demonstration: How to Add a Chat Room to a Web Application by using SignalR

In this demonstration, you will see how to add a simple chat room to a web application by using SignalR.

Demonstration Steps

1. In the Solution Explorer pane of the **OperasWebsite – Microsoft Visual Studio** window, right-click **OperasWebSite**, point to **Add**, and then click **Class**.

2. In the **Name** box of the **Add New Item – OperasWebSite** dialog box, type **ChatHub**, and then click **Add**.
3. In the ChatHub.cs code window, locate the following code.

```
using System.Web;
```

4. Place the mouse cursor at the end of the located code, press Enter, and then type the following code.

```
using Microsoft.AspNet.SignalR;
```

5. In the ChatHub.cs code window, locate the following code.

```
public class ChatHub
```

6. Replace the located code with the following code.

```
public class ChatHub : Hub
```

7. In the **ChatHub** class code block, type the following code.

```
public void Send(string name, string message)
{
}
```

8. In the **Send** method code block, type the following code.

```
Clients.All.broadcastMessage(name, message);
```

Note: The **Send()** method sends any message received, back to all the clients that are connected to the hub. You need to define the **broadcastMessage()** method in the client-side code to receive messages. The client-side code must also call the **Send()** method to broadcast messages.

9. In the Solution Explorer pane, expand **Views,** expand **Home**, and then click **Chat.cshtml**.
10. In the Chat.cshtml code window, within the final **<script>** element, type the following code.

```
$(function() {
});
```

11. Within the anonymous function you just created, type the following code.

```
var chat = $.connection.chatHub;
```

12. Place the mouse cursor at the end of the variable you just created, press Enter, and then type the following code.

```
chat.client.broadcastMessage = function (name, message) {
};
```

Note: This function is the implementation of the **broadcastMessage()** function that you called in the Hub code.

13. Within the anonymous function you just created, type the following code.

```
var listItem = '<li>' + name + ': ' + message + '</li>';
```

14. Place the mouse cursor at the end of the variable you just created, press Enter, and then type the following code.

```
$('#discussion').append(listItem);
```

15. Place the mouse cursor at the end of the **broadcastMessage** function code block, press Enter, and then type the following code.

```
var displayname = prompt('Enter your name:', '');
```

16. Place the mouse cursor at the end of the **displayname** variable code block you just created, press Enter, and then type the following code.

```
$('#chat-message').focus();
```

17. Place the mouse cursor at the end of the code block you just created, press Enter, and then type the following code.

```
$.connection.hub.start().done(function () {
});
```

18. Within the anonymous function code block you just created, type the following code.

```
$('#sendmessage').click(function () {
});
```

19. Within the new anonymous function code block you just created, type the following code.

```
chat.server.send(displayname, $('#chat-message').val());
```

20. Place the mouse cursor at the end of the code block you just created, press Enter, and then type the following code.

```
$('#chat-message').val('').focus();
```

21. On the **DEBUG** menu of the **OperasWebSite – Microsoft Visual Studio** window, click **Start Debugging**.
22. On the Operas I Have Seen page, click the **Enter the Operas Chat Room** link.
23. In the **Enter your name** box of the **localhost needs some information** dialog box, type **Rebecca Laszlo**, and then click **OK**.
24. In the **Message** box of the Operas I Have Seen page, type a message of your choice, and then click **Send**.

Note: SignalR sends the message you typed to the hub. The hub broadcasts the message to all connected clients.

25. On the taskbar, right-click the **Internet Explorer** icon, and then click **Internet Explorer**.
26. In the Address bar of the Internet Explorer window, type **http://localhost:<***portnumber***>**, and then press Enter.
27. On the Operas I Have Seen page, click the **Enter the Operas Chat Room** link.
28. In the **Enter your name** box of the **localhost needs some information** dialog box, type **Elisa Graceffo**, and then click **OK**.
29. In the **Message** box of the Operas I Have Seen page, type a message of your choice, and then click **Send**.
30. On the taskbar, click the first instance of the Internet Explorer window. Note that the message from **Elisa Graceffo** is displayed because both users are connected to the same hub.

31. Close all the Internet Explorer windows.
32. In the **OperasWebSite – Microsoft Visual Studio** window, click the **Close** button.

Lab: Handling Requests in ASP.NET MVC 4 Web Applications

Scenario

The Adventures Works board and managers are pleased with the Photo Sharing application, but have requested that interactivity should be maximized to encourage users to register and participate fully in the community. Therefore, you have been asked to add chat functionality to the application. Authenticated members should be able to start a chat on a particular photo from the Display view. Chat rooms for each photo should be separated from each other. Users in the chat room should be able to send a message to all other users in that chat room, and they should be able to see all the messages that have been sent since they joined the chat room.

You have decided to use SignalR to implement the chat room over Web Sockets.

Objectives

After completing this lab, you will be able to:

- Install SignalR in an ASP.NET MVC 4 web application.
- Configure SignalR on the server and create a SignalR hub.
- Link to the required script files for SignalR in an MVC view.
- Create the script for SignalR connections and send messages to groups.

Lab Setup

Estimated Time:

60 minutes

Virtual Machine: **20486B-SEA-DEV11**

User Name: **Admin**

Password: **Pa$$w0rd**

Note: In Hyper-V Manager, start the **MSL-TMG1** virtual machine if it is not already running.

Before initiating this lab, perform the following steps:

a. Apply the Snapshot of the virtual machine, **20486B-SEA-DEV11**, that was taken after completing the lab in module 13.

b. Navigate to **Allfiles (D):\Labfiles\Mod 13\Starter\PhotoSharingApplication\PhotoSharingApplication**, and then copy the **Web.config** file.

c. Navigate to **Allfiles (D):\Labfiles\Mod 15\Starter\PhotoSharingApplication\PhotoSharingApplication**, and then paste the **Web.config** file.

d. Enable the **Allow NuGet to download missing packages during build** option, by performing the following steps:

   i. On the **TOOLS** menu of the Microsoft Visual Studio window, click **Options**.

   ii. In the navigation pane of the **Options** dialog box, click **Package Manager**.

   iii. Under the Package Restore section, select the **Allow NuGet to download missing packages during build** checkbox, and then click **OK**.

Exercise 1: Creating a SignalR Hub

Scenario

Before you can write JScript code on the client to connect to SignalR, you must configure and code a SignalR Hub on the web server.

In this exercise, you will:

- Install SignalR in the Photo Sharing application.
- Configure routing.
- Create a SignalR hub to accept messages from clients and forward those messages to other clients who are chatting about the same photo.

The main tasks for this exercise are as follows:

1. Install SignalR.
2. Create a Hub class.
3. Configure SignalR routes.

▶ Task 1: Install SignalR.

1. Start the virtual machine, and log on with the following credentials:
   - Virtual machine: **20486B-SEA-DEV11**
   - User name: **Admin**
   - Password: **Pa$$w0rd**
2. Open the PhotoSharingApplication.sln file from the following location:
   - File location: **Allfiles (D):\Labfiles\Mod15\Starter\PhotoSharingApplication**
3. In the **NuGet Package Manager**, search for the following pre-release package:
   - **Microsoft.aspnet.signalr**
4. Install the **Microsoft ASP.NET SignalR** package in the **PhotoSharingApplication** project.
5. Notice the additions that the NuGet package has made to the project references and the Scripts folder.

▶ Task 2: Create a Hub class.

1. Add a new class file named **ChatHub.cs** to the **PhotoSharingApplication**.
2. Remove the following namespace references from the **ChatHub.cs** class file:
   - System.Collections.Generic;
   - System.Linq;
3. Add the following namespace references to the **ChatHub.cs** class file:
   - System.Threading.Tasks
   - Microsoft.AspNet.SignalR
4. Ensure that the **ChatHub** class inherits from the **Microsoft.AspNet.SignalR.Hub** class.
5. In the **ChatHub** class, create a new method by using the following information:
   - Scope: **public**
   - Return type: **Task**

- Name: **Join**
- Parameter: an integer named **photoId**

6. In the **Join** method, return the result of the **Groups.Add()** method by using the following information:
   - Connection ID: **Context.ConnectionId**
   - Group name: **"Photo" + photoId**
7. In the **ChatHub** class, create a new method by using following information:
   - Scope: **public**
   - Return type: **Task**
   - Name: **Send**
   - First parameter: a string named **username**
   - Second parameter: an integer named **photoId**
   - Third parameter: a string named **message**
8. In the **Send** method, create a new **string** variable named **groupName** and set the value to **"Photo" + photoId**.
9. In the **Send** method, return the result of the **addMessage** method for the **groupName** group by using the following information:
   - Method: **Clients.Group(groupName).addMessage()**
   - User name: **username**
   - Message: **message**
10. Save all the changes.

▶ Task 3: Configure SignalR routes.

1. In the **Global.asax** code-behind file, add a reference to the **Microsoft.AspNet.SignalR** namespace.
2. In the **Application_Start()** method, immediately after the **RegisterAllAreas()** code, call the **RouteTable.Routes.MapHubs()** method.
3. Save all the changes.

Results: After completing this exercise, you will be able to install SignalR in an MVC 4 web application, configure routes for SignalR, and create a hub to receive and forward simple text messages.

Exercise 2: Creating a Photo Chat View

Scenario

Now that you have set up and configured SignalR and a SignalR hub on the server side, you must use JScript and the SignalR JScript library to send and receive messages on the client side.

In this exercise, you will:

- Create a new MVC controller action and Razor view to display the chat user interface for a particular photo.

- Link to the JScript libraries that SignalR requires and write client-side script to call the **Join()** and **Send()** methods on the hub.
- Test the chat functionality.

The main tasks for this exercise are as follows:

1. Create a chat action and view.
2. Link to the chat view.
3. Link to JScript files.
4. Script SignalR connections.
5. Script SignalR messages.
6. Test the chat room.

▶ Task 1: Create a chat action and view.

1. Add a new action to the **PhotoController** class by using the following information:
   - Annotation: **[Authorize]**
   - Scope: **public**
   - Return type: **ActionResult**
   - Name: **Chat**
   - Parameter: an integer named **id**
2. Create a new **Photo** object named **photo** and get the **photo** value by passing the **id** parameter to the **context.FindPhotoById()** method.
3. If the **photo** object is null, return an HTTP Not Found status code.
4. At the end of the action, return the **Chat** view and pass the **photo** object as the model class.
5. Add the following view file to the **Views/Photo** folder:
   - **Allfiles (D):\Labfiles\Mod15\Chat View\Chat.cshtml**
6. Save all the changes.

▶ Task 2: Link to the chat view.

1. In the **Display.cshtml** view file, after the **DIV** element with ID **addtofavorites**, add a new **DIV** element, with ID **chataboutthisphoto**.
2. In the new **DIV** element, render a link to the **Chat** view by using the following information:
   - Helper: **Html.ActionLink()**
   - Text: **Chat about this photo**
   - View: **Chat**
   - Route values: pass the **Model.PhotoID** value to the **id** parameter.
3. Start the web application in the debugging mode, display a photo of your choice, and then click **Chat**.
4. Log on with the following credentials:
   - User name: **David Johnson**
   - Password: **Pa$$w0rd2**

5. Attempt to send a chat message.
6. Stop debugging.

▶ Task 3: Link to JScript files.

1. Add a new **SCRIPT** element at the end of the **Chat.cshtml** view file, with type **text/javascript**.
2. In the new **SCRIPT** element, create a new variable named **username**. In a new Razor code block, use the **User.Identity.Name** property to set the value for the variable.
3. Create a second new variable named **photoid** and use the **Model.PhotoID** property to set the value for the variable.
4. Add a new **SCRIPT** element, with type **text/javascript**, and an empty **src** attribute.
5. In the new script element, use the **Url.Content()** helper to set the **src** attribute to the following path:
   - ~/Scripts/jquery.signalR-1.0.0.js

   Note: NuGet installs the latest version of SignalR. Ensure that the name of the script file you enter matches the name of the file in the Scripts folder.
6. Add a new **SCRIPT** element, with type **text/javascript**, and an empty **src** attribute.
7. In the new script element, use the **Url.Content()** helper to set the **src** attribute to the following path:
   - ~/signalr/hubs
8. Add a new **SCRIPT** element, with type **text/javascript**, and an empty **src** attribute.
9. In the new script element, use the **Url.Content()** helper to set the **src** attribute to the following path:
   - ~/Scripts/ChatRoom.js
10. Save all the changes.

▶ Task 4: Script SignalR connections.

1. Add a new JScript file named **ChatRoom.js** to the **Scripts** folder.
2. In the new JavaScript file, use jQuery to create an anonymous function that runs when the page loads.
3. Use the **$.connection.chatHub** property to obtain the **ChatHub** you already created, and then store the hub in a variable named **chat**.
4. Use jQuery to set the initial focus on the element with ID **chat-message**.
5. Create an anonymous function that runs when the **$.connection.hub.start()** method is done.
6. Call the **chat.server.join()** function on the SignalR hub, pass the **photoid** value as a parameter, and then create an anonymous function that runs when the function is done.
7. Save all the changes.

▶ Task 5: Script SignalR messages.

1. In the **ChatRoom.js** JScript file, after creating and instantiating the **chat** variable, set the **chat.client.addMessage** property to be a new anonymous function with two parameters, **name** and **message**.
2. In the new function, create a variable named **encodedName**, and use jQuery to set this variable to a new ** ** with the **name** parameter as its HTML content.
3. In the new function, create a variable named **encodedMessage**, and use jQuery to set this variable to a new ** ** with the **message** parameter as its HTML content.

4. Create a new variable named **listItem**. Set the value of this variable to an HTML **LI** element that includes the **encodedName** and **encodedMessage** variables.
5. Append the **listItem** element to the page element with ID **discussion**.
6. In the function that runs when the **client.server.join()** method is done, create an anonymous function that runs when the button with ID **sendmessage** is clicked.
7. In the new anonymous function, call the **chat.server.send()** method by using the following information:
   - User name: **username**
   - Photo ID: **photoid**
   - Message: use the value of the element with ID **chat-message**
8. Use jQuery to obtain the element with ID **chat-message**, set its value to an empty string, and give it the focus.
9. Save all the changes.

▶ Task 6: Test the chat room.

1. Start the web application in debugging mode and log on with the following credentials:
   - User name: **David Johnson**
   - Password: **Pa$$w0rd2**
2. Browse to the chat page for **Sample Photo 1**.
3. Send a message of your choice and observe the results.
4. Start a new instance of Windows Internet Explorer, and browse to the Photo Sharing application home page.
5. Register a new user account with the following credentials:
   - User name: **Mark Steele**
   - Password: **Pa$$w0rd**
6. Browse to the chat page for **Sample Photo 1**, and then send a message of your choice.
7. Switch to the first instance of Internet Explorer, and then send a second message of your choice. Observe the messages sent between the two users.
8. Stop debugging and close Visual Studio.

Results: After completing this exercise, you will be able to create MVC controller actions and views to display a user interface for SignalR functionality, link to the JScript libraries that SignalR requires, and use JScript to connect to a SignalR hub and send messages.

Question: In the chat functionality that you created, each photo in the Photo Sharing application has a separate chat room. How is this separation possible with one SignalR hub?

Question: In Exercise 2, you wrote JScript code that called the **chat.server.join()** and **chat.server.send()** functions. In which script file are these functions defined?

Module Review and Takeaways

Most web applications do not limit themselves to rendering HTML content. They require functionalities that support custom logic such as custom authentication. To suit such cases, you can use HTTP modules and HTTP handlers to run custom logic before or after rendering a webpage. The web socket technology in MVC 4 provides support for two-way communication between client and server systems. This technology is useful for web applications that require constant updates between the client and server systems.

Real-world Issues and Scenarios

You create an application that obtains the latest product pricing information from the internal database. The pricing is constantly updated every time business users feel the need for updates. Therefore, you need to ensure that the application updates pricing information every five minutes. In such cases, you can to use the web socket technology to implement price update. You can also add code to download the product image stored in the product database by using a generic handler (*.ashx file).

Review Question(s)

Question: You want to select technology that could be used for developing a web based chatting applications for your client. Which technology would you prefer in this scenario?

Module 16

Deploying ASP.NET MVC 4 Web Applications

Contents:

Module Overview

After developing an application, you need to deploy the application to the production environment to make the application available to users. The configuration or deployment method of each application varies based on the setup of the production environment. ASP.NET MVC applications have some special considerations such as the configuration needed for IIS.

Objectives

After completing this module, you will be able to:

- Configure and deploy an ASP.NET web application.
- Deploy an ASP.NET MVC 4 web application.

Lesson 1
Deploying a Web Application

The considerations for deploying a web application to a web farm are different for single and multi-server configurations. For example, in multi-server web farms, you need to ensure that the state information is available on all web servers. You also need to ensure that the dependencies needed for hosting ASP.NET MVC 4 web applications are present in the web farm. If any of the application dependencies are missing on the production web servers, some or all of your applications may not function and users may encounter errors. You need to apply special considerations while deploying web applications on Windows Azure. Deploying a web application on Windows Azure provides advantages such as high availability and flexibility.

Lesson Objectives

After completing this lesson, you will be able to:

- Ensure that dependencies are present for hosting ASP.NET MVC4 applications.
- Deploy an ASP.NET MVC4 web application to a single-server web farm.
- Deploy an ASP.NET MVC4 web application to a multi-server web farm.
- Deploy an ASP.NET MVC 4 web application on Windows Azure.

ASP.NET MVC 4 Dependencies

ASP.NET MVC 4 is built on the ASP.NET Framework 4.0. Each web application requires a range of different components to be present on the web server, to function appropriately. Examples of common requirements include:

ASP.NET MVC 4 dependencies include:

- The ASP.NET Framework 4.0 Common Language Runtime (CLR)
- The MVC 4 runtime
- A Database server
- Entity Framework
- Membership Providers

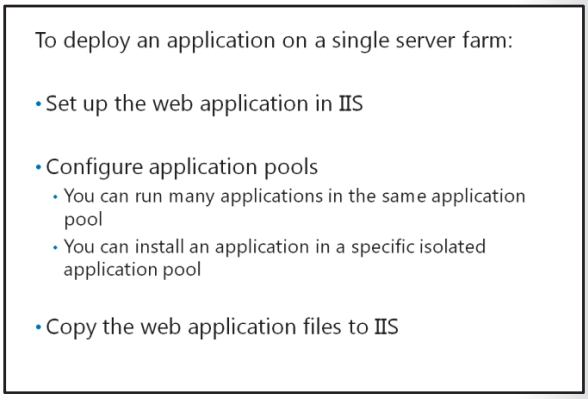
- *The ASP.NET Framework 4.0 Common Language Runtime (CLR)*. The CLR runs any managed code such as C# classes. All ASP.NET web applications require the CLR, and you can install the CLR with Internet Information Services (IIS).
- *The ASP.NET MVC 4 runtime*. The ASP.NET MVC 4 runtime locates controllers and actions to handle each request that the web application receives, and it returns the compiled webpages to users. You can install the MVC runtime in your web application by installing the MVC 4 NuGet package. All MVC web applications require the MVC runtime.
- *A Database server*. Most web applications use a database to store information such as product details, customer details, images, and other entities. You need to ensure that the production environment supports the database that you used during development. For example, if you use SQL Compact during development, you need to ensure that Microsoft SQL Compact is installed on the production server. Alternatively, you can migrate the database to Microsoft SQL Server.
- *Entity Framework*. If you use Entity Framework to model data in your web application, you need to ensure that you deploy the framework with your application.
- *Membership Providers*. If your application uses membership providers to store user accounts and roles, you need to ensure that these providers are available on the server. The .NET Framework

includes the SQL Server Membership Provider. If you use the Universal Membership Providers by installing the NuGet package, you need to ensure that you include the NuGet package with your deployed application.

Question: Which of the common requirements in the above list is required to support forms authentication?

Deploying Web Applications to Web Servers

Before you deploy a web application to a production IIS web server, you must create and configure the IIS application and the folder that hosts the application.



Creating an IIS Web Application

You need to host ASP.NET MVC 4 web applications in an IIS web folder that is marked as a web application starting point. You can use the Internet Service Manager tool to configure this setting on the top-level folder of the web application. When you mark a web folder as a web application, you enable IIS to recognize that the content of the folder is ASP.NET and requires the ASP.NET runtime engine to process each request.

After creating an application, you need to configure the application to use .NET 4.0 or above. This practice enables ASP.NET MVC 4 to run, because the MVC runtime components are part of the .NET framework.

IIS Application Pools

In IIS, web applications are run within a context known as an application pool. You can run many applications in the same application pool or install an application in its own isolated pool. Applications that run in the same application pool share the following resources:

- *Worker Process*. The worker process uses a single execution process to run all the applications in a pool.
- *Memory*. The applications share a single memory range, because the applications in a pool share a single worker process.
- *Identity*. The applications within a single pool share a user account.

You can place two or more applications in an application pool, and thereby reduce the total memory usage, because the applications share memory. However, this arrangement can reduce the reliability of each application, because a malfunctioning application can affect all the applications in the pool. To maximize the reliability of an application, you should install the application in an isolated pool that runs only one specific application.

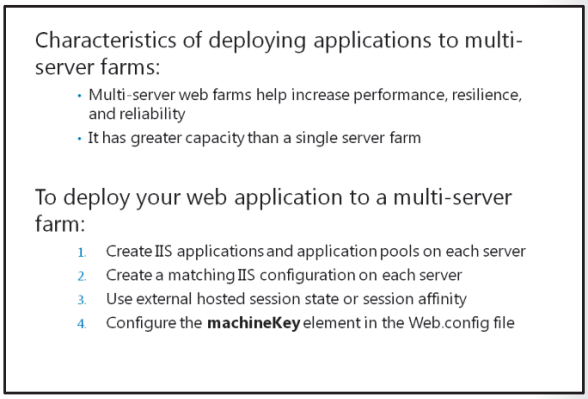
The shared identity within an application pool affects the way the applications authenticate resources such as database servers when using integrated authentication. Each application within a pool shares a single user account. If each application has a separate database, you need to set permissions, to enable the user accounts to access all the databases. If you separate applications into isolated pools, you can use a different user account for each application. This practice can enhance the security of the application, because each account can access only one database.

Deploying the Application

When the IIS web application is ready, you need to copy the web application files to IIS. These files include all the model classes, controllers, views, script files, style sheets, images, and any other content in the Microsoft Visual Studio ASP.NET MVC 4 project. You can use the deployment tools available in Microsoft Visual Studio, or other technologies such as FTP, to copy these files. When the copy operation is complete, the web application is ready for deployment.

Question: What is the purpose of configuring additional application pools?

Deploying Web Applications to Multi-Server Farms



You can deploy a web application to a multi-server web farm to increase performance, resilience, and reliability. In a multi-server web farm, a group of two or more web servers host a single web application. All the servers in the farm share a single server name, such as www.contoso.com. Windows Network Load Balancing or dedicated hardware load balancers distribute browser requests to the servers.

A multi-server farm usually has greater capacity than a single web server, because the multi-server farm almost equally shares the load amongst several servers. Multi-server farms also increase reliability. When a server fails for any reason, the load balancing mechanism automatically directs requests to another server that remains online.

If you decide to deploy your web application to a multi-server farm, you need to perform the following steps. To complete these steps, you may need to work with the web server administrator who is responsible for the server farm:

1. Create IIS applications and application pools on each server. This step is the same as configuring the applications and application pools in a single-server scenario. However, you need to perform this step on every server in the farm. You also need to ensure that applications and application pools have the same configuration, throughout the farm.
2. Create a matching IIS configuration on each server. IIS configuration options, such as any encryption certificates, file extensions, and optional components, should be identical on every server in the farm.
3. Use external hosted session state or session affinity. You can configure ASP.NET web applications to store session state in the Windows State Service or a database. You should use this technique in a multi-server farm, because each request from a single user session might be directed to a different web server. Alternatively, by configuring session affinity, you can ensure that all requests from a user session are always sent to the same web server. If you do not use external hosted session state or session affinity, the user preference on one web server may get lost when a request is sent to another web server. Session affinity is supported by some but not all load balancing solutions. Work with your web server administrator to determine if affinity is available on your farm.
4. Configure the **machineKey** element in the Web.config file. If you are using external hosted session state, you can encrypt the connection between the web servers and the State Service server or database server. This technique improves security by protecting session state during network transmissions. You can configure this encryption by setting up the **machineKey** element in the Windows registry.

Additional Reading: For more information about configuring the **machineKey** element, go to: http://go.microsoft.com/fwlink/?LinkID=288995&clcid=0x424

Question: What is the purpose of configuring the **machineKey** element in the Web.config file?

Deploying Web Applications on Windows Azure

You can deploy any ASP.NET web application, including ASP.NET MVC 4 web applications, on Windows Azure. By choosing Windows Azure as a platform, you can use a highly-available and flexible infrastructure that you can optimize to cope with intense load. You have a choice of subscriptions, which can help you to pay only for the traffic that your web application uses.

To deploy an application on Windows Azure:

1. Create a new web application in the Windows Azure management portal
2. Download a publishing profile
3. Start the Publish wizard and import the publishing profile
4. Configure connection strings
5. Observe that Microsoft Visual Studio publishes the web application on Windows Azure

Deploying a web application on Windows Azure is different from deploying the application on on-premise versions of IIS. To deploy an application on Windows Azure, you perform the following steps:

1. Create a new web application in the Windows Azure management portal.
2. Download and save a publishing profile for the new web application.
3. In Microsoft Visual Studio, open your web application solution. Then, start the Publish wizard and import the publishing profile.
4. Ensure that you select the correct connection strings for Entity Framework connections, service database connections, and any other database connections.
5. Complete the Publish wizard.
6. Observe that Microsoft Visual Studio publishes the web application to Windows Azure.

When the process is complete, you can access the Windows Azure-hosted web application from any Internet-connected browser.

Note: At the time of writing, Windows Azure web applications are still in preview, within the Windows Azure platform.

Windows Azure Web Applications and Windows Azure Cloud Services

Instead of publishing an ASP.NET MVC web application as a web application in Windows Azure, you can choose to build and publish a Windows Azure cloud service. A cloud service is a Windows Azure component that you can use to build a flexible multi-tiered application, with a discrete architecture. A cloud service consists of one or more web roles and worker roles, each with its own application files and configuration.

In a Windows Azure web application, all components of your web application run in a single process. You can administer the web application as a single object.

Windows Azure cloud services help separate user interface logic that you build in a web-role project, from business logic that you build in a worker process project. This separation provides greater flexibility, and it

enables calling the business logic from other processes, such as mobile applications. You can separately administer web roles and worker processes, and run them in separate processes. However, to publish a web application as a cloud service with separate web roles and worker roles, you need to download and install the Windows Azure SDK. You can download and install the Windows Azure SDK by using the Web Platform Installer. You can use the Windows Azure cloud service templates as the starting point for your application.

Question: How does Windows Azure cloud services help enhance the flexibility of the application?

Demonstration: How to Create a Windows Azure Website

In this demonstration, you will see how to create a new database and a new website in Windows Azure.

Demonstration Steps

1. On the taskbar, click the **Internet Explorer** icon.
2. In the Address bar of the Internet Explorer window, type **http://www.windowsazure.com**, and then click the **Go to** button.
3. In the upper-right corner of the Windows Azure: Microsoft's Cloud Platform | Cloud Hosting | Cloud Services page, click **PORTAL**.
4. On the Sign in to your Microsoft account page, in the **Microsoft account** box, type <*your username*>, in the **Password** box, type **Pa$$w0rd**, and then click **Sign in**.
5. In the left pane of the Windows Azure page, click **WEB SITES**.
6. In the lower-left pane of the Windows Azure page, click **NEW**, and then click **CUSTOM CREATE**.
7. In the **URL** box of the Create Web Site page, type <*your username*>**operas**, and then, in the **REGION** box, click <*A region near you*>.
8. In the **DATABASE** box of the Create Web Site page, click **Create a new SQL database**, in the **DB CONNECTION STRING NAME** box, type **OperasDB**, and then click the **Next** button.
9. In the **NAME** box of the Specify database settings page, type **OperasDB**, in the **SERVER** box, click **New SQL database server**, and then, in the **LOGIN NAME** box, type <*your first name*>.
10. In the **PASSWORD** box of the Specify database settings page, type **Pa$$w0rd**, in the **PASSWORD CONFIRMATION** box, type **Pa$$w0rd**, and then click the **Complete** button.

 **Note:** Windows Azure creates the new website and database to support the Operas website.

11. In the Internet Explorer window, click the **Close** button.

Lesson 2

Deploying an ASP.NET MVC 4 Web Application

Deploying ASP.NET MVC applications is similar to deploying other ASP.NET applications. This process is simple and straightforward. However, you need to update the configuration of the application for use in the production environment. Microsoft Visual Studio includes features that simplify updating the configuration of the application. You can also automate the deployment process by using deployment tools available in Microsoft Visual Studio.

Lesson Objectives

After completing this lesson, you will be able to:

- Reconfigure an ASP.NET MVC application to deploy it on a production web server.
- Use the Bin Deploy method to deploy the .NET Framework libraries along with your web application.
- Use the deployment tools in Visual Studio.

Reviewing Configuration for Production

Each ASP.NET MVC web application includes a Web.config file. In addition to this file, Microsoft Visual Studio creates two variance files to specify deployment transformations to the Web.config file. You can use these variance files for different build configurations. The variance files include:

- *The Web.release.config file*. This file stores the changes that Microsoft Visual Studio applies to the Web.config file, when you compile the application in the Release mode.
- *The Web.debug.config file*. This file stores the changes that Microsoft Visual Studio applies to the Web.config file, when you compile the application in the Debug mode.

While reviewing the configuration for production:

- Include the transformation elements in the following Web.config transformation files for generating resultant Web.config files:
  - Web.release.config
  - Web.debug.config
- Modify the Web.config file by using the **debug**, **xdt:Transform**, and **Insert** attributes

Microsoft Visual Studio can automatically transform the Web.config file for use in different environments. Before you publish the application, you can remove the **debug** attribute from the Web.config file.

The following code shows how to remove the **debug** attribute from the Web.config file by using a deployment transformation.

Removing the Debug Attribute for Deployment

```
<system.web>
  <compilation xdt:Transform="RemoveAttributes(debug)" />
</system.web>
```

The following code shows how to remove the **debug** attribute from the Web.config file by using a deployment transformation.

Removing the Debug Attribute by Using a Deployment Transformation

```
<system.web>
```

```
  <compilation xdt:Transform="RemoveAttributes(debug)" />
</system.web>
```

The preceding code sample uses the **xdt:Transform** attribute to modify the existing Web.config file and produce the published Web.config file. You can also use the **Insert** value to add additional elements to the resultant Web.config file.

For example, the following code shows how to insert a new connection string setting in the resultant Web.config file.

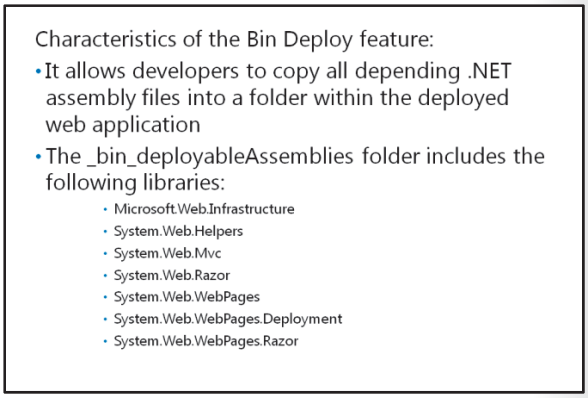
Inserting a Connection String

```
<connectionStrings>
   <add name="DemoConnStr" connectionString="Data Source=|DataDirectory|demo.sdf"
      providerName="System.Data.SqlServerCe.4.0" xdt:Transform="Insert"/>
</connectionStrings>
```

Question: How can you configure the Web.config file for publishing to a production environment without using the Web.release.config configuration file?

Using Bin Deploy

ASP.NET MVC applications depend on a range of .NET assemblies; these assemblies ensure that the application functions as required. These assemblies are dynamic-link library (DLL) files available in your project references. The assembly files are usually already present on the web server because they are included as part of the MVC library installation. However, sometimes the web server administrators do not install all .NET Framework assemblies. Microsoft Visual Studio provides the Bin Deploy feature to copy all depending assembly files into a folder within the deployed web application. This feature helps deploy dependencies on the server.



You can use the **Add Deployable Dependencies** feature to generate the _bin_deployableAssemblies folder. This folder includes the following libraries:

- Microsoft.Web.Infrastructure
- System.Web.Helpers
- System.Web.Mvc
- System.Web.Razor
- System.Web.WebPages
- System.Web.WebPages.Deployment
- System.Web.WebPages.Razor

You can choose a specific set of libraries to include in Microsoft Visual Studio. This practice enables you to ensure that Microsoft Visual Studio deploys the:

- Right assemblies.
- Right versions of the assemblies.

You can then copy the library to the bin folder of the production server, after adding it to the library of your project.

Question: Why should you use the deployable assembly as part of the deployment plan?

Using Visual Studio 2012 Deployment Tools

The Publish feature helps generate a copy of the web application when the application is ready for deployment in the production environment. The feature includes three methods to deploy the application:

- File Share
- FTP
- Web Deploy

The Publish feature:

- Generates a copy of the web application, when the application is ready for deployment in the production environment
- Provides three methods to deploy the application
  - File Share
  - FTP
  - Web Deploy

On the Publish web application page, you can publish the application by using File Share or FTP. You can use File Share for servers to which you have direct network access. Alternatively, you can use File Share to publish a web application first to a local folder. Then, you can manually copy the files to the web server. File Share also allows you to specify a profile to store all deployment options. This practice eliminates the need to specify the deployment options every time you publish the application.

If you select **Web Deploy**, you can specify database settings, such as connection strings, in the publishing wizard. These settings will override the settings in the Web.config file in the target environment. Web Deploy also includes tools to automatically update the schema of the database. These tools eliminate the need to manually update the schema. The publishing tool detects all schema changes and generates scripts to apply those changes to the published database. If your development and production web servers are isolated from each other, you can run these generated scripts on the production environment. This practice replicates schema changes on a server where Microsoft Visual Studio is not installed.

The following image shows the Publish web application page.

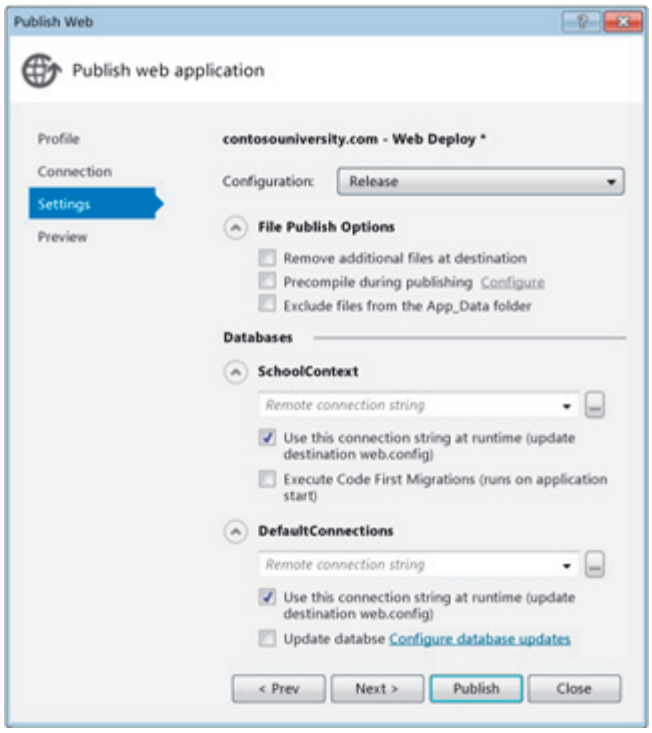


FIGURE 16.1:THE PUBLISH WEB APPLICATION PAGE

Question: What is the benefit of using the Web Deploy publish tool?

Demonstration: How to Deploy a Website to Windows Azure

In this demonstration, you will see how to obtain a publish profile from Windows Azure and use it to publish a website from Visual Studio.

Demonstration Steps

1. On the taskbar, click the **Internet Explorer** icon.
2. In the Address bar of the Internet Explorer window, type **http://www.windowsazure.com**, and then click the **Go to** button.
3. In the upper-right corner of the Windows Azure: Microsoft's Cloud Platform | Cloud Hosting | Cloud Services page, click **PORTAL**.
4. On the Sign in to your Microsoft account page, in the **Microsoft account** box, type <*your username*>, in the **Password** box, type **Pa$$w0rd**, and then click **Sign in**.
5. In the left pane of the Windows Azure page, click **WEB SITES**.
6. In the **NAME** column of the Web Sites – Windows Azure page, click <*your username*>**operas**.
7. On the Web Sites – Windows Azure page, click **DASHBOARD**.

Note: If the quick start page appears, repeat the steps 5 and 6 before proceeding to the next step.

8. In the quick glance section of the Web Sites – Windows Azure page, click the **Download publish profile** link.
9. In the Navigation bar, click **Save**, and then click the **Close** button.
10. On the taskbar, click the **Microsoft Visual Studio** icon.

11. In the Solution Explorer pane of the OperasWebSite - Microsoft Visual Studio window, right-click **OperasWebSite**, and then click **Publish**.
12. On the **Profile** page of the Publish Web wizard, click **Import**.
13. In the **Import Publish Settings** dialog box, click <*your username*>**operas.azurewebsites.net.PublishSettings**, and then click **Open**.
14. On the **Connection** page of the Publish Web wizard, click **Validate Connection**.
15. If the **Certificate Error** dialog box appears, click **Accept**.
16. On the **Connection** page, click **Next**.
17. On the **Settings** page, click **Next**.
18. On the **Preview** page, click **Start Preview**.
19. On the **Preview** page, click **Publish**.

Note: Visual Studio publishes the website. This process can take several minutes. When the publish operation is complete, the website is displayed in the Internet Explorer window.

20. If the **Certificate Error** dialog box appears, click **Accept**.
21. In the Internet Explorer window, if the **Server Error in '/' Application** error is displayed, click the **Refresh** button.

Note: If the Operas I Have Seen page does not appear, you need to re-publish the **OperasWebSite** project.

22. On the Operas I Have Seen page, click **All Operas**.
23. In the Navigation bar, if the message **Intranet settings are turned off by default.** is displayed, click **Turn on Intranet settings**.
24. If the message **Are you sure you want to turn on intranet-level security settings?** appears, click **Yes**.
25. On the Index of Operas page, click the **Details** link corresponding to **Cosi Fan Tutte**.
26. On the Opera: Cosi Fan Tutte page, click the **Back to List** link.
27. On the Index of Operas page, click the **Details** link corresponding to **Nixon in China**.
28. In the Internet Explorer window, click the **Close** button.
29. In the **OperasWebSite - Microsoft Visual Studio** window, click the **Close** button.

Lab: Deploying ASP.NET MVC 4 Web Applications

Scenario

You have completed the development and testing of the photo sharing application. Your managers and senior developers have signed off the project, and have requested you to deploy the application to the Adventure Works Windows Azure account.

Objectives

After completing this lab, you will be able to:

- Prepare an MVC web application for deployment.
- Deploy an MVC web application to Windows Azure.

Lab Setup

Estimated Time: 45 minutes

Virtual Machine: **20486B-SEA-DEV11**

User name: **Admin**

Password: **Pa$$w0rd**

Note: In Hyper-V Manager, start the **MSL-TMG1** virtual machine if it is not already running.

In this lab, you will create a web application in Windows Azure that runs in Free mode. Free mode is an excellent tool for testing and running small, non-intensive web applications. However, there is a limit of 165 megabytes (MB) of data transfer per day on the Free mode tool. After completing Exercise 2, you are encouraged to further test the deployed web application. However, if you do a lot of extra testing, you may encounter the limit and the application may become unavailable.

Also, before initiating this lab, perform the following steps:

1. Apply the Snapshot of the virtual machine, **20486B-SEA-DEV11**, that was taken after completing the lab in module 13.
2. Enable the **Allow NuGet to download missing packages during build** option, by performing the following steps:

1. On the **TOOLS** menu of the Microsoft Visual Studio window, click **Options**.
2. In the navigation pane of the **Options** dialog box, click **Package Manager**.
3. Under the Package Restore section, select the **Allow NuGet to download missing packages during build** checkbox, and then click **OK**.

Exercise 1: Deploying a Web Application to Windows Azure

Scenario

In this exercise, you will:

- Reconfigure the Photo Sharing application for release deployment.
- Configure the **Entity Framework initializer** class, which fills the database with initial data, and ensure that the build configuration and connection strings are correct.
- Create a new web application in Windows Azure and deploy the Photo Sharing application to the new site.

The main tasks for this exercise are as follows:

1. Prepare the Photo Sharing application project for deployment.
2. Create a new website in Windows Azure.
3. Deploy the Photo Sharing application.

▶ Task 1: Prepare the Photo Sharing application project for deployment.

1. Start the virtual machine, and log on with the following credentials:
   - Virtual machine: **20486B-SEA-DEV11**
   - User name: **Admin**
   - Password: **Pa$$w0rd**
2. Open the Photo Sharing application solution from the following location:
   - File location: **Allfiles (D):\Labfiles\Mod16\Starter\PhotoSharingApplication**.
3. In the **Build** properties of the **PhotoSharingApplication** project, select the **Release** configuration.
4. Log on to the Windows Azure portal by using the following information:
   - URL: **http://www.windowsazure.com**
   - User name: <*Your Windows Live account name*>
   - Password: <your password>
5. Copy the **ADO.NET** connection string for the **PhotoSharingDB** to the clipboard.
6. In the Web.config file, paste the connection string into the **connectionString** attribute of the **PhotoSharingDB** connection string. Set the password in the pasted connection string to **Pa$$w0rd**.
7. Log on to the Bing Maps Account Center web application by using the following information:
   - URL: **https://www.bingmapsportal.com**
   - User name: <*Your Windows Live account name*>
   - Password: <*Your Windows Live account password*>
8. Copy the Bing Maps key.
9. Replace the text, **{Your Bing Key}**, in the MapDisplay.js file, with the copied key.
10. Save all the changes.

▶ Task 2: Create a new website in Windows Azure.

1. In Windows Azure, create a new website by using the following information:
   - URL: <*your username*>**PhotoSharing**
   - Region: <*a region near you*>
   - Database: **PhotoSharingDB**
   - Database connection string name: **PhotoSharingDB**.
   - Database user: <*Your first name*>
   - Password: **Pa$$w0rd**

▶ Task 3: Deploy the Photo Sharing application.

1. In Windows Internet Explorer, download the publish profile for the <*your username*>**PhotoSharing** web application.

2. In Microsoft Visual Studio, start the Publish Web wizard and import the publish profile you just downloaded.
3. For the **Photo Sharing Context** database connection, select the **PhotoSharingDB** connection string.
4. For the **Photo Sharing DB** database connection, select the **PhotoSharingDB** connection string.
5. For the **UsersContext** database connection, select the **PhotoSharingDB** connection string.
6. Preview the files that require changes, and then publish the web application.

Results: After completing this exercise, you will be able to prepare an ASP.NET MVC web application for production deployment, create a web application in Windows Azure, and deploy an MVC web application to Windows Azure.

Exercise 2: Testing the Completed Application

Scenario

You have completed and fully deployed the Photo Sharing web application in Windows Azure. Now, you want to perform some final functionality tests before you confirm the completion of the application to your manager.

The main tasks for this exercise are as follows:

1. Add a photo and a comment.
2. Use the slideshow and favorites options.
3. Test the chat room.

▶ Task 1: Add a photo and a comment.

1. Log on to the **Adventure Works Photo Sharing** web application by using the following information:
   - User name: **David Johnson**
   - Password: **Pa$$w0rd2**
2. Add a new photo to the web application by using the following information:
   - Title: **Test New Photo**
   - Photo: **Allfiles (D):\Labfiles\Mod16\Sample Photos\track.JPG**
   - Description: **This is the first photo added to the deployed web application**
3. Display the new photo and check if the data is displayed correctly.
4. Add a new comment to the new photo by using the following information:
   - Subject: **Functionality Check**
   - Body: **The photo metadata is displayed correctly**

▶ Task 2: Use the slideshow and favorites options.

1. Examine the Slideshow webpage and check that the display of all photos functions as designed.
2. Add three photos to your favorites list.
3. View the **Favorites** slideshow.

▶ Task 3: Test the chat room.

1. Enter the chat room for **Sample Photo 1**.
2. Send a message.
3. Start a new instance of Internet Explorer and browse to the Photo Sharing application by using the following information:
   - URL: **http://**<*yourusername*>**PhotoSharing.azurewebsites.net**
   - User name: **Mark Steele**
   - Password: **Pa$$w0rd**
4. Enter the chat room for **Sample Photo 1**.
5. Send a second message and switch to the first instance of Internet Explorer. Send a third message.
6. Close Internet Explorer and Visual Studio.

Results: After completing this exercise, you will be able to confirm that all the functionalities that you built in the Photo Sharing application run correctly in the Windows Azure deployment.

Question: Why is it unnecessary to use bin deployment in this lab?

Question: In the labs for this course, you used the same Windows Azure SQL Database for both development and production. If you wanted to use separate databases for development and production, but did not want to reconfigure the web application every time you deployed to the development and production web servers, how would you configure the web application?

Module Review and Takeaways

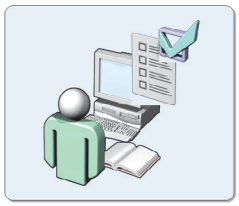
Deployment is usually the last task that developers perform; however, this is the task that developers do not spend much time on. The target environment usually impacts the deployment procedure. In the case of Windows Azure, service packages allow the management portal to deploy the package to the virtual machine that hosts the workload.

To simplify the deployment process, Microsoft Visual Studio consists of a number of tools that help developers generate files for deployment. These tools also help handle the database during deployment. You can also generate the Web.config file based on the environment that it deploys to.

Review Question(s)

Question: You need to create two packages for deployment-for testing and for production environment. This is because of the difference in server name and other environment settings. What should you do?

Course Evaluation



Keep this evaluation topic page if this is the final module in this course. Insert the Product_Evaluation.ppt on this page.

If this is not the final module in the course, delete this page

Your evaluation of this course will help Microsoft understand the quality of your learning experience.

Please work with your training provider to access the course evaluation form.

Microsoft will keep your answers to this survey private and confidential and will use your responses to improve your future learning experience. Your open and honest feedback is valuable and appreciated.

Module 01: Exploring ASP.NET MVC 4

Lab: Exploring ASP.NET MVC 4

Exercise 1: Exploring a Photo Sharing Application

▶ Task 1: Register a user account.

1. In the result pane of the Hyper-V Manager console, in the **Name** list of the **Virtual Machines** area, right-click **20486B-SEA-DEV11**, and then click **Connect**.
2. To log on to **20486B-SEA-DEV11**, click the **Ctrl+Alt+Delete** button.
3. In the **User name** box, type **Admin**, in the **Password** box, type **Pa$$w0rd**, and then click the **Forward** button.
4. On the Windows 8 Start screen, click **Visual Studio 2012**.
5. On the taskbar, right-click the **Start Page - Microsoft Visual Studio** icon, and then click **Pin this program to taskbar**.
6. On the **FILE** menu of the **Start Page – Microsoft Visual Studio** window, point to **Open**, and then click **Project/Solution**.
7. In the **Open Project** dialog box, navigate to **Allfiles (D):\Labfiles\Mod01\PhotoSharingSample**, click **PhotoSharingSample.sln**, and then click **Open**.
8. On the **DEBUG** menu of the **PhotoSharingSample – Microsoft Visual Studio** window, click **Start Without Debugging**.
9. On the Home page of the **http://localhost:**<*yourportnumber*>**/** window, in the upper-right corner, click **Register**.

Note: In the **http://localhost**:<yourportnumber>/ window, if a notification bar appears, click **Don't show this message box again.**

10. On the Register page, in the **User name** box, type <*A user name of your choice*>.
11. On the Register page, in the **Password** box, type <*A password of your choice*>, in the **Confirm password** box, type <*A password of your choice*>, and then click **Register**.

▶ Task 2: Upload and explore photos.

1. On the Home page of the **http://localhost:**<*yourportnumber*>**/**window, click **All Photos**.
2. On the Photo Index page, click the **Details** link corresponding to the "Orchard" image.
3. On the Photo: Orchard page, in the **Subject** box, type **Just a Test Comment**.
4. In the **Body** box, type **This is a Sample**, and then click **Create**.

Note: The comment you have added appears in the Comments section.

5. On the Photo: Orchard page of the **http://localhost:**<*yourportnumber*>**/photo/2** window, click **Add a Photo**.
6. On the Create New Photo page, in the **Title** box, type **Fall Fungi**.
7. In the **Upload new Picture** box, click **Browse**.

8. In the **Choose File to Upload** dialog box, navigate to **Allfiles (D):\ Labfiles\Mod01\Pictures**, click **fungi.JPG**, and then click **Open**.
9. In the **Description** box, type **Sample Text**, and then click **Create**.
10. On the Photo Index page, verify that the newly added photo is displayed.
11. On the Photo Index page, click the **Details** link corresponding to the "Fall Fungi" image.
12. On the Photo: Fall Fungi page, note that the description provided for the photo is updated.

▶ Task 3: Use slideshows.

1. On the Photo: Fall Fungi page of the **http://localhost:**<*yourportnumber*>**/photo/6** window, click the **Slideshow** tab.

Note: You can view all the images on the Photo Index page. Verify that all the images in the gallery are displayed in the slideshow.

2. On the SlideShow page of the **http://localhost:**<*yourportnumber*>**/Photo/SlideShow** window, click **All Photos**.
3. On the Photo Index page, click the **Details** link corresponding to the "Fall Fungi" image.
4. On the Photo: Fall Fungi page, click the **Add to favorites** link.

Note: Verify that the message, "The picture has been added to your favorites", is displayed.

5. At the lower-left corner of the Photo: Fall Fungi page, click the **Back to List** link.
6. On the Photo Index page, click the **Details** link corresponding to the "Orchard" image.
7. On the Photo: Orchard page, click the **Add to favorites** link.

Note: Verify that the message, "The picture has been added to your favorites", is displayed.

8. At the lower-left corner of the Photo: Orchard page, click the **Back to List** link.
9. On the Photo Index page, click the **Details** link corresponding to the "Flower" image.
10. On the Photo: Flower page, click the **Add to favorites** link.

Note: Verify that the message, "The picture has been added to your favorites", is displayed.

11. On the Photo: Flower page of the **http://localhost:**<*yourportnumber*>**/photo/1** window, click **Favorites**.

Note: Verify that all the images that you have added as favorites are displayed in the slideshow.

▶ Task 4: Test the authorization.

1. In the upper-right corner of the SlideShow page, click the **Log Off** link.

Note: Before clicking **All Photos**, on the **Privacy** tab of the **Internet Options** dialog box, click **Clear Sites**, and then click **OK**.
After clearing the sites, press F5, or click the **Refresh** button, before you click **All Photos**.

2. On the Home page, click **All Photos**.
3. On the Photo Index page, click the **Details** link corresponding to the "Fall Fungi" image.
4. On the Photo: Fall Fungi page, in the Comments section, note that the message, "To comment you must log in", is displayed.
5. On the Photo: Fall Fungi page, click **All Photos**.
6. On the Photo Index page, click the **Create New** link.

Note: Ensure that you are not able to view your user name in the web application. If you are able to view your user name, press F5 or click the **Refresh** button, before proceeding to the next step.

Note: You are directed to the Logon page, and prompted to enter the logon credentials.

7. In the Windows Internet Explorer window, click the **Close** button.
8. In the **PhotoSharingSample – Microsoft Visual Studio** window, click the **Close** button.

Results: At the end of this exercise, you will be able to understand the functionality of a photo sharing application, and implement the required application structure in the Adventure Works photo sharing application.

Exercise 2: Exploring a Web Pages Application

▶ Task 1: Create a Web Pages application.

1. On the Windows 8 Start screen, click **Visual Studio 2012**.
2. On the **FILE** menu of the **Start Page - Microsoft Visual Studio** window, point to **New**, and then click **Web Site**.
3. In the navigation pane of the **New Web Site** dialog box, expand **Installed**, expand **Templates**, and then click **Visual C#**.
4. In the result pane of the **New Web Site** dialog box, click **ASP.NET Web Site (Razor v2)**, and then click **OK**.
5. On the **DEBUG** menu of the **WebSite1 – Microsoft Visual Studio** window, click **Start Debugging**.
6. On the home page of the **http://localhost:**<*yourportnumber*>**/** window, click **Contact** to review its contents.
7. In the Windows Internet Explorer window, click the **Close** button.

▶ Task 2: Explore the application structure.

1. In the Solution Explorer pane of the **WebSite1(1) - Microsoft Visual Studio** window, click **Web.config**.
2. In the Web.config code window, locate the **<DbProviderFactories>** element, and then locate the **<add>** element.
3. In the **<add>** element, note that the value of the **description** attribute is **.NET Framework Data Provider for Microsoft SQL Server Compact**.
4. In the Solution Explorer pane of the **WebSite1(1) - Microsoft Visual Studio** window, click **Default.cshtml**.
5. In the Default.cshtml code window, note that the value of Layout is **~/_SiteLayout.cshtml**.
6. In the Solution Explorer pane, click **Contact.cshtml**.
7. In the Contact.cshtml code window, note that the value of Layout is **~/_SiteLayout.cshtml**.

Note: This indicates that the Default.cshtml and the Contact.cshtml files use the same layout file, **~/_SiteLayout.cshtml**.

8. In the Contact.cshtml code window, examine the Razor code and note that there are no links to .css files.
9. In the Solution Explorer pane, click **_SiteLayout.cshtml**.
10. In the _SiteLayout.cshtml code window, in the **HEAD** element, note that there is a link to **~/Content/Site.css**.
11. In the Solution Explorer pane, expand **Content**, and then click **Site.css**.
12. In the Site.css code window, verify the style sheet for _SiteLayout.cshtml.

▶ Task 3: Add simple functionality.

1. In the **WebSite1(1) – Microsoft Visual Studio** window, in the Solution Explorer pane, right-click **WebSite1(1)**, point to **Add**, and then click **Add New Item**.
2. In the **Add New Item – WebSite1(1)** dialog box, click **Web Page (Razor v2)**.

3. In the **Name** box of the **Add New Item – WebSite1(1)** dialog box, type **TestPage.cshtml**, and then click **Add**.
4. In the TestPage.cshtml code window of the **WebSite1(1) – Microsoft Visual Studio** window, in the **BODY** element, type the following code.

```
<h1>This is a Test Page</h1>
```

5. On the **FILE** menu of the **WebSite1(1) – Microsoft Visual Studio** window, click **Save All**.
6. In the Solution Explorer pane of the **WebSite1(1) – Microsoft Visual Studio** window, click **Default.cshtml**.
7. In the Default.cshtml code window, in the **P** element, type the following code.

```
<a href="~/TestPage.cshtml">
  Test Page
</a>
```

8. On the **FILE** menu of the **WebSite1(1) – Microsoft Visual Studio** window, click **Save All**.
9. On the **DEBUG** menu of the **WebSite1(1) – Microsoft Visual Studio** window, click **Start Debugging**.
10. In the **http://localhost:**<*yourportnumber*>**/Default.cshtml** window, click **Test Page** to view its contents.

Note: The text, "This is a Test Page", is displayed on the webpage that you just added.

11. In the Windows Internet Explorer window, click the **Close** button.

▶ Task 4: Apply the site layout.

1. In the Solution Explorer pane of the **WebSite1(1) - Microsoft Visual Studio** window, click **TestPage.cshtml**.
2. In the TestPage.cshtml code window, place the mouse cursor before <!DOCTYPE html>, and then press Enter twice.
3. In the first line of the TestPage.cshtml code window, type the following code block.

```
@{
}
```

4. In the Razor code block of the TestPage.cshtml code window, type the following code.

```
Layout = "~/_SiteLayout.cshtml";
```

5. On the **FILE** menu of the **WebSite1(1) – Microsoft Visual Studio** window, click **Save All**.
6. On the **DEBUG** menu of the **WebSite1(1) – Microsoft Visual Studio** window, click **Start Debugging**.
7. In the **http://localhost:**<*yourportnumber*>**/TestPage.cshtml** window, note that the standard site layout and styles have been applied.
8. In the **http://localhost:**<*yourportnumber*>**/TestPage.cshtml** window, click the **Close** button.
9. In the **WebSite1(1) – Microsoft Visual Studio** window, click the **Close** button.

Results: At the end of this exercise, you will be able to build a simple Web Pages application in Visual Studio.

Exercise 3: Exploring a Web Forms Application

▶ Task 1: Create a Web Forms application.

1. On the taskbar, click the **Start Page - Microsoft Visual Studio** icon.
2. On the **FILE** menu of the **Start Page -Microsoft Visual Studio** window, point to **New**, and then click **Project**.
3. In the left pane of the **New Project** dialog box, expand **Installed**, expand **Templates**, and then expand **Visual C#**.
4. Under Visual C#, click **Web**, and then, in the result pane, click **ASP.NET Web Forms Application**.
5. In the **Name** box, replace **WebApplication1** with **TestWebFormsApplication**, and then click **OK**.
6. On the **DEBUG** menu of the **TestWebFormsApplication – Microsoft Visual Studio** window, click **Start Debugging**.
7. In the **http://localhost:**<*yourportnumber*>**/Default.aspx** window, click **Contact** to review the contents.
8. In the **http://localhost:**<*yourportnumber*>**/Contact.aspx** window, click the **Close** button.

▶ Task 2: Explore the application structure.

1. In the Solution Explorer pane of the **TestWebFormsApplication – Microsoft Visual Studio** window, click **Web.config**.
2. In the Web.config code window, locate the **<configuration>** element, locate the **<connectionStrings>** element, and then locate the **<add>** element.
3. In the **<add>** element, note that the value of the **providerName** attribute is **System.Data.SqlClient**.
4. In the Solution Explorer pane of the **TestWebFormsApplication – Microsoft Visual Studio** window, click **Default.aspx**.
5. In the first line of the Default.aspx code window, note that the value of the **MasterPageFile** attribute is **~/Site.Master**.
6. In the Solution Explorer pane, click **Contact.aspx**.
7. In the first line of the Contact.aspx code window, note that the value of the **MasterPageFile** attribute is **~/Site.Master**.

Note: The Default.aspx and Contact.aspx files are linked to the same layout file.

8. In the Contact.aspx code window, examine the code, and note that there are no links for files with a .css extension.
9. In the Solution Explorer pane of the **TestWebFormsApplication – Microsoft Visual Studio** window, click **Site.Master**.
10. In the Site.Master code window, note that the path for **BundleReference** is **~/Content/css**.
11. In the Solution Explorer pane, click **Bundle.config**.
12. In the Bundle.config code window, note that there are links to files with a .css extension.
13. In the Solution Explorer pane, expand **Content**, and then click **Site.css**.
14. In the Site.css code window, note that the style sheet for Site.Master is displayed.

▶ Task 3: Add simple functionality.

1. In the Solution Explorer pane of the **TestWebFormsApplication – Microsoft Visual Studio** window, right-click **TestWebFormsApplication**, point to **Add**, and then click **Web Form**.
2. In the **Item name** box of the **Specify Name for Item** dialog box, replace **WebForm1** with **TestPage**, and then click **OK**.
3. In the Testpage.aspx code window, in the **DIV** element, type the following code.

```
<h1>This is a Test Page</h1>
```

4. On the **FILE** menu of the **TestWebFormsApplication – Microsoft Visual Studio** window, click **Save All**.
5. In the Solution Explorer pane of the **TestWebFormsApplication – Microsoft Visual Studio** window, click **Default.aspx**.
6. In the Default.aspx code window, in the **P** element, type the following code.

```
<a href="TestPage.aspx">
  Test Page
</a>
```

7. On the **FILE** menu of the **TestWebFormsApplication – Microsoft Visual Studio** window, click **Save All.**
8. On the **DEBUG** menu of the **TestWebFormsApplication – Microsoft Visual Studio** window, click **Start Debugging**.
9. In the **http://localhost:**<*yourportnumber*>**/Default.aspx** window, click the **Test Page** link to view the newly created Web Form page.

Note: The text, "This is a Test Page", is displayed on the Web Form page.

10. In the **http://localhost:**<*yourportnumber*>**/TestPage.aspx** window, click the **Close** button.

▶ Task 4: Apply the master page.

1. In the Solution Explorer pane of the **TestWebFormsApplication – Microsoft Visual Studio** window, click **TestPage.aspx**.
2. In the first line of the Testpage.aspx code window, place the mouse cursor after the word, "Page", press Spacebar, and then type the following code.

```
MasterPageFile="~/Site.master"
```

3. In the TestPage.aspx code window, select the code from <!DOCTYPE html> to </html>, and then press Delete.
4. In the third line of the TestPage.aspx code window, type the following code.

```
<asp:Content runat="server"
   ID="BodyContent"
   ContentPlaceHolderID="MainContent">
<h1>This is a Test Page</h1>
</asp:Content>
```

5. On the **FILE** menu of the **TestWebFormsApplication – Microsoft Visual Studio** window, click **Save All**.

6. On the **DEBUG** menu of the **TestWebFormsApplication – Microsoft Visual Studio** window, click **Start Debugging**.
7. In the **http://localhost:**<*yourportnumber*>**/TestPage.aspx** window, click **Home**.
8. In the **http://localhost:**<*yourportnumber*>**/**window, click the **Test Page** link, and then note that the standard site layout and styles have been applied to the page.
9. In the **http://localhost:**<*yourportnumber*>**/TestPages.aspx** window, click the **Close** button.
10. In the **TestWebFormsApplication – Microsoft Visual Studio** window, click the **Close** button.

Results: At the end of this exercise, you will be able to build a simple Web Forms application in Visual Studio.

Exercise 4: Exploring an MVC Application

▶ Task 1: Create an MVC 4 application.

1. On the taskbar, click the **Start Page - Microsoft Visual Studio** icon.
2. On the **FILE** menu of the **Start Page – Microsoft Visual Studio** window, point to **New**, and then click **Project**.
3. In the navigation pane of the **New Project** dialog box, expand **Installed**, expand **Templates**, and then expand **Visual C#**.
4. In the navigation pane of the **New Project** dialog box, under Visual C#, click **Web**, and then, in the result pane, click **ASP.NET MVC 4 Web Application**.
5. In the **Name** box, replace **MvcApplication** with **TestMVCApplication**, and then click **OK**.
6. In the Select a template section of the **New ASP.NET MVC 4 Project** dialog box, ensure that **Internet Application** is selected, and then click **OK**.
7. On the **DEBUG** menu of the **TestMVCApplication – Microsoft Visual Studio** window, click **Start Debugging**.
8. In the **http://localhost:**<*yourportnumber*>**/** window, click **Contact** to review the contents of the page.
9. In the **http://localhost:**<*yourportnumber*>**/Home/Contact** window, click the **Close** button.

▶ Task 2: Explore the application structure.

1. In the Solution Explorer pane of the **TestMVCApplication – Microsoft Visual Studio** window, click **Web.config**.
2. In the Web.config code window, locate the **<configuration>** element, locate the **<connectionStrings>**element, and then locate the **<add>** element.
3. In the **<add>** element, note that the value for the **providerName** attribute is **System.Data.SqlClient**.
4. In the Solution Explorer pane, expand **Views**, and then click **_ViewStart.cshtml**.
5. In the _ViewStart.cshtml code window, note that the value for Layout is **~/Views/Shared/_Layout.cshtml**.
6. In the Solution Explorer pane, under Views, expand **Home**, and then click **Contact.cshtml**.
7. In the Contact.cshtml code window, note that there are no files with the .css extension.
8. In the Solution Explorer pane, under Views, collapse **Home**, expand **Shared**, and then click **_Layout.cshtml.**
9. In the _Layout.cshtml code window, note that there are calls to the Styles helper.
10. In the Solution Explorer pane, expand **Content**, and then click **Site.css**.
11. In the Site.css code window, note that the style sheet for _Layout.cshtml is displayed.

▶ Task 3: Add simple functionality.

1. In the Solution Explorer pane of the **TestMVCApplication – Microsoft Visual Studio** window, expand **Home**.
2. In the Solution Explorer pane, right-click **Home**, point to **Add**, and then click **View**.
3. In the **View name** box of the **AddView** dialog box, replace **View1** with **TestPage**, clear the **Use a layout or master page** check box, and then click **Add**.

4. In the **DIV** element of the TestPage.cshtml code window, type the following code.

```
<h1>This is a Test Page</h1>
```

5. On the toolbar, click the **Save** button.
6. In the Solution Explorer pane, expand **Controllers**, and then click **HomeController.cs**.
7. In the HomeController.cs code window, in the **HomeController** class, type the following code.

```
public ActionResult TestPage ()
{
   return View();
}
```

8. On the toolbar, click the **Save** button.
9. In the Solution Explorer pane, under Home, click **Index.cshtml**.
10. In the Index.cshtml code window, in the first **P** element, type the following code.

```
<a href="~/Home/TestPage">
  Test Page
</a>
```

11. On the **FILE** menu of the **TestMVCApplication – Microsoft Visual Studio** window, click **Save All**.
12. On the **DEBUG** menu of the **TestMVCApplication – Microsoft Visual Studio** window, click **Start Debugging**.
13. In the **http://localhost:**<*portnumber*>**/** window, click the **Test Page** link to view the newly created page.

Note: The text, "This is a Test Page", is displayed on the newly created page.

14. In the **http://localhost:**<*portnumber*>**/home/TestPage** window, click the **Close** button.

▶ Task 4: Apply the site layout.

1. In the Solution Explorer pane of the **TestMVCApplication – Microsoft Visual Studio** window, expand **Views**, expand **Home**, and then click **TestPage.cshtml**.
2. In the TestPage.cshtml code window, remove the following line of code.

```
Layout = null;
```

3. In the TestPage.cshtml code window, delete all tags, except the following.

```
<h1>This is a Test Page</h1>
```

4. On the toolbar, click the **Save** button.
5. On the **FILE** menu of the **TestMVCApplication – Microsoft Visual Studio** window, click **Save All**.
6. On the **DEBUG** menu of the **TestMVCApplication – Microsoft Visual Studio** window, click **Start Debugging**.
7. In the **http://localhost:**<*yourportnumber*>**/** window, click the **Test Page** link.
8. In the **http://localhost:**<*yourportnumber*>**/home/TestPage** window, note that the standard site layout and styles have been applied.

9. In the **http://localhost:**<*yourportnumber*>**/home/TestPage** window, click the **Close** button.
10. In the **TestMVCApplication - Microsoft Visual Studio** window, click the **Close** button.

Results: At the end of this exercise, you will be able to build a simple MVC application in Visual Studio.

Module 02: Designing ASP.NET MVC 4 Web Applications

Lab: Designing ASP.NET MVC 4 Web Applications

Exercise 1: Planning Model Classes

▶ Task 1: Examine the initial investigation.

1. In the result pane of the Hyper-V Manager console, in the **Name** list of the **Virtual Machines** area, right-click **20486B-SEA-DEV11**, and then click **Connect**.
2. To log on to **20486B-SEA-DEV11**, click the **Ctrl+Alt+Delete** button.
3. In the **User name** box, type **Admin**, in the **Password** box, type **Pa$$w0rd**, and then click the **Forward** button.
4. On the Windows 8 Start screen, click **Desktop**.
5. On the taskbar, click the **File Explorer** icon.
6. In the **Libraries** window, navigate to **Allfiles (D):\Labfiles\Mod02**, and then double-click **InitialInvestigation.docx**.
7. On the **View** tab of the **InitialInvestigation.docx - Microsoft Word** window, in the **Show** group, ensure that the **Navigation Pane** check box is selected.
8. In the Navigation pane, click **Introduction** to view the contents of the Introduction section.
9. In the Navigation pane, click **General Description of the Photo Sharing Application** to view the contents.
10. In the Navigation pane, click **Use Cases** to view the contents.
11. In the Navigation pane, under Use Cases, view all the Use Cases in the document.
12. In the InitialInvestigation.docx - Microsoft Word window, review Figure 1: Use Case Summary.
13. In the **InitialInvestigation.docx - Microsoft Word** window, click the **Close** button.

▶ Task 2: Plan the photo model class.

1. On the taskbar, click the **File Explorer** icon.
2. In the **D:\Labfiles\Mod02** window, double-click **DetailedPlanningDocument.docx**.
3. In the **DetailedPlanningDocument.docx - Microsoft Word** window, locate the MVC Model section.
4. In the MVC Model section, locate the **Table 1: MVC Model** table.
5. In the first row of the **Model Class** column, type <*Model class name1*>.
6. In the first row of the **Description** column, type <*Description of the model class*>.
7. In the **Table 1: MVC Model** table, in the first row of the **Properties** column, type <*Property1*>.
8. Add other properties to the model class.
9. In the **Table 1: MVC Model** table, in the first row of the **Data Types** column, type <*Data type1*>.
10. Add other data types corresponding to the photo properties.
11. In the **Model Class** column of the **Table 1: MVC Model** table, select all the rows adjacent to the properties you have added, right-click, and then click **Merge Cells**.

12. In the **Description** column of the **Table 1: MVC Model** table, select all the rows adjacent to the properties you have added, right-click, and then click **Merge Cells**.
13. On the Quick Access Toolbar, click **Save**.
14. On the taskbar, move the pointer to the lower-left corner, and then click the **Start** icon.
15. On the Windows 8 Start screen, click **Microsoft Visio 2010**.
16. On the **File** tab of the **Microsoft Visio** window, click the **New** tab, and then, in the Template Categories section, click **Software and Database**.
17. In the Choose a Template section, click **UML Model Diagram**, click **Metric Units**, and then, in the result pane, click **Create**. If prompted to add a new template, click **Yes**.
18. In the Shapes pane of the **Drawing1 - Microsoft Visio** window, ensure **UML Static Structure (Metric)** is selected.
19. In the **UML Static Structure** stencil, drag the **Class** shape onto the drawing page.
20. On the drawing page, right-click the **Class** shape, and then click **Properties**.
21. In the **Name** box of the **UML Class Properties** dialog box, type <*Model name1*>.
22. In the Categories pane of the **UML Class Properties** dialog box, click **Attributes**.
23. In the **Attributes** table, in the first row of the **Attribute** column, type <*Property1*>.
24. In the first row of the **Type** column, click <*Data type1*>.
25. Add the attributes for all the properties you added, and then click **OK**.
26. On the Quick Access Toolbar of the **Drawing1 - Microsoft Visio** window, click **Save**.
27. In the **Save As** dialog box, navigate to **Allfiles (D):\Labfiles\Mod02**, in the **File name** box, type **PhotoSharingLDM**, and then click **Save**.

▶ Task 3: Plan the comment model class.

1. On the taskbar, click the **Microsoft Word** icon.
2. In the **DetailedPlanningDocument.docx - Microsoft Word** window, locate the MVC Model section.
3. In the **DetailedPlanning Document.docx - Microsoft Word** window, locate the **Table 1: MVC Model** table.
4. In the second row of the **Model Class** column, type <*Model name2*>.
5. In the second row of the **Description** column, type <*Description of the model*>.
6. In the **Properties** column of the **Table 1: MVC Model** table, add the properties corresponding to the model class you created.
7. In the **Data Types** column of the **Table 1: MVC Model** table, add data types corresponding to the comment properties.
8. In the **Model Class** column of the **Table 1: MVC Model** table, select all the rows adjacent to the properties you added, right-click, and then click **Merge Cells**.

 **Note:** The rows in the **Model Class** column should only be merged if you have created more than one property.

9. In the **Description** column, select all the rows adjacent to the properties you added, right-click, and then click **Merge Cells**.

Note: The rows in the **Description** column should only be merged if you have created more than one property.

10. On the Quick Access Toolbar, click **Save**.
11. On the taskbar, click the **PhotoSharingLDM.vsd - Microsoft Visio 2010** icon.
12. In the Shapes pane of the **PhotoSharingLDM.vsd - Microsoft Visio** window, ensure that the **UML Static Structure (Metric)** stencil is selected.
13. In the **UML Static Structure** stencil, drag the **Class** shape onto the drawing page.
14. On the drawing page, right-click the **Class** shape, and then click **Properties**.
15. In the **Name** box of the **UML Class Properties** dialog box, type <*Model name2*>.
16. In the Categories pane of the **UML Class Properties** dialog box, click **Attributes**.
17. In the **Attributes** table, in the first row of the **Attribute** column, type the name of the property you added.
18. In the first row of the **Type** column, select the corresponding data type.
19. Add the attributes for all the properties you added, and then click **OK**.
20. On the **Home** tab of the **PhotoSharingLDM.vsd - Microsoft Visio** window, in the **Tools** group, click **Connector**.
21. On the drawing page, move the mouse pointer over one of the model classes, click the blue cross, and then drag the connector to the second model class.
22. On the drawing page, right-click the connector, and then click **Shape Display Options**.
23. In the End Options section of the **UML Shape Display Options** dialog box, clear the **First end name** and the **Second end name** check boxes, and then click **OK**.
24. On the drawing page of **PhotoSharingLDM.vsd - Microsoft Visio** window, right-click the connector, and then click **Properties**.
25. In the Categories pane of the **UML Association Properties** dialog box, ensure that Association is selected.
26. In the **Association Ends** table, for each end of the connector, select an appropriate value for the **Multiplicity** column, and then click **OK**.

Note: The values you choose should be determined by the number of photos that can be associated with a comment and the number of comments that can be associated with a photo.

27. On the Quick Access Toolbar of the **PhotoSharingLDM.vsd - Microsoft Visio** window, click **Save**.

Results: After completing this exercise, you will be able to create proposals for a model, and configure the properties and data types of the model classes.

Exercise 2: Planning Controllers

▶ Task 1: Plan the photo controller.

1. On the taskbar, click the **Microsoft Word** icon.
2. In the **DetailedPlanningDocument.docx - Microsoft Word** window, locate the MVC Controllers section.
3. In the **DetailedPlanning Document.docx - Microsoft Word** window, locate the **Table 2: MVC Controllers** table.
4. In the first row of the **Controller** column, type <*Controller1*>.
5. In the first row of the **Action** column, type <*Action1*>.
6. Add other required actions for the controller for photos.
7. In the first row of the **Description** column, type <*Description1*>.
8. Add other descriptions to the subsequent rows of the **Action** column.

Note: Describe when each action is run, what it does, and any views it returns.

9. In the **Controller** column of the **Table 2: MVC Controllers** table, select all the rows adjacent to the **Action** column, right-click, and then click **Merge Cells**.
10. On the Quick Access Toolbar, click **Save**.

▶ Task 2: Plan the comment controller.

1. In the **Table 2: MVC Controllers** table, in the second row of the **Controller** column, type <*Controller2*>.
2. In the **Action** column, corresponding to <*Controller2*>, type <*Action1*>.
3. Add other required actions in the subsequent rows of the **Action** column.
4. In the **Description** column, type the description of the corresponding actions.
5. In the **Controller** column of the **Table 2: MVC Controllers** table, select all the adjacent rows to the actions you planned, right-click, and then click **Merge Cells**.
6. On the Quick Access Toolbar, click **Save**.

Results: After completing this exercise, you will be able to create proposals for controllers and configure their properties and data types.

Exercise 3: Planning Views

▶ Task 1: Plan the single photo view.

1. In the **DetailedPlanningDocument.docx - Microsoft Word** window, locate the MVC Views section, and then locate the **Table 3: MVC Views** table.
2. In the first row of the **Controller** column, type <*Controller1*>.
3. Add other required controllers to the **Table 3: MVC Views**.
4. In the first row of the **View** column, type <*View1*>.
5. In the subsequent rows of the **View** column, add other required views corresponding to the controllers.
6. In the **Description** column, type the description of the corresponding views.
7. In the **Controller** column of the **Table 3: MVC Views** table, select all the rows adjacent to the actions you planned for <*Controller1*>, right-click, and then click **Merge Cells**.
8. In the **Controller** column, merge the rows corresponding to the other controllers.
9. On the Quick Access Toolbar, click **Save**.
10. On the taskbar, click the **PhotoSharingLDM.vsd - Microsoft Visio** icon.
11. On the **File** tab of the **PhotoSharingLDM.vsd - Microsoft Visio** window, click the **New** tab.
12. In the Template Categories section of the **New** tab, click **Software and Database**.
13. In the Choose a Template section, click **Wireframe Diagram**, and then, in the result pane, click **Create**.
14. In the Shapes pane of the **Drawing1 - Microsoft Visio** window, ensure **Dialogs (Metric)** is selected.
15. In the **Dialogs (Metric)** stencil, drag the **Application Form** shape to the drawing page.
16. On the drawing page, resize the new shape to fill most of the page.
17. In the **Application Form** shape, double-click **Application Title**, and then type the title you want to be displayed.
18. In the **Dialogs (Metric)** stencil, drag the **Panel** shape to the **Application Form** shape, and drop it under the title.
19. Resize the panel to fill the width of the **Application Form** shape.
20. Resize the panel to the same height as the **Application Form** shape.
21. In the **Shapes** pane, click **Controls (Metric)**.
22. In the **Controls (Metric)** stencil, drag a **Hyperlink** shape to the panel.
23. In the **Hyperlinks** dialog box, click **Cancel**.
24. On the drawing page of the **Drawing2 - Microsoft Visio** window, double-click **Hyperlink**, and then type **Home**.
25. Add links for **Gallery**, **Add a Photo**, and **Slideshow** on the panel.
26. In the **Shapes** pane, click **Dialogs (Metric)**.
27. In the **Dialogs (Metric)** stencil, drag a **Panel** shape to the **Application Form** shape, and drop the panel beneath the menu.
28. Resize the panel to display a large photo.

29. In the **Shapes** pane, click **Web and Media Icons (Metric)**.
30. In the **Web and Media Icons (Metric)** stencil, drag a **Photos** shape to the panel you created.

Note: Add other elements to the wireframe diagram that you feel are necessary based on your reading of the **InitialInvestigation** document. Steps will vary, depending on the elements you feel are necessary.

31. On the Quick Access Toolbar of the **Drawing2 - Microsoft Visio** window, click **Save**.
32. In the **Save As** dialog box, navigate to **Allfiles (D):\Labfiles\Mod02**, in the **File name** box, type **SinglePhotoWireframe**, and then click **Save**.

▶ Task 2: Plan the gallery view.

1. On the **File** tab of the **SinglePhotoWireframe.vsd - Microsoft Visio** window, click the **New** tab.
2. In the Template Categories section of the **New** tab, click **Software and Database**.
3. In the Choose a Template section, click **Wireframe Diagram**, and then, in the result pane, click **Create**.
4. In the **Shapes** pane of the **Drawing3 - Microsoft Visio** window, ensure **Dialogs (Metric)** is selected.
5. In the **Dialogs (Metric)** stencil, drag an **Application Form** shape to the drawing page.
6. On the drawing page, resize the new shape to fill most of the page.
7. In the **Application Form** shape, double-click **Application Title**, and type the title you want to be displayed.
8. In the **Dialogs (Metric)** stencil, drag the **Panel** shape to the **Application Form** shape, and drop it under the title.
9. Resize the panel to fill the width of the **Application Form** shape.
10. Resize the panel to be approximately the same height as the title.
11. In the **Shapes** pane, click **Controls (Metric)**.
12. In the **Controls (Metric)** stencil drag a **Hyperlink** shape to the panel.
13. In the **Hyperlinks** dialog box, click **Cancel**.
14. On the drawing page of the **Drawing3 - Microsoft Visio** window, double-click **Hyperlink**, and then type **Home**.
15. On the panel, add **Hyperlinks** with the following labels: **Gallery**, **Add a Photo**, and **Slideshow**.
16. In the **Shapes** pane, click **Dialogs (Metric)**.
17. In the **Dialogs (Metric)** stencil, drag a **Panel** shape to the **Application Form** shape, and drop the panel beneath the menu.
18. Resize the panel to display a thumbnail photo.
19. In the **Shapes** pane, click **Web and Media Icons (Metric)**.
20. In the **Web and Media Icons (Metric)** stencil, drag a **Photos** shape to the panel you created.

Note: You can add multiple panels to display many thumbnails in the Gallery view.

Note: Add other elements to the wireframe diagram that you feel are necessary based on your reading of the **InitialInvestigation** document.

Steps will vary, depending on the elements you feel are necessary.

21. On the Quick Access Toolbar, click **Save**.
22. In the **Save As** dialog box, navigate to **Allfiles (D):\Labfiles\Mod02**, in the **File name** box, type **PhotoGalleryWireframe**, and then click **Save**.

Results: After completing this exercise, you will create proposals for views and their layouts.

Exercise 4: Architecting an MVC Web Application

▶ Task 1: Hosting options.

1. On the taskbar, click the **DetailedPlanningDocument(docx) - Microsoft Word** icon.
2. In the **DetailedPlanningDocument(docx) - Microsoft Word** window, locate the Hosting Recommedations section, and then locate the Web Server section.
3. Under Web Server, type the recommended host and web server configuration**.**

▶ Task 2: Choose a data store.

1. In the Database section of the **DetailedPlanningDocument(docx) - Microsoft Word** window, type the recommended database and server configuration.
2. On the Quick Access Toolbar, click **Save**.

Results: After completing this exercise, you will be able to create proposals for hosting arrangements.

Module 03: Developing ASP.NET MVC 4 Models

Lab: Developing ASP.NET MVC 4 Models

Exercise 1: Creating an MVC Project and Adding a Model

▶ Task 1: Create a new MVC project.

1. In the result pane of the Hyper-V Manager console, in the **Name** list of the **Virtual Machines** area, right-click **20486B-SEA-DEV11**, and then click **Connect**.
2. To log on to **20486B-SEA-DEV11**, click the **Ctrl+Alt+Delete** button.
3. In the **User name** box, type **Admin**, in the **Password** box, type **Pa$$w0rd**, and then click the **Submit** button.
4. On the Windows 8 Start screen, click **Visual Studio** 2012.
5. On the **File** menu of the **Start Page - Microsoft Visual Studio** window, point to **New**, and then click **Project**.
6. In the left pane of the **New Project** dialog box, expand **Installed**, expand **Templates**, and then expand **Visual C#**.
7. Under Visual C#, click **Web**, and then, in the result pane, click **ASP.NET MVC 4 Web Application**.
8. In the **Name** box of the **New Project** dialog box, type **PhotoSharingApplication**.
9. In the **Location** box, ensure that the location specified is **Allfiles (D):\Labfiles\Mod03**.
10. In the **New Project** dialog box, ensure that the **Create directory for solution** check box is selected, and then click **OK**.
11. In the **Select a template** list of the **New ASP.NET MVC 4 Project** dialog box, click **Empty**, and then click **OK**.

▶ Task 2: Add a new MVC model.

1. In the Solution Explorer pane of the **PhotoSharingApplication - Microsoft Visual Studio** window, right-click **Models**, point to **Add**, and then click **Class**.
2. In the **Name** box of the **Add New Item - PhotoSharingApplication** dialog box, type **Photo**, and then click **Add**.
3. In the Solution Explorer pane of the **PhotoSharingApplication - Microsoft Visual Studio** window, right-click **Models**, point to **Add**, and then click **Class**.
4. In the **Name** box of the **Add New Item - PhotoSharingApplication** dialog box, type **Comment**, and then click **Add**.

Results: After completing this exercise, you will be able to create an MVC 4 web application and add model classes to the web application.

Exercise 2: Adding Properties to MVC Models

▶ Task 1: Add properties to the Photo model class.

1. In the Solution Explorer pane of the PhotoSharingApplication - Microsoft Visual Studio window, click Photo.cs.
2. In the Photo class of the Photo.cs code window, type the following code, and then press Enter.

```
public int PhotoID { get; set; }
```

3. Place the mouse cursor at the end of the **PhotoID** property code, press Enter, and then type the following code.

```
public string Title { get; set; }
```

4. Place the mouse cursor at the end of the **Title** property code, press Enter, and then type the following code.

```
public byte[] PhotoFile
   { get; set; }
```

5. Place the mouse cursor at the end of the **PhotoFile** property code, press Enter, and then type the following code.

```
public string ImageMimeType
   { get; set; }
```

6. Place the mouse cursor at the end of the **ImageMimeType** property code, press Enter, and then type the following code.

```
public string Description
   { get; set; }
```

7. Place the mouse cursor at the end of the **Description** property code, press Enter, and then type the following code.

```
public DateTime CreatedDate
   { get; set; }
```

8. Place the mouse cursor at the end of the **CreatedDate** property code, press Enter, and then type the following code.

```
public string UserName
   { get; set; }
```

▶ Task 2: Add properties to the Comment model class.

1. In the Solution Explorer pane of the PhotoSharingApplication - Microsoft Visual Studio window, click Comment.cs.
2. In the Comment class of the Comment.cs code window, type the following code, and then press Enter.

```
public int CommentID { get; set; }
```

3. Place the mouse cursor at the end of the **CommentID** property code, press Enter, and then type the following code.

```
public int PhotoID { get; set; }
```

4. Place the mouse cursor at the end of the **PhotoID** property code, press Enter, and then type the following code.

```
public string UserName
   { get; set; }
```

5. Place the mouse cursor at the end of the **UserName** property code, press Enter, and then type the following code.

```
public string Subject { get; set; }
```

6. Place the mouse cursor at the end of the **Subject** property code, press Enter, and then type the following code.

```
public string Body { get; set; }
```

▶ Task 3: Implement a relationship between model classes.

1. In the Solution Explorer pane of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Photo.cs**.
2. In the Photo.cs code window, place the mouse cursor at the end of the **UserName** property code, press Enter, and then type the following code.

```
public virtual ICollection<Comment>
   Comments { get; set; }
```

3. In the Solution Explorer pane, click **Comment.cs**.
4. In the Comment.cs code window, place the mouse cursor at the end of the **Body** property code, press Enter, and then type the following code.

```
public virtual Photo Photo
   { get; set; }
```

Results: After completing this exercise, you will be able to add properties to classes to describe them to the MVC runtime. You will also implement a one-to-many relationship between classes.

Exercise 3: Using Data Annotations in MVC Models

▶ Task 1: Add display and edit data annotations to the model.

1. In the Solution Explorer pane of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Photo.cs**.
2. In the Photo.cs code window, place the mouse cursor at the end of the **Title** property code, press Enter, and then type the following code.

```
[DisplayName("Picture")]
```

3. In the Photo.cs code window, place the mouse cursor at the end of the System.web namespace code, press Enter, and then type the following code.

```
using System.ComponentModel;
```

4. Place the mouse cursor at the end of the **ImageMimeType** property code, press Enter, and then type the following code.

```
[DataType(DataType.MultilineText)]
```

5. Place the mouse cursor at the end of the System.ComponentModel namespace code, press Enter, and then type the following code.

```
using System.ComponentModel.DataAnnotations;
```

6. Place the mouse cursor at the end of the **Description** property code, press Enter, and then type the following code.

```
[DataType(DataType.DateTime)]
[DisplayName("Created Date")]
[DisplayFormat(DataFormatString =
   "{0:MM/dd/yy}")]
```

7. In the Solution Explorer pane, click **Comment.cs**.
8. In the Comment.cs code window, place the mouse cursor at the end of the **Subject** property code, press Enter, and then type the following code.

```
[DataType(DataType.MultilineText)]
```

9. Place the mouse cursor at the end of the System.Web namespace code, press Enter, and then type the following code.

```
using System.ComponentModel.DataAnnotations;
```

▶ Task 2: Add validation data annotations to the model.

1. In the Solution Explorer pane of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Photo.cs**.
2. In the Photo.cs code window, place the mouse cursor at the end of the **PhotoID** property code, press Enter, and then type the following code.

```
[Required]
```

3. In the Solution Explorer pane, click **Comment.cs**.

4. In the Comment.cs code window, place the mouse cursor at the end of the **UserName** property code, press Enter, and then type the following code.

```
[Required]
[StringLength(250)]
```

Results: After completing this exercise, you will be able to add property descriptions and data annotations to the two model classes in the MVC web application.

Exercise 4: Creating a New Windows Azure SQL Database

▶ Task 1: Add an Entity Framework Context to the model.

1. In the Solution Explorer pane of the **PhotoSharingApplication - Microsoft Visual Studio** window, right-click **References**, and then click **Manage NuGet Packages**.
2. In the navigation pane of the **PhotoSharingApplication - Manage NuGet Packages** window, click **Online**, click **EntityFramework**, and then click **Install**.
3. On the **License Acceptance** page, click **I Accept**.
4. In the **PhotoSharingApplication - Manage NuGet Packages** window, click **Close**.
5. In the Solution Explorer pane, right-click **Models**, point to **Add**, and then click **Class**.
6. In the **Name** box of the **Add New Item - PhotoSharingApplication** dialog box, type **PhotoSharingContext**, and then click **Add**.
7. In the PhotoSharingContext.cs code window, place the mouse cursor at the end of the System.Web namespace, press Enter, and then type the following code.

```
using System.Data.Entity;
```

8. In the PhotoSharingContext.cs code window, locate the following code.

```
public class PhotoSharingContext
```

9. Append the following code to the existing line of code.

```
: DbContext
```

10. In the **PhotoSharingContext** class, type the following code.

```
public DbSet<Photo> Photos
   { get; set; }
public DbSet<Comment> Comments
   { get; set; }
```

▶ Task 2: Add an Entity Framework Initializer.

1. In the Solution Explorer pane of the PhotoSharingApplication - Microsoft Visual Studio window, right-click **Models**, point to **Add**, and then click **Class**.
2. In the **Name** box of the **Add New Item - PhotoSharingApplication** dialog box, type **PhotoSharingInitializer**, and then click **Add**.
3. In the PhotoSharingInitializer.cs code window, place the mouse cursor at the end of the System.web namespace code, press Enter, and then type the following code.

```
using System.Data.Entity;
using System.IO;
```

4. In the PhotoSharingInitializer.cs code window, locate the following code.

```
public class
   PhotoSharingInitializer
```

5. Append the following to the existing line of code.

```
: DropCreateDatabaseAlways
   <PhotoSharingContext>
```

6. On the taskbar, click the **File Explorer** icon.
7. In the **Libraries** dialog box, navigate to **Allfiles (D):\Labfiles\Mod03\CodeSnippets**, and then double-click **getFileBytes.txt**.
8. In the **getFileBytes.txt - Notepad** window, press Ctrl+A, and then press Ctrl+C.
9. On the taskbar, click the **PhotoSharingApplication - Microsoft Visual Studio** icon.
10. Place the mouse cursor in the PhotoSharingInitializer class of the PhotoSharingInitializer.cs code window, press Enter, and then press Ctrl+V.
11. In a new line of the PhotoSharingInitializer class, type the following code, press Spacebar, and then click **Seed(PhotoSharingContext context)**.

```
override
```

12. In the **Seed** method, place the mouse cursor after the call to base.Seed, press Enter twice, and then type the following code.

```
var photos = new List<Photo>
{
   new Photo {
      Title = "Test Photo",
Description = "Your
         Description",
      UserName = "NaokiSato",
PhotoFile = getFileBytes
         ("\\Images\\flower.jpg"),
ImageMimeType =
         "image/jpeg",
      CreatedDate = DateTime.Today
   }
};
```

13. Place the mouse cursor at the end of the list of Photo objects, press Enter twice, and then type the following code.

```
photos.ForEach(s =>
   context.Photos.Add(s));
context.SaveChanges();
```

14. Place the mouse cursor at the end of the Entity Framework context code, press Enter twice, and then type the following code.

```
var comments = new List<Comment>
{
   new Comment {
      PhotoID = 1,
      UserName = "NaokiSato",
      Subject = "Test Comment",
      Body = "This comment " +
         "should appear in " +
         "photo 1"
   }
};
```

15. Place the mouse cursor at the end of the list of Comment objects, press Enter twice, and then type the following code.

```
comments.ForEach(s =>
   context.Comments.Add(s));
context.SaveChanges();
```

16. In the Solution Explorer pane, click **Global.asax**.
17. In the Global.asax code window, place the cursor at the end of the System.Web.Routing namespace, press Enter, and then type the following code.

```
using System.Data.Entity;
using PhotoSharingApplication.Models;
```

18. In the **Application_Start** method code block, type the following code.

```
Database.SetInitializer
   <PhotoSharingContext>
   (new PhotoSharingInitializer());
```

▶ Task 3: Create a Windows Azure SQL database and obtain a connection string.

1. On the taskbar, click the **Internet Explorer** icon.
2. In the Address bar of the Internet Explorer window, type **http://www.windowsazure.com/** and then press Enter.
3. On the Windows Azure page, click **PORTAL**.
4. In the **Username** box, type <*your Windows Live user name*>.
5. In the **Password** box, type <*your Windows Live password*>, and then click **Sign in**.
6. In the left pane of the Windows Azure page, click **SQL DATABASES**.
7. In the lower pane, click **NEW**.
8. In the **New** dialog box, click **SQL DATABASE**, and then click **CUSTOM CREATE**.
9. In the **Name** box of the **NEW SQL DATABASE - CUSTOM CREATE** page, type **PhotoSharingDB**.
10. In the **SERVER** box, click **New SQL database Server**, and then click the **Next** button.
11. In the **LOGIN NAME** box, type <*your first name*>.
12. In the **LOGIN PASSWORD** box, type **Pa$$w0rd**.
13. In the **LOGIN PASSWORD CONFIRMATION** box, type **Pa$$w0rd**.
14. In the **REGION** box, click <*a region close to you*>, and then click the **Complete** button.

Note: In the sql databases result pane, note that a new database, PhotoSharingDB, has been created.

15. In the **Name** column of the sql databases page, click **PhotoSharingDB**, and then click the **DASHBOARD** tab.
16. In the **quick glance** section, click the **Manage allowed IP addresses** link.
17. In the **RULE NAME** box of the result window, type **First Address Range**.
18. In the **START IP ADDRESS** box, type <*first address in range*>.
19. In the **END IP ADDRESS** box, type <*last address in range*>.
20. In the lower pane of the **SQL Databases – Windows Azure** window, click **Save**.

Note: To add more IP address ranges, you can repeat the steps 18–20.

21. In the **Name** column of the sql databases page, click **PhotoSharingDB**, and then click the **DASHBOARD** tab.
22. In the **quick glance** section, click the **Show connection strings** link.
23. In the **ADO.NET** box of the **Connection Strings** window, select the given text, and then press Ctrl+C.
24. On the taskbar, click the **PhotoSharingApplication - Microsoft Visual Studio icon**.
25. In the Solution Explorer pane, click **Web.config**.
26. In the Web.config code window, place the mouse cursor after the </appsettings> tag, press Enter, and then type the following code.

```
<connectionStrings>
  <add name="PhotoSharingDB"
     connectionString = ""
     providerName =
        "System.Data.SqlClient" />
</connectionStrings>
```

27. In the connectionStrings code block, place the mouse cursor within the quotes after **connectionString =** and then press Ctrl-V.
28. In the pasted content, locate the text **{your_password_here}** and then replace it with **Pa$$w0rd**.
29. On the **Build** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window click **Build Solution**, and then note that the application is built successfully.

Results: After completing this exercise, you will be able to create an MVC application that uses Windows Azure SQL Database as its data store.

Exercise 5: Testing the Model and Database

▶ Task 1: Add a controller and views.

1. In the Solution Explorer pane of the **PhotoSharingApplication - Microsoft Visual Studio** window, right-click **Controllers**, point to **Add**, and then click **Controller**.
2. In the **Controller name** box of the **Add Controller** dialog box, type **PhotoController**.
3. In the **Template** list, click **MVC Controller with read/write actions and views, using Entity Framework**.
4. In the **Model class** list, click **Photo (PhotoSharingApplication.Models)**.
5. In the **Data context class** list, click **PhotoSharingContext (PhotoSharingApplication.Models)**.
6. In the **Views** list, ensure that the **Razor (CSHTML)** view is selected, and then click **Add**.

▶ Task 2: Add an image and run the application.

1. In the **File Explorer** window, navigate to **Allfiles (D):\Labfiles\Mod03\Images**.
2. In the **Images** window, right-click **flower.JPG** and then click **Copy**.
3. On the taskbar, click the **PhotoSharingApplication - Microsoft Visual Studio** icon.
4. In the Solution Explorer pane of the **PhotoSharingApplication - Microsoft Visual Studio** window, right-click **PhotoSharingApplication**, point to **Add**, and then click **New Folder**.
5. In the **NewFolder1** box, type **Images**.
6. Under PhotoSharingApplication, right-click **Images**, and then click **Paste**.
7. In the **PhotoSharingApplication - Microsoft Visual Studio** window, press F5.

Note: The Internet Explorer window is displayed with an error message. The error message is expected because the home page view has not been added.

8. In the Address bar of the Internet Explorer window, append the existing URL with **photo/index**, and then press Enter.

Note: The details of the added image are displayed on the Index page. The scaffold templates do not display the image itself. You will see how to display images in later labs.

9. On the Index page, click the **Details** link.

Note: The details of the added image such as the image type, image description, and image creation date are displayed on the Details page. The scaffold templates do not display the image itself.

Results: After completing this exercise, you will be able to add controllers, views, and images to an MVC web application and test the application by displaying data from a Windows Azure SQL database.

Module 04: Developing ASP.NET MVC 4 Controllers

Lab: Developing ASP.NET MVC 4 Controllers

Exercise 1: Adding an MVC Controller and Writing the Actions

▶ Task 1: Create a photo controller.

1. In the result pane of the Hyper-V Manager console, in the **Name** list of the **Virtual Machines** area, right-click **20486B-SEA-DEV11**, and then click **Connect**.
2. To log on to **20486B-SEA-DEV11**, click the **Ctrl+Alt+Delete** button.
3. In the **User name** box, type **Admin**, in the **Password** box, type **Pa$$w0rd**, and then click the **Forward** button.
4. On the Windows 8 Start screen, click **Desktop**.
5. On the taskbar, click the **File Explorer** icon.
6. In the **Libraries** window, navigate to **Allfiles (D):\Labfiles\Mod04\Starter\PhotoSharingApplication**, and then double-click **PhotoSharingApplication.sln.**
7. In the Solution Explorer pane, expand **PhotoSharingApplication**, right-click **Controllers**, point to **Add**, and then click **Controller**.
8. In the **Controller name** box of the **Add Controller** dialog box, type **PhotoController**, in the **Template** box, ensure that **Empty MVC controller** is selected, and then click **Add**.
9. In the PhotoController.cs code window, place the mouse cursor at the end of the System.Web.MVC namespace, press Enter, and then type the following code.

```
using System.Collections.Generic;
using System.Globalization;
using PhotoSharingApplication.Models;
```

10. In the PhotoController class code block, press Enter, type the following code, and then press Enter.

```
private PhotoSharingContext context =
   new PhotoSharingContext();
```

▶ Task 2: Create the Index action.

1. In the PhotoController.cs code window, replace the **Index** action code with the following code.

```
return View("Index",
   context.Photos.ToList());
```

▶ Task 3: Create the Display action.

1. In the PhotoController.cs code window, place the mouse cursor at the end of the **Index** action code block, press Enter, type the following code, and then press Enter.

```
public ActionResult Display
   (int id)
{
}
```

2. In the **Display** action code block, type the following code, and then press Enter.

```
Photo photo =
   context.Photos.Find(id);
```

3. In the **Display** action code block, type the following code, and then press Enter.

```
if (photo == null)
{
   return HttpNotFound();
}
```

4. In the **Display** action code block, type the following code, and then press Enter.

```
return View("Display", photo);
```

▶ Task 4: Write the Create actions for GET and POST HTTP verbs.

1. In the PhotoController.cs code window, place the mouse cursor at the end of the **Display** action code block, press Enter twice, and then type the following code.

```
public ActionResult Create ()
{
}
```

2. In the **Create** action code block, type the following code, and then press Enter.

```
Photo newPhoto = new Photo();
newPhoto.CreatedDate =
   DateTime.Today;
```

3. In the **Create** action code block, type the following code.

```
return View("Create", newPhoto);
```

4. Place the mouse cursor at the end of the **Create** action code block, press Enter, and then type the following code.

```
[HttpPost]
public ActionResult Create
(Photo photo, HttpPostedFileBase image)
{
}
```

5. In the **Create** action code block that you newly created with HTTP verb POST, type the following code, and then press Enter.

```
photo.CreatedDate = DateTime.Today;
```

6. In the **Create** action code block created with HTTP verb POST, type the following code, and then press Enter.

```
if (!ModelState.IsValid)
{
   return View("Create", photo);
}
else
{
   if (image != null)
   {
      photo.ImageMimeType =
         image.ContentType;
```

```
photo.PhotoFile = new
         byte[image.ContentLength];
      image.InputStream.Read(
         photo.PhotoFile, 0,
         image.ContentLength);
   }
}
```

7. In the **Create** action code block created with HTTP verb POST, immediately after the code you just added, press Enter, type the following code, and then press Enter.

```
context.Photos.Add(photo);
context.SaveChanges();
return RedirectToAction("Index");
```

▶ Task 5: Create the Delete actions for GET and POST HTTP verbs.

1. In the PhotoController.cs code window, place the mouse cursor at the end of the **Create** action code block that you created with HTTP verb POST, press Enter twice, and then type the following code.

```
public ActionResult Delete (int id)
{
}
```

2. In the **Delete** action code block, type the following code, and then press Enter.

```
Photo photo =
   context.Photos.Find(id);
```

3. In the **Delete** action code block, type the following code, and then press Enter.

```
if (photo == null)
{
   return HttpNotFound();
}
```

4. In the **Delete** action code block, type the following code.

```
return View("Delete", photo);
```

5. Place the mouse cursor at the end of the **Delete** action code block, press Enter twice, and then type the following code.

```
[HttpPost]
[ActionName("Delete")]
public ActionResult DeleteConfirmed
   (int id)
{
}
```

6. In the **Delete** action code block that you newly created, type the following code, and then press Enter.

```
Photo photo =
   context.Photos.Find(id);
```

7. In the **Delete** action code block, type the following code.

```
context.Photos.Remove(photo);
context.SaveChanges();
return RedirectToAction("Index");
```

▶ Task 6: Create the GetImage action.

1. In the PhotoController.cs code window, place the mouse cursor at the end of the **Delete** action code block, press Enter twice, and then type the following code.

```
public FileContentResult GetImage
   (int id)
{
}
```

2. In the **GetImage** action code block, type the following code, and then press Enter.

```
Photo photo =
   context.Photos.Find(id);
```

3. In the **GetImage** action code block, type the following code, and then press Enter.

```
if (photo != null)
{
   return File(photo.PhotoFile,
      photo.ImageMimeType);
}
else
{
   return null;
}
```

4. On the toolbar of the PhotoSharingAppliaction - Microsoft Visual Studio window, click the Save Controller\PhotoController.CS (Ctrl+S) button.

Results: After completing this exercise, you will be able to create an MVC controller that implements common actions for the Photo model class in the Photo Sharing application.

Exercise 2: Optional—Writing the Action Filters in a Controller

▶ Task 1: Add an action filter class.

1. In the Solution Explorer pane of the **PhotoSharingApplication - Microsoft Visual Studio** window, right-click **Controllers**, point to **Add**, and then click **Class**.
2. In the **Add New Item - PhotoSharingApplication** dialog box, in the **Name** box, type **ValueReporter**, and then click **Add**.
3. In the ValueReporter.cs code window of the **PhotoSharingApplication - Microsoft Visual Studio** window, place the mouse cursor at the end of System.Web namespace, press Enter, and then type the following code.

```
using System.Diagnostics;
using System.Web.Mvc;
using System.Web.Routing;
```

4. In the ValueReporter.cs code window, locate the following code.

```
public class ValueReporter
```

5. Append the following code to the existing line of code.

```
: ActionFilterAttribute
```

▶ Task 2: Add a logValues method to the action filter class.

1. In the ValueReporter class code block of the ValueReporter.cs code window, press Enter, and then type the following code.

```
private void logValues
   (RouteData routeData)
{
}
```

2. In the **logValues** method code block, type the following code.

```
var controller =
   routeData.Values["controller"];
var action =
   routeData.Values["action"];
string message = string.Format(
   "Controller: {0}; Action: {1}",
   controller, action);
Debug.WriteLine(message,
   "Action Values");
```

3. In the **logValues** method code block, press Enter twice, and then type the following code.

```
foreach (var item in
   routeData.Values)
{
}
```

4. In the **foreach** loop, type the following code.

```
Debug.WriteLine(
   ">> Key: {0}; Value: {1}", item.Key,
   item.Value);
```

▶ Task 3: Add a handler for the OnActionExecuting event.

1. In the ValueReporter class code block, place the mouse cursor before the **logValues** method code block, press Enter, and then type the following code.

```
override
```

2. Press the Spacebar, and then double-click the following code from the list.

```
OnActionExecuting(ActionExecutingContext filterContext)
```

3. In the **OnActionExecuting** event code block, select the following code, and then press Delete.

```
base.OnActionExecuting(filterContext);
```

4. In the **OnActionExecuting** event code block, type the following code.

```
logValues(filterContext.RouteData);
```

5. On the toolbar of the **PhotoSharingApplication - Microsoft Visual Studio** window, click the **Save Controllers\ValueReporter.cs (Ctrl+S)** button.

▶ Task 4: Register the Action Filter with the Photo Controller.

1. In the Solution Explorer pane, under Controllers, click **PhotoController.cs**
2. In the **PhotoController.cs** code window, locate the following code.

```
public class PhotoController
   : Controller
```

3. Place the mouse cursor before the PhotoController class code block, press Enter, and then type the following code.

```
 [ValueReporter]
```

4. On the toolbar of the **PhotoSharingApplication - Microsoft Visual Studio** window, click the **Save Controllers\PhotoController.cs (Ctrl+S)** button.

Results: After completing this exercise, you will be able to create an action filter class that logs the details of actions, controllers, and parameters to the Visual Studio Output window, whenever an action is called.

Exercise 3: Using the Photo Controller

▶ Task 1: Create the Index and Display views.

1. On the **BUILD** menu of the **PhotoSharingApplication – Microsoft Visual Studio** window, click **Build Solution**.
2. In the PhotoController.cs code window, locate the following code, right-click the code, and then click **Add View**.

```
public ActionResult Index()
```

3. In the **Add View** dialog box, ensure that the name in the **View name** box is **Index**.
4. In the **View engine** box, ensure that the **Razor (CSHTML)** view is selected.
5. In the **Add View** dialog box, select the **Create a strongly-typed view** check box.
6. In the **Model class** box, click **Photo (PhotoSharingApplication.Models)**.
7. In the **Scaffold template** box, click **List**.
8. In the **Add View** dialog box, ensure that the **Reference script libraries** and the **Use a layout or master page** check boxes are selected, and then click **Add**.
9. In the Index.cshtml code window, select the following code.

```
@Html.ActionLink("Details", "Details",
   new { id=item.PhotoID } )
```

10. Replace the selected code with the following code.

```
@Html.ActionLink("Display", "Display",   new { id=item.PhotoID } )
```

11. In the Solution Explorer pane, under Controllers, click **PhotoController.cs**.
12. In the PhotoController.cs code window, locate the following code, right-click the code, and then click **Add View.**

```
public ActionResult Display(int id)
```

13. In the **Add View** dialog box, ensure that the name in the **View name** box is **Display**.
14. In the **Views engine** box, ensure that the **Razor (CSHTML)** view is selected.
15. In the **Add View** dialog box, ensure that the **Create a strongly-typed view** check box is selected.
16. In the **Model class** box, ensure that the **Photo (PhotoSharingApplication.Models)** model class is selected.
17. In the **Scaffold template** box, click **Details**.
18. In the **Add View** dialog box, ensure that the **Reference script libraries** and the **Use a layout or master page** check boxes are selected, and then click **Add**.

▶ Task 2: Use the GetImage action in the Display view.

1. In the Display.cshtml code window, locate the following code.

```
<div class="display-field">
@Html.DisplayFor(model =>
      model.Title)
</div>
```

2. Place the mouse cursor at the end of the **model.Title** property code block, press Enter, and then type the following code.

```
@if (Model.PhotoFile != null) {
}
```

3. In the **if** statement code block, type the following code.

```
<img width="800" src="" />
```

4. In the **src** attribute of the <img> tag, type the following code.

```
@Url.Action("GetImage", "Photo",
   new { id=Model.PhotoID })
```

5. On the toolbar of the **PhotoSharingApplication - Microsoft Visual Studio** window, click the **Save Views\Photo\Display.cshtml (Ctrl+S)** button.
6. On the **BUILD** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Build Solution**.

▶ Task 3: Run the application and display a photo.

1. On the **DEBUG** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Start Debugging**.

 **Note:** The Internet Explorer window displays an error message. The error message is expected because the home page view has not been added.

2. In the Address bar of the Internet Explorer window, append the existing URL with **photo/index**, and then press Enter.
3. On the taskbar, click the **PhotoSharingApplication - Microsoft Visual Studio** icon.
4. In the Output pane of the **PhotoSharingApplication (Running) - Microsoft Visual Studio** window, click the **Output** tab.
5. On the **Output** tab, locate the following line of code.

```
>>Key: controller; Value : photo
>>Key: action; Value: index
```

 **Note:** In the above line of code, note that there are no calls to the **Display** and **GetImage** actions.

6. On the taskbar, click the **Internet Explorer** icon.
7. On the Index page, click the **Display** link corresponding to the title **Me standing on top of a mountain**.

 **Note:** The selected image is displayed on the Display page.

8. On the taskbar, click the **PhotoSharingApplication - Microsoft Visual Studio** icon.

9. In the Output pane of the **PhotoSharingApplication (Running) - Microsoft Visual Studio** window, click the **Output** tab.
10. On the **Output** tab, locate the following code.

```
>>Key: controller; Value: photo
>>Key: action; Value: Display
>>Key: id; Value: 1 Action Values: Controller: Photo; Action: GetImage
>>Key: controller; Value: Photo
>>Key: action; Value: GetImage
>>Key: id; Value: 1
```

Note: In the above code, note that there are calls to the **Display** and **GetImage** actions. Each call passes a **PhotoID**.

11. On the **DEBUG** menu of the **PhotoSharingApplication (Running) - Microsoft Visual Studio** window, click **Stop Debugging**.
12. On the **FILE** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Exit**.

Results: After completing this exercise, you will be able to create an MVC application with views that you can use to test controllers, actions, and action filters.

Module 05: Developing ASP.NET MVC 4 Views

Lab: Developing ASP.NET MVC 4 Views

Exercise 1: Adding a View for Photo Display

▶ Task 1: Add a new display view.

1. In the result pane of the Hyper-V Manager console, in the **Name** list of the **Virtual Machines** area, right-click **20486B-SEA-DEV11**, and then click **Connect**.
2. To log on to **20486B-SEA-DEV11**, click the **Ctrl+Alt+Delete** button.
3. In the **User name** box, type **Admin**, in the **Password** box, type **Pa$$w0rd**, and then click the **Forward** button.
4. On the Windows 8 Start screen, click **Desktop**.
5. On the taskbar, click the **File Explorer** icon.
6. In the **Libraries** window, navigate to **Allfiles (D):\Labfiles\Mod05\Starter\PhotoSharingApplication**, and then double-click **PhotoSharingApplication.sln**.
7. In the Solution Explorer pane of the **PhotoSharingApplication - Microsoft Visual Studio** window, expand **PhotoSharingApplication**, expand **Controllers**, and then click **PhotoController.cs**.
8. On the **BUILD** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click Build Solution.
9. In the PhotoController.cs code window, locate the following code, right-click the code, and then click **Add View**.

```
public ActionResult Display (int id)
```

10. In the **Add View** dialog box, ensure that the name in the **View name** box is **Display**.
11. In the **View engine** box, ensure that the **Razor (CSHTML)** view is selected.
12. In the **Add View** dialog box, select the **Create a strongly-typed view** check box.
13. In the **Model class** box, click **Photo (PhotoSharingApplication.Models)**.
14. In the **Scaffold template** box, ensure that the value is **Empty**.
15. In the **Add View** dialog box, clear the **Use a layout or master page** check box, and then click **Add**.

▶ Task 2: Complete the photo display view.

1. In the Display.cshtml code window, locate the following code.

```
<title>Display</title>
```

2. Replace the **TITLE** element with the following code.

```
<title> Photo: @Model.Title </title>
```

3. In the **DIV** element, type the following code.

```
<h2>"@Model.Title"</h2>
```

4. Place the mouse cursor at the end of the **</h2>** tag, press Enter twice, and then type the following code.

```
<img width="800" src="" />
```

5. In the **src** attribute of the **<img>** tag, type the following code.

```
@Url.Action("GetImage", "Photo",
   new { id = Model.PhotoID })
```

6. Place the mouse cursor at the end of the **<img>** tag, press Enter twice, and then type the following code.

```
<p>
   @Html.DisplayFor(
model =>
      model.Description)
</p>
```

7. Place the mouse cursor at the end of the **</p>** tag corresponding to the **Description** property, press Enter twice, and then type the following code.

```
<p>
   @Html.DisplayNameFor( model => model.UserName):
   @Html.DisplayFor( model => model.UserName)
</p>
```

8. Place the mouse cursor at the end of the **</p>** tag corresponding to the **UserName** property, press Enter twice, and then type the following code.

```
<p>
   @Html.DisplayNameFor(
      model => model.CreatedDate):
   @Html.DisplayFor(
      model => model.CreatedDate)
</p>
```

9. Place the mouse cursor at the end of the **</p>** tag corresponding to the **CreatedDate** property, press Enter twice, and then type the following code.

```
<p>
   @Html.ActionLink("Back to List", "Index")
</p>
```

10. On the toolbar of the **PhotoSharingApplication - Microsoft Visual Studio** window, click the **Save Views\Photo\Display.cshtml(Ctrl+S)** button.

Results: After completing this exercise, you will be able to add a single display view to the Photo Sharing web application and display the properties of a photo.

Exercise 2: Adding a View for New Photos

▶ Task 1: Add a new create view.

1. In the Solution Explorer pane of the **PhotoSharingApplication - Microsoft Visual Studio** window, under Controllers, click **PhotoController.cs**.
2. In the PhotoController.cs code window, locate the following code, right-click the code, and then click **Add View**.

```
public ActionResult Create ()
```

3. In the **Add View** dialog box, ensure that the name in the **View name** box is **Create**.
4. In the **View engine** box, ensure that the **Razor (CSHTML)** view is selected.
5. In the **Add View** dialog box, ensure that the **Create a strongly-typed view** check box is selected.
6. In the **Model class** box, ensure that the **Photo (PhotoSharingApplication.Models)** class is selected.
7. In the **Scaffold template** box, ensure that the **Empty** template is selected.
8. Ensure that the **Create as a partial view** and **Use a layout or master page** check boxes are clear, and then click **Add**.

▶ Task 2: Complete the photo create view.

1. In the Create.cshtml code window, locate the following code.

```
<title>Create</title>
```

2. Replace the **TITLE** element with the following code.

```
<title>Create New Photo</title>
```

3. In the **DIV** element of the Create.cshtml code window, type the following code, and then press Enter twice.

```
<h2>Create New Photo</h2>
```

4. In the **DIV** element, place the mouse cursor at the end of the **</h2>** tag, press Enter, and then type the following code.

```
@using (Html.BeginForm("Create",
   "Photo", FormMethod.Post,
new { enctype =
      "multipart/form-data" }))
{
}
```

5. In the **using** code block, press Enter, and then type the following code.

```
@Html.ValidationSummary(true)
```

6. Place the mouse cursor at the end of the **ValidationSummary** helper, press Enter, and then type the following code.

```
<p>
@Html.LabelFor(
      model => model.Title):
@Html.EditorFor(
      model => model.Title)   @Html.ValidationMessageFor(
      model => model.Title)
```

```
</p>
```

7. Place the mouse cursor at the end of the **</p>** tag corresponding to the **Title** property, press Enter twice, and then type the following code.

```
<p>
   @Html.LabelFor(
      model => model.PhotoFile):
   <input type="file" name="Image" />
</p>
```

8. Place the mouse cursor at the end of the **</p>** tag corresponding to the **Image** controls, press Enter twice, and then type the following code.

```
<p>
@Html.LabelFor(
      model => model.Description):
@Html.EditorFor(
      model => model.Description)
   @Html.ValidationMessageFor(
      model => model.Description)
</p>
```

9. Place the mouse cursor at the end of the **</p>** tag corresponding to the **Description** controls, press Enter twice, and then type the following code.

```
<p>
@Html.LabelFor(
      model => model.UserName):
@Html.DisplayFor(
      model => model.UserName)
</p>
```

10. Place the mouse cursor at the end of the **</p>** tag corresponding to the **UserName** controls, press Enter twice, and then type the following code.

```
<p>
@Html.LabelFor(
      model => model.CreatedDate):
@Html.DisplayFor(
      model => model.CreatedDate) </p>
```

11. Place the mouse cursor at the end of the **</p>** tag corresponding to the **CreatedDate** controls, press Enter twice, and then type the following code.

```
<p>
<input type="submit"
      value="Create" />
   @Html.ActionLink("Back to List",
      "Index")
</p>
```

12. On the toolbar of the PhotoSharingApplication - Microsoft Visual Studio window, click the Save Views\Photo\Create.cshtml(Ctrl+S) button.

Results: After completing this exercise, you will be able to create a web application with a Razor view to display new photos.

Exercise 3: Creating and Using a Partial View

▶ Task 1: Add a gallery action to the Photo Controller.

1. In the Solution Explorer pane of the **PhotoSharingApplication - Microsoft Visual Studio** window, under Controllers, click **PhotoController.cs**.
2. In the PhotoController.cs code window, locate the **Index** action code block.
3. Place the mouse cursor at the end of the **Index** action code block, press Enter twice, and then type the following code.

```
 [ChildActionOnly]
public ActionResult _PhotoGallery
   (int number = 0)
{
}
```

4. In the **_PhotoGallery** action code block, type the following code.

```
List<Photo> photos;
if (number == 0)
{
   photos = context.Photos.ToList();
}
```

5. Place the mouse cursor at the end of the **if** code block, press Enter, and then type the following code.

```
else
{
photos = (
      from p in context.Photos
orderby p.CreatedDate descending
      select p).Take(number).ToList();
}
```

6. Place the mouse cursor at the end of the **else** code block, press Enter, and then type the following code.

```
return PartialView("_PhotoGallery",
   photos);
```

7. On the toolbar of the **PhotoSharingApplication - Microsoft Visual Studio** window, click the **Save Controllers\PhotoController.cs(Ctrl+S)** button.

▶ Task 2: Add a photo gallery partial view.

1. In the Solution Explorer pane, under Controllers, click **PhotoController.cs**.
2. In the PhotoController.cs code window, locate the following code, right-click the code, and then click **Add View**.

```
public ActionResult _PhotoGallery
(int number = 0)
```

3. In the **View name** box of the **Add View** dialog box, ensure that the name is **_PhotoGallery**, and then ensure that the **Create a strongly-typed view** check box is selected.
4. In the **Model class** box, ensure that the value is **Photo (PhotoSharingApplication.Models)**, and then, in the **Scaffold template** box, ensure that the value is **Empty**.
5. In the **Add View** dialog box, select the **Create as a partial view** check box, and then click **Add**.

Note: In the _PhotoGallery.cshtml code windo the code block @model PhotosharingApplication.Models.Photo is displayed.

6. In the Solution Explorer pane of the **PhotoSharingApplication - Microsoft Visual Studio** window, right-click **Views**, point to **Add**, and then click **New Folder**.
7. In the Solution Explorer pane, name the newly created folder as **Shared**, and then press Enter.
8. In the Solution Explorer pane, drag the **_PhotoGallery.cshtml** file from the Photo folder to the Shared folder.

▶ Task 3: Complete the photo gallery partial view.

1. In the _PhotoGallery.cshtml code window, locate the following code.

```
@model
PhotoSharingApplication.Models.Photo
```

2. Replace the preceding code with the following code.

```
@model
IEnumerable
<PhotoSharingApplication.Models.Photo>
```

3. Place the mouse cursor at the end of the **@model** code block, press Enter twice, and then type the following code.

```
@foreach(var item in Model)
{
}
```

4. Place the mouse cursor in the **foreach** code block, and then type the following code.

```
<h3>"@item.Title"</h3>
```

5. Place the mouse cursor at the end of the **H3** element, press Enter twice, and then type the following code.

```
if (item.PhotoFile != null)
{
}
```

6. In the **if** code block, type the following code.

```
<img width="200"
src="@Url.Action("GetImage",
   "Photo",
   new { id = item.PhotoID })" />
```

7. Place the mouse cursor at the end of the **if** code block, press Enter twice, and then type the following code.

```
<p>
   Created By:
@Html.DisplayFor(
      model => item.UserName)
</p>
```

8. Place the mouse cursor at the end of the **</p>** tag corresponding to the **UserName** display control, press Enter twice, and then type the following code.

```
<p>
   Created On:
@Html.DisplayFor(
      model => item.CreatedDate)
</p>
```

9. Place the mouse cursor at the end of the **</p>** tag corresponding to the **CreatedDate** display control, press Enter twice, and then type the following code.

```
@Html.ActionLink("Display", "Display",
new { id = item.PhotoID })
```

10. On the toolbar of the PhotoSharingApplication - Microsoft Visual Studio window, click the Save Views\Shared_PhotoGallery.cshtmal(Ctrl+S) button.

▶ Task 4: Use the photo gallery partial view.

1. In the Solution Explorer pane, under Controllers, click **PhotoController.cs**.
2. In the PhotoController.cs code window, locate the following code.

```
return View("Index",
context.Photos.ToList());
```

3. Replace the preceding code with the following code.

```
return View("Index");
```

4. In the PhotoController.cs code window, locate the following code, right-click the code, and then click **Add View**.

```
public ActionResult Index ()
```

5. In the **View name** box of the **Add View** dialog box, ensure that the name is **Index**, and then clear the **Create a strongly-typed view** check box.
6. Ensure that the **Use a layout or master page** check box is clear, and then click **Add**.
7. In the Index.cshtml code window, locate the following code.

```
<title>Index</title>
```

8. Replace the **TITLE** element with the following code.

```
<title>All Photos</title>
```

9. In the **DIV** element, type the following code.

```
<h2>All Photos</h2>
```

10. Place the mouse cursor at the end of the **H2** element, press Enter twice, and then type the following code.

```
<p>
   @Html.ActionLink("Add a Photo",
      "Create", "Photo")
</p>
```

11. Place the mouse cursor at the end of the **</p>** tag, press Enter twice, and then type the following code.

```
@Html.Action("_PhotoGallery", "Photo")
```

12. On the toolbar of the **PhotoSharingApplication - Microsoft Visual Studio** window, click the **Save Views\Photo\Index.cshtml(Ctrl+S)** button.

Results: After completing this exercise, you will be able to create a web application with a partial view to display multiple photos.

Exercise 4: Adding a Home View and Testing the Views

▶ Task 1: Add a Controller and View for the home page.

1. In the **PhotoSharingApplication - Microsoft Visual Studio** window, in the Solution Explorer pane, right-click **Controllers**, point to **Add**, and then click **Controller**.
2. In the **Controller name** box of the **Add Controller** dialog box, type **HomeController**.
3. In the **Template** box, ensure that **Empty MVC Controller** is selected, and then click **Add**.
4. In the HomeController.cs code window, locate the following code, right-click the code, and then click **Add View**.

```
public ActionResult Index ()
```

5. In the **View name** box of the **Add View** dialog box, ensure that the name is **Index**.
6. In the **Add View** dialog box, ensure that the **Create a strongly-typed view** check box is not selected.
7. In the **Add View** dialog box, ensure that the **Use a layout or master page** check box is clear, and then click **Add**.
8. In the Index.cshtml code window, locate the following code.

```
<title>Index</title>
```

9. Replace the **TITLE** element with the following code.

```
<title>Welcome to Adventure Works
   Photo Sharing</title>
```

10. In the **DIV** element, type the following code.

```
<p>
Welcome to Adventure Works Photo
   sharing! Use this site to share
   your adventures.
</p>
```

11. Place the mouse cursor at the end of the **</p>** tag, press Enter, and then type the following code.

```
<h2>Latest Photos</h2>
```

12. In the **Index.cshtml** code window, place the mouse cursor at the end of the **</h2>** tag, press Enter, and then type the following code.

```
@Html.Action("_PhotoGallery",
   "Photo", new { number = 3 })
```

13. On the toolbar of the **PhotoSharingApplication - Microsoft Visual Studio** window, click the **Save Views\Home\Index.cshtml(Ctrl+S)** button.

▶ Task 2: Use the web application.

1. On the **DEBUG** menu of the **PhotoSharingApplication – Microsoft Visual Studio** window, click **Start Debugging**.
2. On the Welcome to Adventure Works Photo Sharing page, note that the total number of photos displayed is three.

3. On the Welcome to Adventure Works Photo Sharing page, click the **Display** link corresponding to the photo that you want to display.
4. On the display page, note that the photo is displayed with the title, description, user name, and created date, and then click the **Back to List** link.
5. On the All Photos page, note the total number of photos displayed.
6. On the All Photos page, click the **Add a Photo** link.
7. On the Create New Photo page, in the **Title** box, type **My First Photo,** and then click **Browse**.
8. In the **Choose File to Upload** dialog box, navigate to **Allfiles (D):\Labfiles\Mod05\SamplePhotos**.
9. In the SamplePhotos folder, select the photo of your choice, and then click **Open**.
10. On the Create New Photo page, in the **Description** box, type **This is the first test of the Create photo view**, and then click **Create**.
11. On the All Photos page, note that the photo titled **My First Photo** is displayed.
12. In the Internet Explorer window, click the **Close** button.
13. In the **PhotoSharingApplication (Running) - Microsoft Visual Studio** window, click the **Close** button.

Note: If you receive the "Do you want to stop debugging?" message, click **Yes**.

Results: After completing this exercise, you will be able to create a web application in which users can upload and view the photos.

Module 06: Testing and Debugging ASP.NET MVC 4 Web Applications

Lab: Testing and Debugging ASP.NET MVC 4 Web Applications

Exercise 1: Performing Unit Tests

▶ Task 1: Create a test project.

1. In the result pane of the Hyper-V Manager console, in the **Name** list of the **Virtual Machines** area, right-click **20486B-SEA-DEV11**, and then click **Connect**.
2. To log on to **20486B-SEA-DEV11**, click the **Ctrl+Alt+Delete** button.
3. In the **User name** box, type **Admin**, in the **Password** box, type **Pa$$w0rd**, and then click the **Forward** button.
4. On the Windows 8 Start screen, click **Desktop**.
5. On the taskbar, click the **File Explorer** icon.
6. In the **Libraries** window, navigate to **Allfiles (D):\Labfiles\Mod06\Starter\PhotoSharingApplication**, and then double-click **PhotoSharingApplication.sln**.
7. In the Solution Explorer pane of the **PhotoSharingApplication - Microsoft Visual Studio** window, right-click **Solution 'Photo Sharing Application' (1 project)**, click **Add**, and then click **New Project**.
8. In the navigation pane of the **Add New Project** dialog box, click **Visual C#**, click **Test**, and then, in the result pane, click **Unit Test Project**.
9. In the **Name** box, type **PhotoSharingTests**, and then click **OK**.
10. In the Solution Explorer pane of the **PhotoSharingApplication - Microsoft Visual Studio** window, under PhotoSharingTests, right-click **References**, and then click **Add Reference**.
11. In the navigation pane of the **Reference Manager - PhotoSharingTests** window, click **Solution**, and then click **Projects**.
12. In the result pane, select the check box corresponding to **PhotoSharingApplication**, and then click **OK**.
13. In the Solution Explorer pane of the **PhotoSharingApplication - Microsoft Visual Studio** window, under PhotoSharingTests, right-click **References**, and then click **Add Reference**.
14. In the navigation pane of the **Reference Manager - PhotoSharingTests** window, click **Assemblies**, and then click **Extensions**.
15. In the result pane, select the check box corresponding to **System.Web.Mvc** with version number **4.0.0.0**, and then click **OK**.

▶ Task 2: Write the tests.

1. In the Solution Explorer pane of the **PhotoSharingApplication - Microsoft Visual Studio** window, under PhotoSharingTests, right-click **UnitTest1.cs**, click **Rename**, type **PhotoControllerTests**, and then press Enter.
2. In the **Microsoft Visual Studio** dialog box, click **Yes**.
3. In the **PhotoControllerTests.cs** code window, locate the following code.

```
public void TestMethod1()
```

4. Replace the code with the following code.

```
public void Test_Index_Return_View()
```

5. Place the mouse cursor at the end of the Microsoft.VisualStudio.TestTools.UnitTesting namespace, press Enter, and then type the following code.

```
using System.Collections.Generic;
using System.Web.Mvc;
using PhotoSharingApplication.Models;
using PhotoSharingApplication.Controllers;
```

6. Place the mouse cursor in the **Test_Index_Return_View** test method code block, and then type the following code.

```
PhotoController controller =
   new PhotoController();
var result = controller.Index()
   as ViewResult;
Assert.AreEqual("Index",
   result.ViewName);
```

7. Place the mouse cursor at the end of the **Test_Index_Return_View** test method code block, press Enter twice, and then type the following code.

```
[TestMethod]
public void Test_PhotoGallery_Model_Type()
{
}
```

8. Place the mouse cursor in the **Test_PhotoGallery_Model_Type** test method code block, and then type the following code.

```
var controller = new PhotoController();
var result = controller._PhotoGallery() as PartialViewResult;
Assert.AreEqual(typeof(List<Photo>), result.Model.GetType());
```

9. Place the mouse cursor at the end of the **Test_PhotoGallery_Model_Type** test method code block, press Enter twice, and then type the following code.

```
[TestMethod]
public void Test_GetImage_Return_Type()
{
}
```

10. Place the mouse cursor in the **Test_GetImage_Return_Type** test method code block, and then type the following code.

```
var controller = new PhotoController();
var result = controller.GetImage(1) as ActionResult;
Assert.AreEqual(typeof(FileContentResult), result.GetType());
```

11. On the TEST menu of the PhotoSharingApplication - Microsoft Visual Studio window, point to Run, and then click All Tests.

12. In the Test Explorer pane, expand **Passed Tests (1)**, and then note that only the **Test_Index_Return_View** test is passed.

13. Under Failed Tests (2), note that the **Test_GetImage_Return_Type** and the **Test_PhotoGallery_Model_Type** tests are failed.
14. Under Failed Tests (2), click **Test_GetImage_Return_Type**, and then note that the test failed while trying to connect to the database.
15. Under Failed Tests (2), click **Test_PhotoGallery_Model_Type**, and then note that the test failed while trying to connect to the database.
16. In the Test Explorer pane, click the **Close** button.

▶ Task 3: Implement a repository.

1. In the Solution Explorer pane of the **PhotoSharingApplication - Microsoft Visual Studio** window, expand PhotoSharingApplication.
2. In the Solution Explorer pane of the **PhotoSharingApplication - Microsoft Visual Studio** window, right-click **Models**, click **Add**, and then click **New Item**.
3. In the navigation pane of the **Add New Item - PhotoSharingApplication** dialog box, click **Code**, and then, in the result pane, click **Interface**.
4. In the **Name** box, type **IPhotoSharingContext**, and then click **Add**.
5. In the IPhotoSharingContext.cs code window, locate the following code.

```
interface IPhotoSharingContext
```

6. Replace the code with the following code.

```
public interface IPhotoSharingContext
```

7. Place the mouse cursor in the **IPhotoSharingContext** interface code block, press Enter, and then type the following code.

```
IQueryable<Photo> Photos { get; }
```

8. Place the mouse cursor at the end of the **Photos** property code block, press Enter, and then type the following code.

```
IQueryable<Comment> Comments { get; }
```

9. Place the mouse cursor at the end of the **Comments** property code block, press Enter, and then type the following code.

```
int SaveChanges();
```

10. Place the mouse cursor at the end of the **SaveChanges** method code block, press Enter, and then type the following code.

```
T Add<T>(T entity) where T : class;
```

11. Place the mouse cursor at the end of the **Add** method code block, press Enter, and then type the following code:

```
Photo FindPhotoById (int ID) ;
```

12. Place the mouse cursor at the end of the **FindPhotoById** method code block, press Enter, and then type the following code.

```
Comment FindCommentById (int ID) ;
```

13. Place the mouse cursor at the end of the **FindCommentById** method code block, press Enter, and then type the following code.

```
T Delete<T>(T entity) where T : class;
```

14. In the Solution Explorer pane, click **PhotoSharingContext.cs**.
15. In the PhotoSharingContext.cs code window, locate the following code.

```
public class PhotoSharingContext : DbContext
```

16. Append the following code to the existing line of code.

```
, IPhotoSharingContext
```

17. Place the mouse cursor at the end of the **Comments DbSet** property code block, press Enter twice, and then type the following code.

```
IQueryable<Photo> IPhotoSharingContext.Photos
{
   get { return Photos; }
}
```

18. Place the mouse cursor at the end of the **Photos** property code block, press Enter twice, and then type the following code.

```
IQueryable<Comment> IPhotoSharingContext.Comments
{
   get { return Comments; }
}
```

19. Place the mouse cursor at the end of the **Comments** property code block, press Enter twice, and then type the following code.

```
int IPhotoSharingContext.SaveChanges ()
{
   return SaveChanges();
}
```

20. Place the mouse cursor at the end of the **SaveChanges** method code block, press Enter twice, and then type the following code.

```
T IPhotoSharingContext.Add<T>(T entity)
{
   return Set<T>().Add(entity);
}
```

21. Place the mouse cursor at the end of the **Add** method code block, press Enter twice, and then type the following code.

```
Photo IPhotoSharingContext.FindPhotoById(int ID)
{
   return Set<Photo>().Find(ID);
}
```

22. Place the mouse cursor at the end of the **FindPhotoById** code block, press Enter twice, and then type the following code.

```
Comment IPhotoSharingContext.FindCommentById(int ID)
{
   return Set<Comment>().Find(ID);
}
```

23. Place the mouse cursor at the end of the **FindCommentById** method code block, press Enter twice, and then type the following code.

```
T IPhotoSharingContext.Delete<T>(T entity)
{
   return Set<T>().Remove(entity);
}
```

24. On the **FILE** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Save All**.

25. On the **BUILD** menu, click **Build Solution**.

▶ Task 4: Refactor the photo controller to use the repository.

1. In the Solution Explorer pane of the **PhotoSharingApplication - Microsoft Visual Studio** window, expand **Controllers**, and then click **PhotoController.cs**.

2. In the PhotoController.cs code window, locate the following code.

```
private PhotoSharingContext context = new PhotoSharingContext();
```

3. Replace the code with the following code.

```
private IPhotoSharingContext context;
```

4. Place the mouse cursor at the end of the **context** object code block, press Enter twice, and then type the following code.

```
public PhotoController ()
{
   context = new PhotoSharingContext();
}
```

5. Place the mouse cursor at the end of the **PhotoController** constructor code block, press Enter twice, and then type the following code.

```
public PhotoController (IPhotoSharingContext Context)
{
   context = Context;
}
```

6. In the PhotoController.cs code window, locate the **Display** action method, and then locate the following code.

```
Photo photo = context.Photos.Find(id);
```

7. Replace the code with the following code.

```
Photo photo = context.FindPhotoById(id);
```

8. Locate the **Create** action method for the POST verb, and then locate the following code.

```
context.Photos.Add(photo);
```

9. Replace the code with the following code.

```
context.Add<Photo>(photo);
```

10. Locate the **Delete** action method, and then locate the following code.

```
Photo photo = context.Photos.Find(id);
```

11. Replace the code with the following code.

```
Photo photo = context.FindPhotoById(id);
```

12. Locate the **DeleteConfirmed** action method, and then locate the following code.

```
Photo photo = context.Photos.Find(id);
```

13. Replace the code with the following code.

```
Photo photo = context.FindPhotoById (id);
```

14. In the **DeleteConfirmed** action method, locate the following code.

```
context.Photos.Remove(photo);
```

15. Replace the code with the following code.

```
context.Delete<Photo>(photo);
```

16. Locate the **GetImage** action method, and then locate the following code.

```
Photo photo = context.Photos.Find(id);
```

17. Replace the code with the following code.

```
Photo photo = context.FindPhotoById(id);
```

18. On the **DEBUG** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Start Debugging**.
19. On the Welcome to Adventure Works Photo Sharing page, click the **Display** link corresponding to any photo of your choice.
20. On the Sample Photo 5 page, note that the changes are displayed, and then, in the Windows Internet Explorer window, click the **Close** button.

▶ Task 5: Refactor the tests to use a mock repository.

1. In the Solution Explorer pane of the **PhotoSharingApplication - Microsoft Visual Studio** window, right-click **PhotoSharingTests**, click **Add**, and then click **New Folder**.
2. In the **NewFolder1** box, type **Doubles**, and then press Enter.
3. In the Solution Explorer pane, right-click **Doubles**, click **Add**, and then click **Existing Item**.
4. In the **Add Existing Item - PhotoSharingTests** dialog box, navigate to Allfiles (**D):\Labfiles\Mod06\Fake Repository\ FakePhotoSharingContext.cs**, and then click **Add**.
5. In the Solution Explorer pane, click **PhotoControllerTests.cs**.
6. In the PhotoControllerTests.cs code window, place the mouse cursor at the end of the PhotoSharingApplication.Contollers namespace, press Enter, and then type the following code.

```
using System.Linq;
```

```
using PhotoSharingTests.Doubles;
```

7. Locate the **Test_Index_Return_View** test method, and then locate the following code.

```
PhotoController = new PhotoController();
```

8. Replace the code with the following code.

```
var context = new FakePhotoSharingContext();
var controller = new PhotoController(context);
```

9. Locate the **Test_PhotoGallery_Model_Type** test method, place the mouse cursor at the beginning of the **Test_PhotoGallery_Model_Type** test method code block, press Enter, and then type the following code.

```
var context = new FakePhotoSharingContext();
context.Photos = new [] {
   new Photo(),
   new Photo(),
   new Photo(),
   new Photo()
}.AsQueryable();
var controller = new PhotoController(context);
```

10. In the **Test_PhotoGallery_Model_Type** test method, locate the following code, select the located code, and then press Delete.

```
var controller = new PhotoController();
```

11. Locate the **Test_GetImage_Return_Type** test method, place the mouse cursor at the beginning of the **Test_GetImage_Return_Type** method, press Enter, and then type the following code.

```
var context = new FakePhotoSharingContext();
```

12. Place the mouse cursor at the end of the code you just typed, press Enter twice, and then type the following code.

```
context.Photos = new [] {
   new Photo{ PhotoID = 1, PhotoFile = new byte[1], ImageMimeType = "image/jpeg" },
   new Photo{ PhotoID = 2, PhotoFile = new byte[1], ImageMimeType = "image/jpeg" },
   new Photo{ PhotoID = 3, PhotoFile = new byte[1], ImageMimeType = "image/jpeg" },
   new Photo{ PhotoID = 4, PhotoFile = new byte[1], ImageMimeType = "image/jpeg" }
}.AsQueryable();
```

13. In the **Test_GetImage_Return_Type** method code block, locate the following code.

```
var controller = new PhotoController();
```

14. Replace the code with the following code.

```
var controller = new PhotoController(context);
```

15. On the **TEST** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Run**, and then click **All Tests**.

16. In the Test Explorer pane, note that all tests have passed.

Note: All the tests passed because the FakePhotoSharingContext test double can return test values without connecting to a database. A test double or mock object is an object used in test projects that behaves like the corresponding object in the web application project.

17. In the Test Explorer pane, click the **Close** button.

▶ Task 6: Add further tests.

1. In the Solution Explorer pane of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **PhotoControllerTests.cs**.
2. In the PhotoControllerTests.cs code window, place the mouse cursor at the end of the **PhotoControllerTests** class, press Enter twice, and then type the following code.

```
[TestMethod]
public void Test_PhotoGallery_No_Parameter ()
{
}
```

3. In the **Test_PhotoGallery_No_Parameter** test method code block, type the following code, and then press Enter twice.

```
var context = new FakePhotoSharingContext();
context.Photos = new [] {
   new Photo(),
   new Photo(),
   new Photo(),
   new Photo()
}.AsQueryable();
var controller = new PhotoController(context);
```

4. In the **Test_PhotoGallery_No_Parameter** test method code block, type the following code, and then press Enter.

```
var result = controller._PhotoGallery() as PartialViewResult;
```

5. In the **Test_PhotoGallery_No_Parameter** test method code block, type the following code.

```
var modelPhotos = (IEnumerable<Photo>)result.Model;
Assert.AreEqual(4, modelPhotos.Count());
```

6. Select all the code in the **Test_PhotoGallery_No_Parameter** method, including the **[TestMethod]** annotation.
7. On the **Edit** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Copy**.
8. Place the mouse cursor at the end of the **Test_PhotoGallery_No_Parameter** method.
9. On the **Edit** menu, click **Paste**.
10. Locate the following code in the code block you just pasted.

```
public void Test_PhotoGallery_No_Parameter()
```

11. Replace the code with the following line of code.

```
public void Test_PhotoGallery_Int_Parameter()
```

12. In the **Test_PhotoGallery_Int_Parameter** method, locate the following code.

```
var result = controller._PhotoGallery() as PartialViewResult;
```

13. In the **Test_PhotoGallery_Int_Parameter** method, replace the located code with the following code.

```
var result = controller._PhotoGallery(3) as PartialViewResult;
```

14. In the **Test_PhotoGallery_Int_Parameter**, method, locate the following code.

```
var modelPhotos = (IEnumerable<Photo>)result.Model;
Assert.AreEqual(4, modelPhotos.Count());
```

15. In the **Test_PhotoGallery_Int_Parameter**, method, replace the located code with the following code.

```
var modelPhotos = (IEnumerable<Photo>)result.Model;
Assert.AreEqual(3, modelPhotos.Count());
```

16. On the **TEST** menu of the **PhotoSharingApplication -Microsoft Visual studio** window, click **TEST**, point to **Run**, and then click **All Tests**.
17. In the Test Explorer pane, note that all tests passed.
18. In the Test Explorer pane, click the **Close** button.

Results: After completing this exercise, you will be able to add a set of PhotoController tests defined in the PhotoSharingTests project of the Photo Sharing application.

Exercise 2: Optional—Configuring Exception Handling

▶ Task 1: Edit Web.config for exception handling.

1. In the Solution Explorer pane, under PhotoSharingApplication, click **Web.config**.

Note: Ensure that you click the Web.config file in the root folder, and not the Web.config file in the Views folder.

2. In the Web.config code window, locate the following code.

```
<system.web>
```

3. Place the mouse cursor at the end of the code, press Enter, and then type the following code.

```
<customErrors mode="On" defaultRedirect="Error">
</customErrors>
```

4. In the **<customErrors>** element, type the following code.

```
<error statusCode="500" redirect="~/Error.html"/>
```

5. In the Solution Explorer pane, right-click **PhotoSharingApplication**, point to **Add**, and then click **New Item**.
6. In the navigation pane of the **Add New Item - PhotoSharingApplication** dialog box, under Visual C#, click **Web**, and then, in the result pane, click **HTML Page**.
7. In the **Name** box, type **Error.html**, and then click **Add**.
8. In the **TITLE** element of the Error.html code window, type the following code.

```
Error
```

9. In the **BODY** element, type the following code.

```
<h1>500 Error</h1>
<p>There has been an internal server error</p>
```

▶ Task 2: Create a custom error view.

1. In the Solution Explorer pane, collapse **Controllers**, collapse **Models**, and then expand **Views**.
2. In the Solution Explorer pane, under Views, right-click **Shared**, point to **Add**, and then click **View**.
3. In the **View name** box of the **Add View** dialog box, type **Error**.
4. In the **Add View** dialog box, ensure that the **Create a strongly-typed view** and the **Create as a partial view** check boxes are clear.
5. In the **Add View** dialog box, clear the **Use a layout or master page** check box, and then click **Add**.
6. In the Error.cshtml code window, locate the following code.

```
<title>Error</title>
```

7. Replace the code with the following code.

```
<title>Custom Error</title>
```

8. In the Error.cshtml code window, place the mouse cursor at the beginning of the code window, press Enter twice, and then type the following code.

```
@model System.Web.Mvc.HandleErrorInfo
```

9. In the **DIV** element, type the following code.

```
<h1>MVC Error</h1>
```

10. Place the mouse cursor at the end of the **H1** element, press Enter, and then type the following code.

```
Controller: @Model.ControllerName <br />
```

11. Place the mouse cursor at the end of the **
** tag corresponding to the **ControllerName** property, press Enter, and then type the following code.

```
Action: @Model.ActionName <br />
```

12. Place the mouse cursor at the end of the **
** tag corresponding to the **ActionName** property, press Enter, and then type the following code.

```
Message: @Model.Exception.Message
```

13. On the **FILE** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Save All**.

▶ Task 3: Configure errors in the PhotoController class.

1. In the Solution Explorer pane, collapse **Views**, expand **Controllers**, and then click **PhotoController.cs**.
2. In the PhotoController.cs code window, locate the following code.

```
[ValueReporter]
public class PhotoController : Controller
```

3. In the PhotoController.cs code window, place the mouse cursor at the beginning of the **[ValueReporter]** annotation, press Enter, and then type the following code.

```
[HandleError(View = "Error")]
```

4. In the **PhotoController** class code block, place the mouse cursor at the end of the **GetImage** action, press Enter, and then type the following code.

```
public ActionResult SlideShow()
{
}
```

5. In the **SlideShow** action code block, type the following code.

```
throw new NotImplementedException("The SlideShow action is not yet ready");
```

▶ Task 4: Raise errors.

1. On the DEBUG menu of the PhotoSharingApplication –Microsoft Visual Studio window, click Start Debugging.
2. On the Welcome to Adventure Works Photo Sharing page, click the **Display** link corresponding to the **"Sample Photo 5"** image.

3. In the Address bar of the Windows Internet Explorer window, replace the URL **/Photo/Display/5** with **/Photo/Display/malformedID**, and then click the **Go** button.
4. In the Windows Internet Explorer window, note that the custom error view displays the controller, action, and error message.
5. In the Address bar of the Windows Internet Explorer window, replace the URL **/Photo/Display/malformedID** with **/Photo/SlideShow**, and then click the **Go** button
6. The Microsoft Visual Studio application breaks on the **throw** statement.
7. In the **NotImplementedException** window, click the **Close** button.
8. In the lower-right part of the **PhotoSharingApplication (Debugging) - Microsoft Visual Studio** window, click the **IntelliTrace** tab.
9. In the IntelliTrace pane, click the **ASP.NET: GET "/photo/slideshow"** message.

 **Note:** Note the details of the HTTP call.

10. In the IntelliTrace pane, click the **Exception: Thrown: "The SlideShow action is not yet ready" (System.NotImplementedException)** message.

 **Note:** In the PhotoController.cs code window, note that the **SlideShow** action code block is highlighted.

11. On the **DEBUG** menu of the **PhotoSharingApplication (Intelligence Debugging) - Microsoft Visual Studio** window, click **Stop Debugging**.
12. In the **PhotoSharingApplication – Microsoft Visual Studio** window, click the **Close** button.

Results: After completing this exercise, you will be able to:

Configure a custom error handling strategy for an MVC application.

Module 07: Structuring ASP.NET MVC 4 Web Applications

Lab: Structuring ASP.NET MVC 4 Web Applications

Exercise 1: Using the Routing Engine

▶ Task 1: Test the routing configuration.

1. In the result pane of the Hyper-V Manager console, in the **Name** list of the **Virtual Machines** area, right-click **20486B-SEA-DEV11**, and then click **Connect**.
2. To log on to **20486B-SEA-DEV11**, click the **Ctrl+Alt+Delete** button.
3. In the **User name** box, type **Admin**, in the **Password** box, type **Pa$$w0rd**, and then click the **Submit** button.
4. On the taskbar, click the **File Explorer** icon.
5. In the **Libraries** window, navigate to **Allfiles (D):\Labfiles\Mod07\Starter\PhotoSharingApplication**, and then double-click **PhotoSharingApplication.sln**.
6. In the Solution Explorer pane of the **PhotoSharingApplication - Microsoft Visual Studio** window, expand **Photo Sharing Tests**, and then expand **Doubles**.
7. In the Solution Explorer pane, right-click **Doubles**, point to **Add**, and then click **Existing Item**.
8. In the **Add Existing Item – PhotoSharingTests** dialog box, navigate to **Allfiles (D):\Labfiles\Mod07\Fake Http Classes**, click **FakeHttpClasses.cs**, and then click **Add**.
9. In the Solution Explorer pane, under PhotoSharingTests, right-click **References**, and then click **Add Reference**.
10. In the navigation pane of the **Reference Manager – PhotoSharingTests** dialog box, under Assemblies, click **Framework**.
11. In the list of assemblies, click **System.Web**, select the check box corresponding to System.Web, and then click **OK**.
12. In the Solution Explorer pane, right-click **PhotoSharingTests**, point to **Add**, and then click **Unit Test**.
13. In the Solution Explorer pane, right-click **UnitTest1.cs**, click **Rename**, type **RoutingTests**, and then press Enter.
14. In the **Microsoft Visual Studio** dialog box, click **Yes**.
15. In the RoutingTests.cs code window, place the mouse cursor at the end of the **Microsoft.VisualStudio.TestTools.UnitTesting** namespace, press Enter, and then type the following code.

```
using System.Web.Routing;
using System.Web.Mvc;
using PhotoSharingTests.Doubles;
using PhotoSharingApplication;
```

16. In the RoutingTests.cs code window, locate the following code, and then select the code.

```
public void TestMethod1()
```

17. Replace the selected code with the following code.

```
public void Test_Default_Route_ControllerOnly()
```

18. In the **Test_Default_Route_ControllerOnly** test method code block, press Enter twice, and then type the following code.

```
var context = new FakeHttpContextForRouting(requestUrl: "~/ControllerName");
```

19. In the RoutingTests.cs code window, place the mouse cursor at the end of the **var context** code block, press Enter, and then type the following code.

```
var routes = new RouteCollection();
RouteConfig.RegisterRoutes(routes);
```

20. Place the mouse cursor at the end of the **RouteConfig.RegisterRoutes()** method code block, press Enter, and then type the following code.

```
RouteData routeData = routes.GetRouteData(context);
```

21. Place the mouse cursor at the end of the **routes.GetRouteData** method code block, press Enter, and then type the following code.

```
Assert.IsNotNull(routeData);
Assert.AreEqual("ControllerName", routeData.Values["controller"]);
Assert.AreEqual("Index", routeData.Values["action"]);
```

22. Place the mouse cursor after the **Test_Default_Route_ControllerOnly()** test method but inside the **RoutingTests** class code block, press Enter, and then type the following code.

```
 [TestMethod]
public void Test_Photo_Route_With_PhotoID()
{
}
```

23. In the **Test_Photo_Route_With_PhotoID** test method code block, type the following code.

```
var context = new FakeHttpContextForRouting(requestUrl: "~/photo/2");
```

24. Place the mouse cursor at the end of the **var context** code block that you just entered, press Enter, and then type the following code.

```
var routes = new RouteCollection();
RouteConfig.RegisterRoutes(routes);
```

25. Place the mouse cursor at the end of the **RouteConfig.RegisterRoutes()** method code block, press Enter, and then type the following code.

```
RouteData routeData = routes.GetRouteData(context);
```

26. Place the mouse cursor at the end of the **routes.GetRouteData()** method code block, press Enter, and then type the following code.

```
Assert.IsNotNull(routeData);
Assert.AreEqual("Photo", routeData.Values["controller"]);
Assert.AreEqual("Display", routeData.Values["action"]);
Assert.AreEqual("2", routeData.Values["id"]);
```

27. Place the mouse cursor after the **Test_Photo_Route_With_PhotoID** test method but inside **RoutingTests** class code block, and then type the following code.

```
 [TestMethod]
public void Test_Photo_Title_Route ()
{
}
```

28. In the **Test_Photo_Title_Route** test method code block, type the following code.

```
var context = new FakeHttpContextForRouting(requestUrl: "~/photo/title/my%20title");
```

29. Place the mouse cursor at the end of the **var context** code block that you just entered, press Enter, and then type the following code.

```
var routes = new RouteCollection();
RouteConfig.RegisterRoutes(routes);
```

30. Place the mouse cursor at the end of the **RouteConfig.RegisterRoutes()** method code block, press Enter, and then type the following code.

```
RouteData routeData = routes.GetRouteData(context);
```

31. Place the mouse cursor at the end of the **routes.GetRouteData()** method code block, press Enter., and then type the following code.

```
Assert.IsNotNull(routeData);
Assert.AreEqual("Photo", routeData.Values["controller"]);
Assert.AreEqual("DisplayByTitle", routeData.Values["action"]);
Assert.AreEqual("my%20title", routeData.Values["title"]);
```

32. On the **TEST** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Run**, and then click **All Tests**.
33. In the Test Explorer pane of the **PhotoSharingApplication - Microsoft Visual Studio** window, expand **Passed Tests**, and then note that the **Test_Default_Route_Controller_Only** route test has passed.
34. In the Test Explorer pane of the **PhotoSharingApplication - Microsoft Visual Studio** window, under **Failed Tests**, note that the **Test_Photo_Route_With_PhotoID** and the **Test_Photo Title_Route** route tests have failed**.**
35. In the Test Explorer pane of the **PhotoSharingApplication - Microsoft Visual Studio** window, click the **Close** button.

▶ Task 2: Add and test the Photo ID route.

1. In the Solution Explorer pane of the **PhotoSharingApplication - Microsoft Visual Studio** window, expand **PhotoSharingApplication**, expand **App_Start**, and then click **RouteConfig.cs**.
2. In the RouteConfig.cs code window, locate the following code.

```
routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
```

3. Place the mouse cursor at the end of the code, press Enter twice, and then type the following code.

```
routes.MapRoute(
   name: "PhotoRoute",
   url: "photo/{id}",
   defaults: new { controller = "Photo", action = "Display" },
   constraints: new { id = "[0-9]+" }
);
```

4. On the TEST menu of the PhotoSharingApplication - Microsoft Visual Studio window, click Run, and then click All Tests.
5. In the Test Explorer pane of the PhotoSharingApplication - Microsoft Visual Studio window, under Passed Tests, note that the Test_Photo_Route_With_PhotoID route test has passed.
6. In the Test Explorer pane of the **PhotoSharingApplication - Microsoft Visual Studio** window, click the **Close** button.

▶ Task 3: Add and test the Photo Title route.

1. In the RouteConfig.cs code window, place the mouse cursor after the **PhotoRoute** route but before the **Default** route code block, and then type the following code.

```
routes.MapRoute(
   name: "PhotoTitleRoute",
   url: "photo/title/{title}",
   defaults: new { controller = "Photo", action = "DisplayByTitle" }
);
```

2. In the Solution Explorer pane of the **PhotoSharingApplication - Microsoft Visual Studio** window, under PhotoSharingApplication, expand **Controllers**, and then click **PhotoController.cs**.
3. In the PhotoController.cs code window, place the mouse cursor after the **Display** action code block, press Enter twice, and then type the following code.

```
public ActionResult DisplayByTitle(string title)
{
}
```

4. In the **DisplayByTitle** action method code block, type the following code.

```
Photo photo = context.FindPhotoByTitle(title);
if (photo == null)
{
   return HttpNotFound();
}
return View("Display", photo);
```

5. On the **TEST** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Run**, and then click **All Tests**.
6. In the Test Explorer pane of the **PhotoSharingApplication - Microsoft Visual Studio** window, under **Passed Tests**, note that the **Test_Photo_Title_Route** route test has passed.
7. In the Test Explorer pane of the **PhotoSharingApplication - Microsoft Visual Studio** window, click the **Close** button.

▶ Task 4: Try out the new routes.

1. On the **DEBUG** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Start Debugging**.
2. On the Welcome to Adventure Works Photo Sharing! Use this site to share your adventures page, right-click the **Display** link corresponding to any image, and then click **Properties**.
3. In the **Properties** dialog box, note that the **Address** property does not include the name of the **Display** action method, and then click **Cancel**.
4. On the Welcome to Adventure Works Photo Sharing! Use this site to share your adventures page, click the **Display** link corresponding to any image.

5. In the Address bar of the Windows Internet Explorer window, note that the URL does not include the name of the **Display** action method.
6. In the Address bar of the Windows Internet Explorer window, select **/photo/<ID>** in the URL, and then press Delete.
7. In the Address bar, append the existing URL with **/photo/title/sample photo 3**, and then click the **Go to** button.

Note: Note that the Sample Photo 3 is displayed.

8. In the Windows Internet Explorer window, click the **Close** button.

Results: After completing this exercise, you will be able to create a Photo Sharing application with three configured routes that enable visitors to easily locate photos by using logical URLs.

Exercise 2: Optional—Building Navigation Controls

▶ Task 1: Install the MVC site map provider.

1. On the **PROJECT** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Manage NuGet Packages**.
2. In the navigation pane of the **PhotoSharingApplication - Manage NuGet Packages** dialog box, click **Online**.
3. In the **Search Online** box of the **PhotoSharingApplication - Manage NuGet Packages** dialog box, type **MvcSiteMapProvider**, and then click the **Search** button.
4. In the **PhotoSharingApplication - Manage NuGet Packages** dialog box, click **Install** corresponding to the **MVCSiteMapProvider** package.
5. In the **PhotoSharingApplication - Manage NuGet Packages** dialog box, click **Close**.

▶ Task 2: Configure the MVC site map provider.

1. In the Solution Explorer pane of the **PhotoSharingApplication - Microsoft Visual Studio** window, under Global.asax, click **Web.config**.

Note: Ensure that you do not open the Web.config file in the Views folder.

2. In the **<siteMap>** element of the Web.config code window, locate the **<add>** element.
3. In the **enableLocalization** attribute, of the **<add name="MvcSiteMapProvider">** element, change the value of the attribute from **true** to **false**.
4. On the **FILE** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Save Web.config**.
5. In the Solution Explorer pane of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Mvc.sitemap**.
6. In the Mvc.sitemap code window, select the following code, and then press Delete.

```
<mvcSiteMapNode title="About" controller="Home" action="About"/>
```

7. In the **<mvcSiteMapNode>** element, type the following code.

```
<mvcSiteMapNode title="All Photos" controller="Photo" action="Index" key="AllPhotos"
/>
```

8. On the **FILE** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Save Mvc.sitemap**.
9. On the **BUILD** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Build Solution**.

▶ Task 3: Render menus and breadcrumb trails.

1. In the Solution Explorer pane, expand **Views**, expand **Home**, and then click **Index.csthml**.
2. In the Index.cshtml code window, place the mouse cursor after the ** ** tag, press Enter, type the following code, and then press Enter.

```
Menu: @Html.MvcSiteMap().Menu(false, false, true)
```

3. In the Index.cshtml code window, place the mouse cursor at the end of the MvcSiteMap code block, press Enter, and then type the following code.

```
Breadcrumb Trail: @Html.MvcSiteMap().SiteMapPath()
```

4. In the Solution Explorer pane, under Views, expand **Photo**, and then click **Index.csthml**.
5. In the Index.cshtml code window, place the mouse cursor after the ** ** tag, press Enter, type the following code, and then press Enter.

```
Menu: @Html.MvcSiteMap().Menu(false, false, true)
```

6. In the Index.cshtml code window, place the mouse cursor at the end of the MvcSiteMap code block, press Enter, and then type the following code.

```
Breadcrumb Trail: @Html.MvcSiteMap().SiteMapPath()
```

▶ Task 4: Try out the menus.

1. On the **DEBUG** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Start Debugging**.
2. On the Welcome to Adventures Works Photo Sharing page, under Menu, click the **All Photos** link, and then note that the All Photos page is displayed.
3. In the Breadcrumb Trail section of the All Photos page, click the **Home** link.
4. In the Windows Internet Explorer window, click the **Close** button.
5. In the PhotoSharingApplication - Microsoft Visual Studio window, click the Close button.

Results: After completing this exercise, you will be able to create a Photo Sharing application with a simple site map, menu, and breadcrumb control.

Module 08: Applying Styles to ASP.NET MVC 4 Web Applications

Lab: Applying Styles to MVC 4 Web Applications

Exercise 1: Creating and Applying Layouts

▶ Task 1: Open and browse through the Photo Sharing application.

1. In the result pane of the Hyper-V Manager console, in the **Name** list of the **Virtual Machines** area, right-click **20486B-SEA-DEV11**, and then click **Connect**.
2. To log on to **20486B-SEA-DEV11**, click the **Ctrl+Alt+Delete** button.
3. In the **User name** box, type **Admin**, in the **Password** box, type **Pa$$w0rd**, and then click the **Forward** button.
4. On the taskbar, click the **File Explorer** icon.
5. In the **Libraries** window, navigate to **Allfiles (D):\Labfiles\Mod08\Starter\PhotoSharingApplication**, and then double-click **PhotoSharingApplication.sln**.
6. On the **DEBUG** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Start Debugging**.

Note: The Welcome to Adventure Works Photo Sharing! Use this site to share your adventures page is displayed. The menu and the breadcrumb trail for the site are displayed on this page.

7. On the Welcome to Adventure Works Photo Sharing! Use this site to share your adventures page, under **Home**, click the **All Photos** link.

Note: The All Photos page is displayed. The menu and the breadcrumb trail for the site are not displayed on this page.

8. Under **Sample Photo 1** of the All Photos page, click the **Display** link.

Note: The menu and the breadcrumb trail are not displayed on the Sample Photo 1 page.

9. In the Windows Internet Explorer window, click the **Close** button.

▶ Task 2: Create a new layout.

1. In the Solution Explorer pane of the **PhotoSharingApplication - Microsoft Visual Studio** window, expand PhotoSharingApplication, and then expand **Views**.
2. Under Views, right-click **Shared**, point to **Add**, and then click **View**.
3. In the **View name** box of the **Add View** dialog box, type **_MainLayout**.
4. Ensure that the **Create a strongly-typed view** check box is cleared.

5. Ensure that the **Create as a partial view** check box is cleared.
6. Ensure that the **Use a layout or master page** check box is cleared, and then click **Add**.
7. In the _MainLayout.cshtml code window, locate the following code.

```
<title>_MainLayout</title>
```

8. Replace the code with the following code.

```
<title>@ViewBag.Title</title>
```

9. In the **DIV** element, type the following code.

```
<h1 class="site-page-title">Adventure Works Photo Sharing
</h1>
```

10. Place the mouse cursor at the end of the **H1** element you just created, press Enter, and then type the following code.

```
<div class="clear-floats" />
```

Note: The clear-floats class you just added to the page will be used with the style sheet for the web application.

11. Place the mouse cursor at the end of the code you just typed, press Enter, type the following code, and then press Enter.

```
<div id="topmenu">
   @Html.MvcSiteMap().Menu(false, true, true)
</div>
```

12. Place the mouse cursor at the end of the code you just typed, press Enter, type the following code, and then press Enter.

```
<div id="breadcrumb">
   @Html.MvcSiteMap().SiteMapPath()
</div>
```

13. Place the mouse cursor at the end of the code you just typed, press Enter, and then type the following code.

```
<div>
   @RenderBody()
</div>
```

14. On the **FILE** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Save Views\Shared_MainLayout.cshtml**.

▶ Task 3: Set the default layout for the application.

1. In the Solution Explorer pane of the **PhotoSharingApplication - Microsoft Visual Studio** window, right-click **Views**, point to **Add**, and then click **View**.
2. In the **View name** box of the **Add View** dialog box, type **_ViewStart**.
3. Ensure that the **Create a strongly-typed view** check box is cleared.
4. Ensure that the **Create as a partial view** check box is cleared.

5. Ensure that the **Use a layout or master page** check box is cleared, and then click **Add**.
6. In the _ViewStart.cshtml code window, locate the following code.

```
Layout = null;
```

7. Replace the code with the following code.

```
Layout = "~/Views/Shared/_MainLayout.cshtml";
```

8. In the _ViewStart.cshtml code window, locate the following code, select the code and all the subsequent lines of code, and then press Delete.

```
<!DOCTYPE html>
```

9. On the **FILE** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Save Views/_ViewStart.cshtml**.

▶ Task 4: Update the views to use the layout.

1. In the Solution Explorer pane of the **PhotoSharingApplication - Microsoft Visual Studio** window, under Views, expand **Home**, and then click **Index.cshtml**.
2. In the Index.cshtml code window, locate the following code.

```
Layout = null;
```

3. Replace the code with the following code.

```
ViewBag.Title = "Welcome to Adventure Works Photo Sharing";
```

4. In the Index.cshtml code window, locate the following code, select the code, and then press Delete.

```
<!DOCTYPE html>
<html>
<head>
   <meta name="viewport" content="width=device-width" />
   <title>Welcome to Adventure Works Photo Sharing</title>
</head>
<body>
   <div>
```

5. In the Index.cshtml code window, locate the following code, select the code, and then press Delete.

```
Menu: @Html.MvcSiteMap().Menu(false, false, true)
Current Location: @Html.MvcSiteMap().SiteMapPath()
```

6. In the Index.cshtml code window, locate the following code, select the code, and then press Delete.

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

7. On the **FILE** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Save Views\Home\Index.cshtml**.
8. In the Solution Explorer pane of the **PhotoSharingApplication - Microsoft Visual Studio** window, under Views, expand **Photo**, and then click **Display.cshtml**.
9. In the Display.cshtml code window, locate the following code.

```
Layout = null;
```

10. Replace the code with the following code.

```
ViewBag.Title = Model.Title;
```

11. Locate the following code, select the code, and then press Delete.

```
<!DOCTYPE html>
<html>
<head>
   <meta name="viewport" content="width=device-width" />
   <title>@Model.Title</title>
</head>
<body>
   <div>
```

12. Locate the following code, select the code, and then press Delete.

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

13. On the **FILE** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Save Views\Photo\Display.cshtml**.
14. In the Solution Explorer pane of the **PhotoSharingApplication - Microsoft Visual Studio** window, in the **Views/Shared** folder, double-click **Error.cshtml**.
15. In the Error.cshtml code window, locate the following code.

```
Layout = null;
```

16. Replace the code with the following code.

```
ViewBag.Title = "Custom Error";
```

17. In the Error.cshtml code window, locate the following code, select the code, and then press Delete.

```
<!DOCTYPE html>
<html>
<head>
   <meta name="viewport" content="width=device-width" />
   <title>Custom Error</title>
</head>
<body>
   <div>
```

18. In the Error.cshtml code window, locate the following code, select the code, and then press Delete.

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

19. On the **FILE** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Save Views\Shared\Error.cshtml**.

▶ Task 5: Browse through the web application.

1. On the **DEBUG** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Start Debugging**.

Note: On the Adventure Works Photo Sharing page, note that the menu and the breadcrumb trail are displayed.

2. On the **Menu** of the Adventure Works Photo Sharing page, click the **All Photos** link.

Note: On the All Photos page, note that the menu and the breadcrumb trail are now displayed.

3. Under **Sample Photo 1** of the All Photos page, click the **Display** link.

Note: On the Adventure Works Photo Sharing page, note that the Sample Photo 1 is displayed. The site title, menu, and breadcrumb trail are now displayed on this webpage.

4. In the Windows Internet Explorer window, click the **Close** button.

Results: After completing this exercise, you will be able to create an ASP.NET MVC 4 web application that uses a single layout to display every page of the application.

Exercise 2: Applying Styles to an MVC Web Application

▶ Task 1: Examine the HTML mockup web application.

1. On the taskbar, click the **File Explorer** icon.
2. In the **PhotoSharingApplication** dialog box, navigate to **Allfiles (D):\Labfiles\Mod08 \Expression Web Mock Up**, and then double-click **default.html**.

Note: On the Welcome to Adventure Works Photo Sharing page, note the colors, fonts, and layout of the page.

3. On the Welcome to Adventure Works Photo Sharing page, click the **All Photos** link.

Note: On the All Photos page, note that the layout of the page is similar to the home page, but more photos are displayed on the page.

4. On the All Photos page, click the **Details** link corresponding to any image.

Note: The photo, metadata, description, and comments are displayed on the Details page.

5. In the Windows Internet Explorer window, click the **Close** button.

▶ Task 2: Import the styles and graphics.

1. Switch to the PhotoSharingApplication - Microsoft Visual Studio window.
2. In the Solution Explorer pane of the **PhotoSharingApplication - Microsoft Visual Studio** window, right-click **PhotoSharingApplication**, point to **Add**, and then click **New Folder**.
3. In the Solution Explorer pane, rename **NewFolder1** as **Content**, and then press Enter.
4. In the Solution Explorer pane, right-click **Content**, point to **Add**, and then click **Existing Item**.
5. In the **Add Existing Item - PhotoSharingApplication** dialog box, navigate to **Allfiles (D):\Labfiles\Mod08\Expression Web Mock Up\Content**, click **PhotoSharingStyles.css**, and then click **Add**.
6. In the Solution Explorer pane, right-click **Content**, point to **Add**, and then click **Existing Item**.
7. In the **Add Existing Item - PhotoSharingApplication** dialog box, click **BackgroundGradient.jpg**, and then click **Add**.
8. In the Solution Explorer pane, under Shared, click **_MainLayout.cshtml**.
9. In the _MainLayout.cshtml code window, locate the following code.

```
<title>@ViewBag.Title</title>
```

10. Place the mouse cursor at the end of the **TITLE** element, press Enter, and then type the following code.

```
<link type="text/css" rel="stylesheet" href="~/content/PhotoSharingStyles.css" />
```

11. On the **FILE** menu of the **PhotoSharingApplication – Microsoft Visual Studio** window, click **Save Views\Shared_MainLayout.cshtml**.

▶ Task 3: Update the element classes to use the styles.

1. In the Solution Explorer pane, under DisplayTemplates, click **_PhotoGallery.cshtml**.
2. In the _PhotoGallery.cshtml window, locate the following code.

```
@foreach (var item in Model)
{
<div>
```

3. In the code, replace the ** ** tag with the following code.

```
<div class="photo-index-card">
```

4. In the **DIV** element of the _PhotoGallery.cshtml window, locate the following code.

```
<img width="200"
```

5. Replace the code with the following code.

```
<img class="photo-index-card-img"
```

6. In the _PhotoGallery.cshtml window, locate the ** ** tag after the **<img>** tag.
7. Replace ** ** tag with the following code.

```
<div class="photo-metadata">
```

8. In the _PhotoGallery.cshtml window, locate the following code.

```
<span>
   Created By:
</span>
```

9. Replace the code with the following code.

```
<span class="display-label">
   Created By:
</span>
```

10. In the _PhotoGallery.cshtml window, locate the following code.

```
<span>
   @Html.DisplayFor(model => item.UserName)
</span>
```

11. Replace the code with the following code.

```
<span class="display-field">
   @Html.DisplayFor(model => item.UserName)
</span>
```

12. In the _PhotoGallery.cshtml window, locate the following code.

```
<span>
   Created On:
</span>
```

13. Replace the code with the following code.

```
<span class="display-label">
   Created On:
```

```
</span>
```

14. In the _PhotoGallery.cshtml window, locate the following code.

```
<span>
   @Html.DisplayFor(model => item.CreatedDate)
</span>
```

15. Replace the code with the following code.

```
<span class="display-field">
   @Html.DisplayFor(model => item.CreatedDate)
</span>
```

▶ Task 4: Browse the styled web application.

1. On the **DEBUG** menu of the **PhotoSharingApplication – Microsoft Visual Studio** window, click **Start Debugging**.

Note: On the Welcome to Adventure Works page, examine the styles applied to the home page. Note the colors, fonts, and background gradient image. Also note the layout of the photo thumbnail cards, and the top menu and breadcrumb trail shortcuts.

2. On the Welcome to Adventure Works page, click the **All Photos** link.

Note: Note the new style applied to the photo gallery on the All Photos page.

3. On the All Photos page, click the **Display** link corresponding to a photo.

Note: Note the new style applied to the display view.

4. In the Windows Internet Explorer window, click the **Close** button.

Results: After completing this exercise, you will be able to create a Photo Sharing application with a consistent look and feel.

Exercise 3: Optional—Adapting Webpages for Mobile Browsers

▶ Task 1: Test the application as a mobile device.

1. On the DEBUG menu of the PhotoSharingApplication - Microsoft Visual Studio window, click Start Debugging.
2. On the **Tools** menu of the Windows Internet Explorer window, click **F12 developer tools**.

Note: In the Windows Internet Explorer window, note that a developer window is displayed.

3. On the **Tools** menu of the developer window, point to **Resize**, and then click **Custom**.
4. In the **Width** box of the **Resize Browser** dialog box, type **480**.
5. In the **Height** box, type **700**, and then click **Add**.
6. In the dimensions area of the **Resize Browser** dialog box, click **480x700**, and then click **Resize**.

Note: The size of the Windows Internet Explorer window has reduced according to the specified dimensions.

7. In the **Resize Browser** dialog box, click **Close**.
8. On the **Tools** menu of the developer window, point to **Change user agent string**, and then click **IE9 for Windows Phone 7**.
9. On the **File** menu of the developer window, click **Exit**.
10. In the Address bar of the Windows Internet Explorer window, click the **Refresh (F5)** button.

Note: The main heading, menu, and other elements of the home page do not display correctly in the mobile view.

11. In the Windows Internet Explorer window, click the **Close** button.

▶ Task 2: Add a new mobile layout.

1. In the Solution Explorer pane of the **PhotoSharingApplication - Microsoft Visual Studio** window, under DisplayTemplates, right-click **_MainLayout.cshtml**, and then click **Copy**.
2. In the Solution Explorer pane, right-click **Shared**, and then click **Paste**.
3. Under DisplayTemplates, right-click **Copy of _MainLayout.cshtml**, click **Rename**, type **_MainLayout.Mobile**, and then press Enter.
4. In the **_MainLayout.Mobile.cshtml** code window, locate the following code.

```
<h1 class="site-page-title">Adventure Works Photo Sharing</h1>
```

5. Replace the code with the following code.

```
<h1 class="site-page-title">
   Adventure Works <br />
   Photo Sharing
</h1>
```

6. Place the mouse cursor at the end of the **H1** element, press Enter, and then type the following code.

```
<h2>Mobile Site</h2>
```

7. On the **FILE** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Save Views\Shared_MainLayout.Mobile.cshtml**.

▶ Task 3: Add a media query to the style sheet.

1. In the Solution Explorer pane of the **PhotoSharingApplication - Microsoft Visual Studio** window, under Content, click **PhotoSharingStyles.css**.
2. In the PhotoSharingStyles.css code window, place the mouse cursor at the end of the style sheet, press Enter, and then type the following code.

```
@media only screen and (max-width: 500px) {
}
```

3. In the PhotoSharingStyles.css code window, locate the following code.

```
.topmenulink {
   float: left;
   background-color: #cccccc;
   width: 200px;
   height: 35px;
   text-align: center;
   line-height: 2em;
   border-radius: 5px;
   margin: 5px;
}
```

4. In the PhotoSharingStyles.css code window, place the mouse cursor in the media query you just added, and then type the following code.

```
.topmenulink {
   float: left;
   background-color: #cccccc;
   width: 200px;
   height: 35px;
   text-align: center;
   line-height: 2em;
   border-radius: 5px;
   margin: 5px;
}
```

5. In the media query you just pasted, locate the following code.

```
width: 200px;
```

6. In the located code, replace **200px** with **100px**.

▶ Task 4: Retest the application as a mobile device.

1. On the **DEBUG** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Start Debugging**.
2. On the **Tools** menu of the Windows Internet Explorer window, click **F12 developer tools**.
3. On the **Cache** menu of the developer window, click **Clear browser cache**.
4. In the **Clear Browse Cache** dialog box, click **Yes**.

5. On the **Tools** menu of the developer window, point to **Change user agent string**, and then click **IE9 for Windows Phone 7**.
6. On the Tools menu of the developer window, point to **Resize**, and then click **Custom**.
7. In the dimension area of the **Resize Browser** dialog box, click **480x700**, and then click **Resize**.

Note: The size of the Windows Internet Explorer window has reduced according to the specified dimensions.

8. In the **Resize Browser** dialog box, click the **Close** button.
9. On the **File** menu of the developer tools window, click **Exit**.
10. In the Address bar of the Windows Internet Explorer window, click the **Refresh (F5)** button, and then note that the home page of the web application is displayed without problems in the mobile view.

Note: This indicates that the mobile view and media query have been successfully applied.

11. In the Windows Internet Explorer window, click the **Close** button.
12. In the **PhotoSharingApplication - Microsoft Visual Studio** window, click the **Close** button.

Results: After completing this exercise, you will be able to create a Photo Sharing application that displays well on mobile devices and devices with small screens.

Module 09: Building Responsive Pages in ASP.NET MVC 4 Web Applications

Lab: Building Responsive Pages in ASP.NET MVC 4 Web Applications

Exercise 1: Using Partial Page Updates

▶ Task 1: Import the Comment controller and Delete view.

1. In the result pane of the Hyper-V Manager console, in the **Name** list of the **Virtual Machines** area, right-click **20486B-SEA-DEV11**, and then click **Connect**.
2. To log on to **20486B-SEA-DEV11**, click the **Ctrl+Alt+Delete** button.
3. In the **User name** box, type **Admin**, in the **Password** box, type **Pa$$w0rd**, and then click the **Forward** button.
4. On the taskbar, click the **File Explorer** icon.
5. In the Libraries window, navigate to Allfiles (D):\Labfiles\Mod09\Starter \PhotoSharingApplication, and then double-click PhotoSharingApplication.sln
6. In the Solution Explorer pane of the **PhotoSharingApplication - Microsoft Visual Studio** window, expand **PhotoSharingApplication**, right-click **Views**, point to **Add**, and then click **New Folder**.
7. In the Solution Explorer pane, rename **New Folder1** as **Comment**, and then press Enter.
8. In the Solution Explorer pane, under Views, right-click **Comment**, point to **Add**, and then click **Existing Item**.
9. In the **Add Existing Item - PhotoSharingApplication** dialog box, navigate to **Allfiles (D):\Labfiles\Mod09\Comment Components**, click **Delete.cshtml**, and then click **Add**.
10. In the Solution Explorer pane, right-click **Controllers**, point to **Add**, and then click **Existing Item**.
11. In the **Add Existing Item - PhotoSharingApplication** dialog box, navigate to **Allfiles (D):\Labfiles\Mod09\Comment Components**, click **CommentController.cs**, and then click **Add**.

▶ Task 2: Add the _CommentsForPhoto action and view.

1. In the Solution Explorer pane of the PhotoSharingApplication - Microsoft Visual Studio window, click CommentController.cs.
2. In the CommentController.cs code window, locate the following code.

```
public CommentController (IPhotoSharingContext Context)
{
   context = Context;
}
```

3. Place the mouse cursor after the located code, press Enter twice, and then type the following code.

```
[ChildActionOnly]
public PartialViewResult _CommentsForPhoto(int PhotoId)
{
}
```

4. In the **_CommentsForPhoto** action code block, type the following code.

```
var comments = from c in context.Comments
```

```
   where c.PhotoID == PhotoId
   select c;
```

5. Place the mouse cursor at the end of the code block you just created, press Enter, and then type the following code.

```
ViewBag.PhotoId = PhotoId;
```

6. Place the mouse cursor at the end of the code block you just created, press Enter, and then type the following code.

```
return PartialView("_CommentsForPhoto", comments.ToList());
```

7. On the **BUILD** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Build Solution**.
8. In the CommentController.cs code window, right-click the following code, and then click **Add View**.

```
public PartialViewResult _CommentsForPhoto(int PhotoId)
```

9. In the **Add View** dialog box, ensure that the name in the **View name** box is **_CommentsForPhoto**.
10. Select the **Create a strongly-typed view** check box.
11. In the **Model Class** box, click **Comment**. **(PhotoSharingApplication.Models)**.
12. Select the **Create as a partial view** check box, and then click **Add**.
13. In the Solution Explorer pane, under **Views**, drag the **_CommentsForPhoto.cshtml** file from the **Comment** folder to the **Shared** folder.
14. In the **_CommentsForPhoto.cshtml** code window, locate the following code.

```
@model PhotoSharingApplication.Models.Comment
```

15. Replace the code with the following code.

```
@model IEnumerable<PhotoSharingApplication.Models.Comment>
```

16. Place the mouse cursor at the end of the code you just entered, press Enter twice, and then type the following code.

```
<h3>Comments</h3>
```

17. Place the mouse cursor at the end of the **H3** element you just created, press Enter, and then type the following code.

```
<div id="comments-tool">
   <div id="all-comments">
   </div>
</div>
```

18. In the ** ** element, type the following code.

```
@foreach (var item in Model)
{
   <div class="photo-comment">
   </div>
 }
```

19. In the ** ** element, type the following code.

```
<div class="photo-comment-from">
   From:
   @Html.DisplayFor(modelItem => item.UserName)
</div>
```

20. Place the mouse cursor at the end of the **DIV** element you just created, press Enter, and then type the following code.

```
<div class="photo-comment-subject">
   Subject:
   @Html.DisplayFor(modelItem => item.Subject)
</div>
```

21. Place the mouse cursor at the end of the **DIV** element you just created, press Enter, and then type the following code.

```
<div class="photo-comment-body">
   @Html.DisplayFor(modelItem => item.Body)
</div>
```

22. Place the mouse cursor at the end of the **DIV** element you just created, press Enter, and then type the following code.

```
@Html.ActionLink("Delete This Comment", "Delete", new { id = item.CommentID})
```

23. In the Solution Explorer pane, under Views, expand **Photo**, and then click **Display.cshtml**.
24. In the Display.cshtml code window, locate the following code.

```
<p>
   @Html.ActionLink("Back To List", "Index")
   @Html.ActionLink("Delete This Photo", "Delete", new { id = Model.PhotoID })
</p>
```

25. Immediately before the located code, press Enter twice, and then type the following code.

```
@Html.Action("_CommentsForPhoto", "Comment", new { PhotoId = Model.PhotoID })
```

26. On the **DEBUG** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Start Debugging**.
27. On the Welcome to Adventure Works Photo Sharing page, click the **All Photos** link.
28. Under Sample Photo 1, click the **Display** link.
29. On the Welcome to Adventure Works Photo Sharing page, scroll to the end of the page, and then note that the two comments are displayed.
30. In the Windows Internet Explorer window, click the **Close** button.

▶ Task 3: Add the _Create Action and the _CreateAComment views.

1. In the Solution Explorer pane, under Controllers, click **CommentController.cs**.
2. In the CommentController.cs code window, place the mouse cursor within the **CommentController** class but outside any method code block, and type the following code.

```
public PartialViewResult _Create(int PhotoId)
{
}
```

3. In the **_Create** action code block, type the following code.

```
Comment newComment = new Comment();
newComment.PhotoID = PhotoId;
```

4. In the **_Create** action code block, place the mouse cursor at the end of the **Comment** object you just created, press Enter, and then type the following code.

```
ViewBag.PhotoID = PhotoId;
```

5. Place the mouse cursor at the end of the **ViewBag** attribute you just created, press Enter, and then type the following code:

```
return PartialView("_CreateAComment");
```

6. In the Solution Explorer pane, under Views, right-click **Shared**, point to **Add**, and then click **View**.
7. In the **View name** box of the **Add View** dialog box, type **_CreateAComment**.
8. Ensure that the **Create a strongly-typed view** check box is selected.
9. In the **Model Class** box, ensure that the value is **Comment (PhotoSharingApplication.Models)**.
10. Select the **Create as a partial view** check box, and then click **Add**.
11. In the _CreateAComment.cshtml code window, place the mouse cursor at the end of the **@model** statement, press Enter twice, and then type the following code.

```
@Html.ValidationSummary(true)
```

12. Place the mouse cursor at the end of the validation messages you just created, press Enter, and then type the following code.

```
<div class="add-comment-tool">
</div>
```

13. In the ** ** element you just created, type the following code.

```
<div>
</div>
```

14. In the **DIV** element you just created, type the following code.

```
<span class="editor-label">
   Subject:
</span>
```

15. Place the mouse cursor at the end of the **SPAN** element you just created, press **Enter**, and then type the following code.

```
<span class="editor-field">
   @Html.EditorFor(model => model.Subject)
</span>
```

16. In the ** ** element, place the mouse cursor after the end tag of the **DIV** element, press Enter, and then type the following code.

```
<div>
</div>
```

17. In the **DIV** element you just created, type the following code.

```
<span class="editor-label">
```

```
   Body:
</span>
```

18. Place the mouse cursor at the end of the **SPAN** element you just created, press Enter, and then type the following code.

```
<span class="editor-field">
   @Html.EditorFor(model => model.Body)
</span>
```

19. In the ** ** element, place the mouse cursor at the end of the second **DIV** element, press Enter, and then type the following code.

```
<input type="submit" value="Create" />
```

20. On the **FILE** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Save All**.

▶ Task 4: Add the _CommentsForPhoto POST action.

1. In the Solution Explorer pane, under Controllers, click **CommentController.cs**.
2. In the CommentController.cs code window, place the cursor within the **CommentController** class but outside any method code block, and then type the following code.

```
[HttpPost]
public PartialViewResult _CommentsForPhoto(Comment comment, int PhotoId)
{
}
```

3. In the **_CommentsforPhoto** action code block you just created, type the following code.

```
context.Add<Comment>(comment);
context.SaveChanges();
```

4. In the **_CommentsforPhoto** action code block, place the mouse cursor at the end of the **comment** object you just created, press Enter, and then type the following code.

```
var comments = from c in context.Comments
   where c.PhotoID == PhotoId
   select c;
```

5. In the **_CommentsforPhoto** action code block, place the mouse cursor at the end of the query you just created, press Enter, and then type the following code.

```
ViewBag.PhotoId = PhotoId;
```

6. In the **_CommentsforPhoto** action code block, place the mouse cursor at the end of the **ViewBag** attribute you just created, press Enter, and then type the following code.

```
return PartialView("_CommentsForPhoto", comments.ToList());
```

7. On the **FILE** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Save All**.

▶ Task 5: Complete the _CommentsForPhoto view.

1. In the Solution Explorer pane, under **Shared**, click **_CommentsForPhoto.cshtml**.
2. In the _CommentsForPhoto.cshtml code window, locate the following code.

```
<h3>Comments</h3>
```

3. Place the mouse cursor at the end of the located code, press Enter twice, and then type the following code.

```
@using (Ajax.BeginForm("_CommentsForPhoto",
   new { PhotoId = ViewBag.PhotoId },
   new AjaxOptions { UpdateTargetId = "comments-tool" }))
{
```

4. In the _CommentsForPhoto.cshtml code window, locate the last ** ** tag, press Enter, and then add the following code.

```
}
```

5. In the _CommentsForPhoto.cshtml code window, locate the last ** ** tag.
6. Before the located tag, type the following code.

```
<div id="add-comment" class="add-comment-box">
</div>
```

7. In the **DIV** element you just created, type the following code.

```
@Html.Action("_Create", "Comment", new { PhotoId = ViewBag.PhotoId })
```

8. In the Solution Explorer pane, under Shared, click **_MainLayout.cshtml**.
9. In the _MainLayout.cshtml code window, locate the following code.

```
</head>
```

10. Just before the located code, insert the following code.

```
<script src="http://ajax.aspnetcdn.com/ajax/jquery/jquery-1.8.0.min.js"
type="text/javascript"></script>
<script src="http://ajax.aspnetcdn.com/ajax/mvc/3.0/jquery.unobtrusive-ajax.js"
type="text/javascript"></script>
```

11. On the **DEBUG** menu of the **PhotoSharingApplication – Microsoft Visual Studio** window, click **Start Debugging**.
12. On the Welcome to Adventure Works Photo Sharing page, click the **All Photos** link.
13. Under Sample Photo 1, click the **Display** link, and then, on the Welcome to Adventure Works Photo Sharing page, note that the two comments are displayed.
14. In the **Subject** box, type **Test Comment**.
15. In the **Body** box, type **This comment is to test AJAX-based partial page updates.**, and then click **Create**.

Note: Note that the new comment is displayed without a page reload.

16. In the Windows Internet Explorer window, click the **Close** button.

Results: At the end of this exercise, you will have ensured that new comments can be added and displayed on the pages of the application without a complete page reload. You will create a Photo Sharing application with a comments tool, implemented by using partial page updates.

Exercise 2: Optional—Configuring the ASP.NET Caches

▶ Task 1: Test the All Photos page with no caching.

1. On the DEBUG menu of the PhotoSharingApplication – Microsoft Visual Studio window, click Start Debugging.
2. On the Welcome to Adventure Works Photo Sharing page, click the **Tools** button, and then click **F12 developer tools**.
3. On the **Cache** menu of the developer window, click **Always refresh from server**.
4. On the **Network** tab of the developer window, click **Start capturing**.
5. On the Welcome to Adventure Works Photo Sharing page, click the **All Photos** link.
6. When the page is fully loaded, in the developer window, click **Stop capturing.**
7. In the **URL** column of the developer window, click **http://localhost:**<*portnumber*>**/Photo**, and then click **Go to detailed view**.
8. On the **Timings** tab, click the Request entry, and then, in the **Duration** column, note the time taken.
9. On the **Network** tab of the developer window, click **Clear**, and then click **Start capturing**.
10. On the Welcome to Adventure Works Photo Sharing page, click the **All Photos** link.
11. When the page is fully loaded, in the developer window, click **Stop capturing**.
12. In the **URL** column of the developer window, click **http://localhost:**<*portnumber*>**/Photo**, and then click **Go to detailed view.**
13. On the **Timings** tab, click the **Request** entry, and then, in the **Duration** column, note that the time taken is reduced.
14. In the Windows Internet Explorer window, click the **Close** button.

▶ Task 2: Configure caching for the Index action.

1. In the Solution Explorer pane, under Controllers, click **PhotoController.cs**.
2. In the PhotoController.cs code window, locate the following code.

```
using System.Web.Mvc;
```

3. Place the mouse cursor at the end of the located code, press Enter, and then type the following code.

```
using System.Web.UI;
```

4. In the PhotoController.cs code window, locate the following code.

```
public ActionResult Index()
```

5. Place the mouse cursor before the located code, type the following code, and then press Enter.

```
[OutputCache(Duration=600, Location=OutputCacheLocation.Server, VaryByParam="none")]
```

6. On the **FILE** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Save All**.

▶ Task 3: Retest the All Photos page with Index caching.

1. On the DEBUG menu of the PhotoSharingApplication - Microsoft Visual Studio window, click Start Debugging.

2. On the **Cache** menu of the developer window, click **Always refresh from server**.
3. On the **Network** tab, click **Start capturing**.
4. On the Welcome to Adventure Works Photo Sharing page, click the **All Photos** link.
5. When the page is fully loaded, in the developer window, click **Stop capturing.**
6. In the **URL** column of the developer window, click **http://localhost:**<*portnumber*>**/Photo**, and then click **Go to detailed view**.
7. On the **Timings** tab, click the **Request** entry, and then, in the **Duration** column, note the time taken.
8. On the **Network** tab, click **Clear**, and then click **Start capturing**.
9. On the Welcome to Adventure Works Photo Sharing page, click the **All Photos** link.
10. When the page is fully loaded, in the developer window, click **Stop capturing**.
11. In the **URL** column of the developer window, click **http://localhost:**<*portnumber*>**/Photo**, and then click **Go to detailed view**.
12. On the **Timings** tab, click the **Request** entry, and then, in the **Duration** column, note that the time taken is reduced.
13. On the **Network** tab of the developer window, click **Back to summary view**.
14. In the **URL** column, click **/Photo/GetImage/1**, and then click **Go to detailed view**.
15. Click the **Request** entry, and then, in the **Duration** column, note the duration.
16. In the Windows Internet Explorer window, click the **Close** button.

▶ Task 4: Configure caching for the GetImage action.

1. In the **PhotoController.cs** code window, locate the following code.

```
public FileContentResult GetImage (int id)
```

2. Place the mouse cursor immediately before the located code, type the following code, and then press Enter.

```
[OutputCache(Duration=600, Location=OutputCacheLocation.Server, VaryByParam="id")]
```

3. On the **FILE** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Save All**.

▶ Task 5: Retest the All Photo page with GetImage caching.

1. On the **DEBUG** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Start Debugging**.
2. On the **Cache** menu of the developer window, click **Always refresh from server**.
3. On the **Network** tab, click **Start capturing**.
4. On the Welcome to Adventure Works Photo Sharing page, click the **All Photos** link.
5. When the page is fully loaded, in the developer window, click **Stop capturing.**
6. In the **URL** column of the developer window, click **/Photo/GetImage/1**, and then click **Go to detailed view**.
7. On the **Timings** tab, click the **Request** entry, and then, in the **Duration** column, note the duration.
8. Click **Clear**, and then click **Start capturing**.

9. On the Welcome to Adventure Works Photo Sharing page, click the **All Photos** link.
10. When the page is fully loaded, in the developer window, click **Stop Capturing**.
11. In the **URL** column of the developer window, click **/Photo/GetImage/1**, and then click **Go to detailed view**.
12. On the **Timings** tab, click the **Request** entry, and then, in the **Duration** column, note that the time taken is reduced.
13. On the **File** menu of the developer window, click **Exit**.
14. In the Windows Internet Explorer window, click the **Close** button.
15. In the **PhotoSharingApplication - Microsoft Visual Studio** window, click the **Close** button.

Results: At the end of this exercise, you will create a Photo Sharing application with the Output Cache configured for caching photos.

Module 10: Using JavaScript and jQuery for Responsive MVC 4 Web Applications

Lab: Using JavaScript and jQuery for Responsive MVC 4 Web Applications

Exercise 1: Creating and Animating the Slideshow View

▶ Task 1: Import and test the slideshow view.

1. In the result pane of the Hyper-V Manager console, in the **Name** list of the **Virtual Machines** area, right-click **20486B-SEA-DEV11**, and then click **Connect**.
2. To log on to **20486B-SEA-DEV11**, click the **Ctrl+Alt+Delete** button.
3. In the **User name** box, type **Admin**, in the **Password** box, type **Pa$$w0rd**, and then click the **Forward** button.
4. On the taskbar, click the **File Explorer** icon.
5. In the Libraries window, navigate to Allfiles (D):\Labfiles\Mod10\Starter \PhotoSharingApplication, and then double-click PhotoSharingApplication.sln.
6. In the Solution Explorer pane of the, **PhotoSharingApplication - Microsoft Visual Studio** window, under PhotoSharingApplication, expand **Views**, right-click **Photo**, point to **Add**, and then click **Existing Item**.
7. In the **Add Existing Item – PhotoSharingApplication** dialog box, navigate to **Allfiles (D):\Labfiles\Mod10\Slide Show View**, click **SlideShow.cshtml**, and then click **Add**.
8. In the Solution Explorer pane, expand **Controllers**, and then click **PhotoControllers.cs**.
9. In the PhotoControllers.cs code window, locate the following code.

```
throw new NotImplementedException("The Slideshow action is not yet ready");
```

10. Replace the code with the following code.

```
return View("SlideShow", context.Photos.ToList());
```

11. In the Solution Explorer pane, click **Mvc.sitemap**.
12. In the Mvc.sitemap code window, locate the following code.

```
<mvcSiteMapNode title="Add a Photo" visibility="*" controller="Photo" action="Create"
/>
```

13. Place the mouse cursor at the end of the located code, press Enter, and then type the following code.

```
<mvcSiteMapNode title="Slideshow" visibility="*" controller="Photo"
action="SlideShow" />
```

14. On the **DEBUG** menu of the **PhotoSharingApplication – Microsoft Visual Studio** window, click **Start Debugging**.
15. On the Adventure Works Photo Sharing page, click the **Tools** button, and then click **F12 developer tools**.
16. On the **Cache** menu of the developer window, click **Clear browser cache**.

17. A message stating "Are you sure you want to clear the browser cache?" is displayed. Click **Yes** to clear the browser cache.
18. On the **File** menu of the developer window, click **Exit**.
19. On the Adventure Works Photo Sharing page, click the **Slideshow** link.

Note: Note that the Slide Show view displays all the photos in a large size, one below the other.

20. In the Windows Internet Explorer window, click the **Close** button.

▶ Task 2: Modify the style sheet.

1. In the Solution Explorer pane, expand **Content**, and then click **PhotoSharingStyles.css**.
2. In the PhotoSharingStyles.css code window, locate the following code.

```
#slide-show DIV.slide-show-card {
```

3. Place the mouse cursor at the end of the located code, press Enter, and then type the following code.

```
position: absolute;
top: 0;
left: 0;
z-index: 8;
```

4. In the PhotoSharingStyles.css code window, locate the following code.

```
#slideshow-progress-bar-container {
```

5. Place the mouse cursor immediately before the located code, type the following code, and then press Enter.

```
#slide-show DIV.active-card {
   z-index: 10;
}
```

6. Place the mouse cursor at the end of the code you just typed, press Enter twice, and then type the following code.

```
#slide-show DIV.last-active-card {
   z-index: 9;
}
```

7. On the **DEBUG** menu of the **PhotoSharingApplication – Microsoft Visual Studio** window, click **Start Debugging**.
8. On the Adventure Works Photo Sharing page, click the **Tools** button, and then click **F12 developer tools**.
9. On the **Cache** menu of the developer window, click **Clear browser cache**.
10. A message stating "Are you sure you want to clear the browser cache?" is displayed. Click **Yes** to clear the browser cache.
11. On the **File** menu of the developer window, click **Exit**.
12. On the Adventure Works Photo Sharing page, click the **Slideshow** link.

Note: Note that only **Sample Photo 13** is displayed. The other photos are rendered in the same absolute position behind **Sample Photo 13**. Your script will use the Z-order of different style classes to move each photo to the top of the stack.

13. In the Windows Internet Explorer window, click the **Close** button.

▶ Task 3: Animate the photo cards in the slideshow.

1. In the Solution Explorer pane, right - click **PhotoSharingApplication**, point to **Add**, and then click **New Folder**.
2. In the **NewFolder1** box, type **Scripts**, and then press Enter.
3. In the Solution Explorer pane, right-click **Scripts**, point to **Add**, and then click **New Item**.
4. In the **Add New Item – PhotoSharingApplication** dialog box, click **JavaScript File**.
5. In the **Name** box, type **SlideShow.js**, and then click **Add**.
6. In the SlideShow.js code window, type the following code.

```
var percentIncrement;
var percentCurrent = 100;
```

7. Place the mouse cursor at the end of the variable you just created, press Enter twice, and then type the following code.

```
function slideSwitch () {
}
```

8. In the **slideSwitch()** function code block, type the following code.

```
var $activeCard = $('#slide-show DIV.active-card');
```

9. Place the mouse cursor at the end of the code you just typed, press Enter, and then type the following code.

```
if ($activeCard.length == 0) {
   $activeCard = $('#slide-show DIV.slide-show-card:last');
}
```

10. Place the mouse cursor at the end of the **if** statement you just created, press Enter, and then type the following code.

```
var $nextCard = $activeCard.next();
```

11. Place the mouse cursor at the end of the code you just typed, press Enter, and then type the following code.

```
if ($nextCard.length == 0) {
   $nextCard = $('#slide-show DIV.slide-show-card:first');
}
```

12. Place the mouse cursor at the end of the **if** statement you just created, press Enter, and then type the following code.

```
$activeCard.addClass('last-active-card');
```

Note: Note that this **last-active-card** class applies the z-order value 9, from the style sheet.

13. Place the mouse cursor at the end of the last-**active-card** class you just added, press Enter, and then type the following code.

```
$nextCard.css({ opacity: 0.0 });
```

14. Place the mouse cursor at the end of the code you just typed, press Enter, and then type the following code.

```
$nextCard.addClass('active-card');
```

Note: Note that this **active-card** class applies the z-order value 10, from the style sheet.

15. Place the mouse cursor at the end of the **active-card** class you just added, press Enter, and then type the following code.

```
$nextCard.animate({ opacity: 1.0 }, 1000, function () {
   $activeCard.removeClass('active-card last-active-card');
});
```

16. Place the mouse cursor at the end of the **slideSwitch()** function, press Enter twice, and then type the following code.

```
$(document).ready(function() {
});
```

17. In the anonymous function you just created, type the following code.

```
setInterval("slideSwitch()", 5000);
```

18. On the **FILE** menu of the **PhotoSharingApplication – Microsoft Visual Studio** window, click **Save All**.

▶ Task 4: Link to the script and test the animation.

1. In the Solution Explorer pane, under Photo, click **SlideShow.cshtml**.
2. In the SlideShow.cshtml code window, locate the following code.

```
<h2>Slideshow</h2>
```

3. Place the mouse cursor immediately before the located code, type the following code, and then press Enter twice.

```
<script type="text/javascript" src="@Url.Content("~/Scripts/SlideShow.js")"></script>
```

4. On the **DEBUG** menu of the **PhotoSharingApplication – Microsoft Visual Studio** window, click **Start Debugging**.
5. On the Adventure Works Photo Sharing page, click the **Slideshow** link.

Note: The script fades between different photos in the application.

6. In the Windows Internet Explorer window, click the **Close** button.

Results: At the end of this exercise, you will have created a Photo Sharing application with a slideshow page that displays all the photos in the application, sequentially.

Exercise 2: Optional—Adding a jQueryUI ProgressBar Widget

▶ Task 1: Complete the slideshow view and template view.

1. In the Solution Explorer pane, under **Photo**, click **SlideShow.cshtml**.
2. In the SlideShow.cshtml code window, locate the following code.

```
<div id="slideshow-progress-bar-container">
```

3. Place the mouse cursor at the end of the located code, press Enter, and then type the following code.

```
<div id="slideshow-progress-bar"></div>
```

4. In the Solution Explorer pane, under Views, expand **Shared**, and then click **_MainLayout.cshtml**.
5. In the _MainLayout.cshtml code window, locate the following code.

```
</head>
```

6. Place the mouse cursor immediately before the located code, type the following code, and then press Enter.

```
<script type="text/javascript"
src="http://ajax.aspnetcdn.com/ajax/jquery.ui/1.10.0/jquery-ui.min.js"></script>
```

7. In the _MainLayout.cshtml code window, locate the following code.

```
<link type="text/css" rel="stylesheet" href="~/Content/PhotoSharingStyles.css">
```

8. Place the mouse cursor immediately before the located code, type the following code, and then press Enter.

```
<link type="text/css" rel="stylesheet"
href="http://code.jquery.com/ui/1.9.2/themes/base/jquery-ui.css" />
```

9. On the **FILE** menu of the **PhotoSharingApplication – Microsoft Visual Studio** window, click **Save All**.

▶ Task 2: Modify the slideshow script.

1. In the Solution Explorer pane, under Scripts, click **SlideShow.js**.
2. In the SlideShow.js code window, locate the following code.

```
$(document).ready(function() {
```

3. Place the mouse cursor immediately before the located code, press Enter twice, and then type the following code.

```
function calculateIncrement() {
}
```

4. Place the cursor within the **calculateIncrement()** function you just created, and then type the following code.

```
var cardCount = $('#slide-show DIV.slide-show-card').length;
```

5. Place the mouse cursor at the end of the variable you just created, press Enter, and then type the following code.

```
percentIncrement = 100 / cardCount;
```

6. Place the mouse cursor at the end of the code you just typed, press Enter, and then type the following code.

```
$('#slideshow-progress-bar').progressbar({
   value: 100
});
```

Note: Note that the jQueryUI **progressbar()** function displays a progress bar widget.

7. In the SlideShow.js code window, locate the following code.

```
setInterval("slideSwitch()", 5000);
```

8. Place the mouse cursor immediately before the located code, type the following code, and then press Enter.

```
calculateIncrement();
```

9. In the SlideShow.js code window, locate the following code.

```
var $activeCard = $('#slide-show DIV.active-card');
```

10. Place the mouse cursor immediately before the located code, type the following code, and then press Enter.

```
percentCurrent += percentIncrement;
```

11. Place the mouse cursor at the end of the code you just typed, press Enter, and then type the following code.

```
if (percentCurrent > 100) {
   percentCurrent = percentIncrement;
}
```

12. Place the mouse cursor at the end of the **if** statement you just added, press Enter, and then type the following code.

```
$('#slideshow-progress-bar').progressbar({
   value: percentCurrent
});
```

13. On the **FILE** menu of the **PhotoSharingApplication – Microsoft Visual Studio** window, click **Save All**.

▶ Task 3: Test the slideshow view.

1. On the **DEBUG** menu of the **PhotoSharingApplication – Microsoft Visual Studio** window, click **Start Debugging**.
2. On the Adventure Works Photo Sharing page, click the **Tools** button, and then click **F12 developer tools**.
3. On the **Cache** menu of the developer window, click **Clear browser cache**.
4. A message stating "Are you sure you want to clear the browser cache?" is displayed. Click **Yes** to clear the browser cache.

5. On the **File** menu of the developer window, click **Exit**.
6. On the Adventure Works Photo Sharing page, click the **Slideshow** link.

Note: On the **Slideshow** tab of the Adventure Works Photo Sharing page, note that all the photos in the application are displayed, with a fade animation and a progress bar.

7. In the Windows Internet Explorer window, click the **Close** button.
8. In the **PhotoSharingApplication – Microsoft Visual Studio** window, click the **Close** button.

Results: At the end of this exercise, you will have created a slideshow page with a progress bar that displays the position of the current photo in the list of photos.

Module 11: Controlling Access to ASP.NET MVC 4 Web Applications

Lab: Controlling Access to ASP.NET MVC 4 Web Applications

Exercise 1: Configuring Authentication and Membership Providers

▶ Task 1: Configure a new Windows Azure SQL database.

1. In the result pane of the Hyper-V Manager console, in the **Name** list of the **Virtual Machines** area, right-click **20486B-SEA-DEV11**, and then click **Connect**.
2. To log on to **20486B-SEA-DEV11**, click the **Ctrl+Alt+Delete** button.
3. In the **User name** box, type **Admin**, in the **Password** box, type **Pa$$w0rd**, and then click the **Forward** button.
4. On the Windows 8 Start screen, click **Desktop**.
5. On the taskbar, click the **Internet Explorer** icon.
6. In the Address bar of the Internet Explorer window, type **http://www.windowsazure.com/**, and then press Enter.
7. On the Windows Azure page, click **PORTAL**.
8. In the **someone@example.com** box, type <*username*>.
9. In the **Password** box, type <*password*>, and then click **Sign in**.
10. In the left pane of the Windows Azure page, click **SQL DATABASES**, and then click **NEW**.
11. In the **New** dialog box, click **SQL DATABASE**, and then click **CUSTOM CREATE**.
12. In the **Name** box of the **NEW SQL DATABASE - CUSTOM CREATE** dialog box, type **PhotoAppServices**.
13. In the **SERVER** box, click **New SQL Database Server**, and then click the **Next** button.
14. In the **LOGIN NAME** box, type <*your first name*>.
15. In the **LOGIN PASSWORD** box, type **Pa$$w0rd**.
16. In the **LOGIN PASSWORD CONFIRMATION** box, type **Pa$$w0rd**.
17. In the **REGION** box, click <*a region close to you*>, and then click the **Complete** button.

 **Note:** In the sql databases result pane, note that a new database, PhotoAppServices, is created.

18. In the **Name** column of the sql databases page, click **PhotoAppServices**, and then click the **DASHBOARD** tab.
19. In the **quick glance** section, click the **Manage allowed IP addresses** link.
20. In the **RULE NAME** box of the result window, type **First Address Range**.
21. In the **START IP ADDRESS** box, type <*first address in range*>.
22. In the **END IP ADDRESS** box, type <*last address in range*>.

23. In the lower pane of the **SQL Databases – Windows Azure** window, click **Save**.

Note: To add more IP address ranges, you can repeat the steps 18–23.

▶ Task 2: Install universal providers.

1. On the taskbar, click the **File Explorer** icon.
2. In the Libraries window, navigate to Allfiles (D):\Labfiles\Mod11\Starter \PhotoSharingApplication, and then double-click PhotoSharingApplication.sln.
3. On the **PROJECT** menu of the **PhotoSharingApplication – Microsoft Visual Studio** window, click **Manage NuGet Packages**.
4. In the navigation pane of the **PhotoSharingApplication – Manage NuGet Packages** dialog box, click **Online**.
5. In the **Search Online** box of the **PhotoSharingApplication – Manage NuGet Packages** dialog box, type **Microsoft.aspnet.providers**.
6. On the **PhotoSharingApplication – Manage NuGet Packages** dialog box, click **Microsoft ASP.NET Universal Providers**, and then click **Install**.
7. In the **License Acceptance** dialog box, click **I Accept**.
8. When the installation is complete, in the **PhotoSharingApplication – Manage NuGet Packages** dialog box, click **Close**.

▶ Task 3: Configure providers in Web.config.

1. In the Solution Explorer pane of the **PhotoSharingApplication – Microsoft Visual Studio** window, expand **PhotoSharingApplication**, and then click **Web.config**.
2. In the **<connectionStrings>** element of the Web.config code window, locate the following code, select the located code, and then press Delete.

```
<add name="DefaultConnection" providerName="System.Data.SQLClient"
connectionString="Data Source=.\SQLEXPRESS;Initial Catalog=aspnet-
PhotoSharingApplication-<id>;Integrated Security=SSPI" />
```

3. Switch to Internet Explorer.
4. In the Internet Explorer window, click the **Back** button.
5. In the **NAME** column of the sql databases result pane, click **PhotoAppServices**.
6. On the photoappservices page, click **DASHBOARD**.
7. In the quick glance section of the photoappservices page, click the **Show connection strings** link.
8. In the **ADO.NET** box of the **Connection Strings** dialog box, select the given text, right-click the selected text, and then click **Copy**.
9. On the taskbar, click the **PhotoSharingApplication - Microsoft Visual Studio icon**.
10. In the Web.config code window, place the mouse cursor before the **</connectionStrings>** tag, press Enter, and then type the following code.

```
<add name="PhotoAppServices"
  connectionString = ""
  providerName =
     "System.Data.SqlClient" />
```

11. In the connectionStrings code block you just typed, place the mouse cursor within the quotes after **connectionString =**, right-click, and then click **Paste**.
12. In the content you just pasted, locate the text **{your_password_here}**, and then replace the text with **Pa$$w0rd**.
13. In the Web.config code window, locate the following code.

```
<system.web>
```

14. Place the mouse cursor at the end of the located code, press Enter, and then type the following code.

```
<authentication mode="Forms">
</authentication>
```

15. Place the cursor within the **<authentication>** element you just created, and then type the following code.

```
<forms loginUrl="~/Account/Login" timeout="2880" />
```

16. In the Web.config code window, locate the following code.

```
<add name="DefaultProfileProvider" type="System.Web.Providers.DefaultProfileProvider,
System.Web.Providers, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35" connectionStringName="DefaultConnection"
applicationName="/"/>
```

17. Within the located code, select the following text.

```
DefaultConnection
```

18. Replace the selected text with the following text.

```
PhotoAppServices
```

19. In the Web.config code window, locate the following code.

```
<add name="DefaultMembershipProvider"
type="System.Web.Providers.DefaultMembershipProvider, System.Web.Providers,
Version=1.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35"
connectionStringName="DefaultConnection" enablePasswordRetrieval="false"
enablePasswordReset="true" requiresQuestionAndAnswer="false"
requiresUniqueEmail="false" maxInvalidPasswordAttempts="5"
minRequiredPasswordLength="6" minRequiredNonalphanumericCharacters="0"
passwordAttemptWindow="10" applicationName="/" />
```

20. Within the located code, select the following text.

```
DefaultConnection
```

21. Replace the selected text with the following text.

```
PhotoAppServices
```

22. In the Web.config code window, locate the following code.

```
<add name="DefaultRoleProvider" type="System.Web.Providers.DefaultRoleProvider,
System.Web.Providers, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35" connectionStringName="DefaultConnection"
applicationName="/"/>
```

23. Within the located code, select the following text.

```
DefaultConnection
```

24. Replace the selected text with the following text.

```
PhotoAppServices
```

25. In the Web.config code window, locate the following code.

```
<add name="DefaultSessionProvider"
type="System.Web.Providers.DefaultSessionStateProvider, System.Web.Providers,
Version=1.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35"
connectionStringName="DefaultConnection"/>
```

26. Within the located code, select the following text.

```
DefaultConnection
```

27. Replace the selected text with the following text.

```
PhotoAppServices
```

28. On the **FILE** menu of the **PhotoSharingApplication – Microsoft Visual Studio** window, click **Save All**.

Results: At the end of this exercise, you will have created a Photo Sharing application that is configured to use Windows Azure SQL database for user accounts and membership information. In subsequent exercises, you will add model classes, actions, and views to implement authentication for this database.

Exercise 2: Building the Logon and Register Views

▶ Task 1: Add account model classes.

1. In the Solution Explorer pane, right-click **Models**, point to **Add**, and then click **Class**.
2. In the **Name** box of the **Add New Item – PhotoSharingApplication** dialog box, type **AccountModelClasses.cs**, and then click **Add**.
3. In the AccountModelClasses.cs code window, locate the following code.

```
using System.Web;
```

4. Place the mouse cursor at the end of the located code, press Enter, and then type the following code.

```
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;
```

5. In the AccountModelClasses.cs code window, locate the following code.

```
public class AccountModelClasses
```

6. Replace the located code with the following code.

```
public class UsersContext : DbContext
```

7. Place the mouse cursor within the **UsersContext** class code block you just created, and then type the following code.

```
public UsersContext ()
   : base("PhotoAppServices")
{
}
```

8. Place the mouse cursor after the **UsersContext** class code block, press Enter twice, and then type the following code.

```
public class Login
{
}
```

9. Place the mouse cursor within the **Login** class code block you just created, and then type the following code.

```
[Required]
[Display(Name = "User name")]
public string UserName { get; set; }
```

10. Place the mouse cursor at the end of the **UserName** property code block you just created, press Enter twice, and then type the following code.

```
[Required]
[DataType(DataType.Password)]
public string Password { get; set; }
```

11. Place the mouse cursor at the end of the **Password** property code block you just created, press Enter twice, and then type the following code.

```
[Display(Name = "Remember me?")]
public bool RememberMe { get; set; }
```

12. Place the mouse cursor after the **Login** class code block, press Enter twice, and then type the following code.

```
public class Register
{
}
```

13. Place the mouse cursor within the **Register** class code block you just created, and then type the following code.

```
[Required]
[Display(Name = "User name")]
public string UserName { get; set; }
```

14. Place the mouse cursor at the end of the **UserName** property code block you just created, press Enter twice, and then type the following code.

```
[Required]
[DataType(DataType.Password)]
public string Password { get; set; }
```

15. Place the mouse cursor at the end of the **Password** property code block you just created, press Enter twice, and then type the following code.

```
[DataType(DataType.Password)]
[Display(Name = "Confirm password")]
[Compare("Password", ErrorMessage = "The password and confirmation password do not
match.")]
public string ConfirmPassword { get; set; }
```

16. On the **FILE** menu of the **PhotoSharingApplication – Microsoft Visual Studio** window, click **Save All**.

▶ Task 2: Add an account controller.

1. In the Solution Explorer pane, right-click **Controllers**, point to **Add**, and then click **Controller**.
2. In the **Controller name** box of the **Add Controller** dialog box, type **AccountController**.
3. In the **Template** box, ensure that the **Empty MVC controller** template is selected, and then click **Add**.
4. In the AccountController.cs code window, locate the following code, select the code, and then press Delete.

```
//
//GET: /Account/
public ActionResult Index ()
{
   return View();
}
```

5. Locate the following code.

```
using System.Web.Mvc;
```

6. Place the mouse cursor at the end of the located code, press Enter, and then type the following code.

```
using System.Web.Security;
using PhotoSharingApplication.Models;
```

7. Place the mouse cursor within the **AccountController** class code block, and then type the following code.

```
public ActionResult Login(string returnUrl)
{
}
```

8. Place the mouse cursor within the **Login** action method code block, and then type the following code.

```
ViewBag.ReturnUrl = returnUrl;
return View();
```

9. Place the mouse cursor within the **AccountController** class but outside any action method code block, and then type the following code.

```
[HttpPost]
public ActionResult Login(Login model, string returnUrl)
{
}
```

10. Place the mouse cursor within the **Login** action method code block for the **HTTP POST** verb, and then type the following code.

```
if (ModelState.IsValid)
{
}
```

11. Place the mouse cursor within the **if** code block you just added, and then type the following code.

```
if (Membership.ValidateUser(model.UserName, model.Password))
{
}
else
{
}
```

12. Place the cursor within the **if** statement code block you just added, and then type the following code.

```
FormsAuthentication.SetAuthCookie(model.UserName, model.RememberMe);
```

13. Place the mouse cursor at the end of the code you just typed, press Enter, and then type the following code.

```
if (Url.IsLocalUrl(returnUrl))
{
   return Redirect(returnUrl);
}
else
{
   return RedirectToAction("Index", "Home");
}
```

14. Place the mouse cursor within the empty **else** clause you created in the first instance, and then type the following code.

```
ModelState.AddModelError("", "The username or password is incorrect");
```

15. Place the mouse cursor at the end of the **Login** action method for the **HTTP POST** verb, outside any of the **if** statements, and then type the following code.

```
return View(model);
```

16. Place the mouse cursor within the **AccountController** class but outside any action method, and then type the following code.

```
public ActionResult LogOff()
{
}
```

17. Place the mouse cursor within the **LogOff** action code block you just created, and then type the following code.

```
FormsAuthentication.SignOut();
return RedirectToAction("Index", "Home");
```

18. Place the cursor within the **AccountController** class but outside any action method, and then type the following code.

```
public ActionResult Register()
{
}
```

19. Place the mouse cursor within the **Register** action code block you just created, and then type the following code.

```
return View();
```

20. Place the mouse cursor within the **AccountController** class but outside any action method code block, and then type the following code.

```
[HttpPost]
public ActionResult Register(Register model)
{
}
```

21. Place the mouse cursor within the **Register** action method code block for the **HTTP POST** verb, and then type the following code.

```
if (ModelState.IsValid)
{
}
```

22. Place the mouse cursor within the **if** code block you just created, and then type the following code.

```
try
{
}
catch (MembershipCreateUserException e)
{
}
```

23. Place the mouse cursor within the **try** code block you just created, and then type the following code.

```
MembershipUser NewUser = Membership.CreateUser(model.UserName, model.Password);
```

24. Place the mouse cursor at the end of the code you just typed, press Enter, and then type the following code.

```
FormsAuthentication.SetAuthCookie(model.UserName, false);
```

```
return RedirectToAction("Index", "Home");
```

25. Place the mouse cursor within the **catch** code block you created, and then type the following code.

```
ModelState.AddModelError("Registration Error", "Registration error: " +
e.StatusCode.ToString());
```

26. Place the mouse cursor at the end of the **Register** action method for the **HTTP POST** verb, outside any of the **if** statements, and then type the following code.

```
return View(model);
```

27. On the **FILE** menu of the **PhotoSharingApplication – Microsoft Visual Studio** window, click **Save All**.

▶ Task 3: Import Logon and Register views.

1. In the Solution Explorer pane, right-click **Views**, point to **Add**, and then click **New Folder**.
2. In the **NewFolder1** box, type **Account**, and then press Enter.
3. Under Views, right-click **Account**, point to **Add**, and then click **Existing Item**.
4. In the **Add Existing Item – PhotoSharingApplication** dialog box, navigate to **Allfiles (D):\Labfiles\Mod11\Account Views**, click **Login.cshtml**, and then click **Add**.
5. In the Solution Explorer pane, right-click **Account**, point to **Add**, and then click **Existing Item**.
6. In the **Add Existing Item – PhotoSharingApplication** dialog box, navigate to **Allfiles (D):\Labfiles\Mod11\Account Views**, click **Register.cshtml**, and then click **Add**.

▶ Task 4: Add authentication controls to the Template view.

1. In the Solution Explorer pane, under Views, expand **Shared**, and then click **_MainLayout.cshtml**.
2. In the _MainLayout.cshtml code window, locate the following code.

```
<div class="clear-floats" />
```

3. Place the mouse cursor immediately before the located code, press Enter, and then type the following code.

```
<div class="login-controls">
</div>
```

4. Place the mouse cursor within the **DIV** element you just created, and then type the following code.

```
@if (Request.IsAuthenticated)
{
}
else
{
}
```

5. Place the mouse cursor within the **@if** statement code block you just created, and then type the following code.

```
@:Hello, @User.Identity.Name
```

6. Place the mouse cursor at the end of the greeting message you just created, press Enter, and then type the following code.

```
@Html.ActionLink("Log Off", "LogOff", "Account")
```

7. Place the mouse cursor within the **else** code block you created, and then type the following code.

```
@Html.ActionLink("Log In", "Login", "Account")
```

8. Place the mouse cursor at the end of the code you just typed, press Enter, and then type the following code.

```
@Html.ActionLink("Register", "Register", "Account")
```

9. On the **FILE** menu of the **PhotoSharingApplication – Microsoft Visual Studio** window, click **Save All**.

▶ Task 5: Test registration, log on, and log off.

1. On the **DEBUG** menu of the **PhotoSharingApplication – Microsoft Visual Studio** window, click **Start Debugging**.
2. On the Welcome to Adventure Works Photo Sharing page, click **Register**.
3. In the **User name** box of the Register section, type **David Johnson**.
4. In the **Password** box, type **Pa$$w0rd**.
5. In the **Confirm password** box, type **Pa$$w0rd**, and then click **Register**.

Note: At the upper-right corner of the Welcome to Adventure Works Photo Sharing page, note that the authentication controls show your first name to indicate that you are logged on.

6. On the Welcome to Adventure Works Photo Sharing page, click the **Log Off** link.

Note: Before clicking the **Log In** link, click any other link or menus to ensure that you have successfully logged off.

7. On the Welcome to Adventure Works Photo Sharing page, click the **Log In** link
8. In the **User name** box of the Login section, type **David Johnson**.
9. In the **Password** box, type **Pa$$w0rd**, and then click **Log in**.

Note: At the upper-right corner of the Welcome to Adventure Works Photo Sharing page, if you see your first name in the authentication controls, it implies that the model classes, controller, and views are working as expected.

10. In the Windows Internet Explorer window, click the **Close** button.

Results: At the end of this exercise, you will create a Photo Sharing application in which users can register for an account, log on, and log off.

Exercise 3: Authorizing Access to Resources

▶ Task 1: Restrict access to Photo actions.

1. In the Solution Explorer pane of the **PhotoSharingApplication - Microsoft Visual Studio** window, under Controllers, click **PhotoController.cs**.
2. In the PhotoController.cs code window, locate the following code.

```
public ActionResult Create()
```

3. Place the mouse cursor before the located code, type the following code, and then press Enter.

```
[Authorize]
```

4. In the PhotoController.cs code window, locate the following code.

```
public ActionResult Create(Photo photo, HttpPostedFileBase image)
```

5. Place the mouse cursor before the located code, type the following code, and then press Enter.

```
 [Authorize]
```

6. In the PhotoController.cs code window, locate the following code.

```
public ActionResult Delete(int id)
```

7. Place the mouse cursor before the located code, type the following code, and then press Enter.

```
[Authorize]
```

8. In the PhotoController.cs code window, locate the following code.

```
public ActionResult DeleteConfirmed(int id)
```

9. Place the mouse cursor before the located code, type the following code, and then press Enter.

```
[Authorize]
```

10. On the **FILE** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Save All**.

▶ Task 2: Restrict access to the Comment actions.

1. In the Solution Explorer pane of the **PhotoSharingApplication - Microsoft Visual Studio** window, under Controllers, click **CommentController.cs**.
2. In the CommentController.cs code window, locate the following code.

```
public PartialViewResult _Create(int PhotoId)
```

3. Place the mouse cursor before the located code, type the following code, and then press Enter.

```
[Authorize]
```

4. In the CommentController.cs code window, locate the following code.

```
public ActionResult Delete(int id = 0)
```

5. Place the mouse cursor before the located code, type the following code, and then press Enter.

```
[Authorize]
```

6. In the CommentController.cs code window, locate the following code.

```
public ActionResult DeleteConfirmed(int id)
```

7. Place the mouse cursor before the located code, type the following code, and then press Enter.

```
 [Authorize]
```

8. On the **FILE** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Save All**.

▶ Task 3: Restrict access to the Account actions.

1. In the Solution Explorer pane, under Controllers, click **AccountController.cs**.
2. In the AccountController.cs code window, locate the following code.

```
public class AccountController : Controller
```

3. Place the mouse cursor before the located code, type the following code, and then press Enter.

```
[Authorize]
```

4. In the AccountController.cs code window, locate the following code.

```
public ActionResult Login(string returnUrl)
```

5. Place the mouse cursor before the located code, type the following code, and then press Enter.

```
[AllowAnonymous]
```

6. In the AccountController.cs code window, locate the following code.

```
public ActionResult Login(Login model, string returnUrl)
```

7. Place the mouse cursor before the located code, type the following code, and then press Enter.

```
[AllowAnonymous]
```

8. In the AccountController.cs code window, locate the following code.

```
public ActionResult Register()
```

9. Place the mouse cursor before the located code, type the following code, and then press Enter.

```
[AllowAnonymous]
```

10. In the AccountController.cs code window, locate the following code.

```
public ActionResult Register(Register model)
```

11. Place the mouse cursor before the located code, type the following code, and then press Enter.

```
[AllowAnonymous]
```

12. On the **FILE** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Save All**.

▶ Task 4: Check authentication status in a view.

1. In the Solution Explorer pane, under **Views**, under **Shared**, click **_CommentsForPhoto.cshtml**.
2. In the _CommentsForPhoto.cshtml code window, locate the following code.

```
@Html.Action("_Create", "Comment", new { PhotoID = ViewBag.PhotoId })
```

3. Place the mouse cursor before the located code, type the following code, and then press Enter.

```
@if (Request.IsAuthenticated)
{
```

4. Place the mouse cursor after the located code, press Enter, and then type the following code.

```
}
```

5. In the _CommentsForPhoto.cshtml code window, at the end of the code you just typed, press Enter, and then type the following code.

```
else
{
   @Html.ActionLink("To comment you must log in", "Login", "Account");
}
```

6. On the **FILE** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Save All**.

▶ Task 5: Test authorization.

1. On the **DEBUG** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Start Debugging**.
2. On the Welcome to Adventure Works Photo Sharing page, click **Add a Photo**.

 **Note:** ASP.NET forwards the browser to the **Login** action of the **Accounts** controller because the request is unauthenticated.

3. On the Login page, click **All Photos**.
4. On the All Photos page, click the **Display** link corresponding to a photo of your choice.
5. On the result page, view the comments for the photo.

 **Note:** A link to the **Login** action is displayed at the lower end of the page, instead of the comment creation form.

6. On the result page, click the **To comment you must log in** link.
7. On the Login page, in the **User name** box, type **David Johnson**, in the **Password** box, type **Pa$$w0rd**, and then click **Log in**.
8. On the Welcome to Adventure Works Photo Sharing page, click the **Display** link corresponding to a photo of your choice.
9. On the result page, view the comments for the photo.

Note: The create comment form is displayed because the request is authenticated.

10. In the Comments section of the result page, in the **Subject** box, type **Authenticated Test Comment**.
11. In the **Body** box, type <*a comment of your choice*>, and then click **Create**.
12. In the Windows Internet Explorer window, click the **Close** button.

Results: At the end of this exercise, you will have authorized anonymous and authenticated users to access resources in your web application.

Exercise 4: Optional—Building a Password Reset View

▶ Task 1: Add a local password model class.

1. In the Solution Explorer pane of the **PhotoSharingApplication - Microsoft Visual Studio** window, under **Models**, click **AccountModelClasses.cs**.
2. In the AccountModelClasses.cs code window, place the mouse cursor in the **PhotoSharingApplication.Models** namespace code block, but outside any class code block, and then type the following code.

```
public class LocalPassword
{
}
```

3. In the **LocalPassword** class code block, type the following code.

```
[Required]
[DataType(DataType.Password)]
[Display(Name = "Current password")]
public string OldPassword { get; set; }
```

4. Place the mouse cursor at the end of the **OldPassword** property code block that you just created, press Enter twice, and then type the following code.

```
[Required]
[DataType(DataType.Password)]
[Display(Name = "New password")]
public string NewPassword { get; set; }
```

5. Place the mouse cursor at the end of the **NewPassword** property code block you just created, press Enter twice, and then type the following code.

```
[DataType(DataType.Password)]
[Display(Name = "Confirm new password")]
[Compare("NewPassword", ErrorMessage = "The password and confirmation password do not
match.")]
public string ConfirmPassword { get; set; }
```

6. On the **FILE** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Save All**.

▶ Task 2: Add reset password actions.

1. In the Solution Explorer pane, under Controllers, click **AccountControllers.cs**.
2. In the AccountControllers.cs code window, place the mouse cursor in the **AccountController** class code block, but outside any action method code block, press Enter, and then type the following code.

```
public enum ManageMessageId
{
   ChangePasswordSuccess,
   SetPasswordSuccess,
}
```

3. Place the mouse cursor at the end of the **ManageMessageId** code block, press Enter, and then type the following code.

```
public ActionResult ResetPassword (ManageMessageId? message)
{
}
```

4. In the **ResetPassword** action code block, type the following code.

```
if (message != null)
{
   ViewBag.StatusMessage = "Your password has been changed";
}
```

5. In the **ResetPassword** action code block, place the mouse cursor at the end of the code you just typed, press Enter, and then type the following code.

```
ViewBag.ReturnUrl = Url.Action("ResetPassword");
```

6. In the **ResetPassword** action code block, place the mouse cursor at the end of the code you just typed, press Enter, and then type the following code.

```
return View();
```

7. In the AccountControllers.cs code window, place the mouse cursor within the **AccountController** class, but outside any action method code block, press Enter, and then type the following code.

```
[HttpPost]
public ActionResult ResetPassword (LocalPassword model)
{
}
```

8. In the **ResetPassword** action code block for the HTTP POST verb, type the following code.

```
ViewBag.ReturnUrl = Url.Action("ResetPassword");
```

9. In the **ResetPassword** action code block for the HTTP POST verb, press Enter, and then type the following code.

```
if (ModelState.IsValid)
{
}
```

10. In the **if** statement code block, type the following code.

```
bool changePasswordSucceeded;
try
{
}
catch (Exception)
{
}
```

11. In the **try** code block, type the following code.

```
changePasswordSucceeded = Membership.Provider.ChangePassword(User.Identity.Name,
model.OldPassword, model.NewPassword);
```

12. In the **catch** block, type the following code.

```
changePasswordSucceeded = false;
```

13. Place the cursor at the end of the **try…catch** code block, press Enter, and then type the following code.

```
if (changePasswordSucceeded)
{
```

```
   return RedirectToAction("ResetPassword", new { message =
ManageMessageId.ChangePasswordSuccess });
}
```

14. Place the mouse cursor at the end of the **if** statement code block, press Enter, and then type the following code.

```
else
{
   ModelState.AddModelError("", "The current password is incorrect or the new
password is invalid");
}
```

15. Place the mouse cursor at the end of the **ResetPassword** action code block for the **HTTP POST** verb, but outside the **if** statements, press Enter, and then type the following code.

```
return View(model);
```

16. On the **FILE** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Save All**.

▶ Task 3: Import the reset password view.

1. In the Solution Explorer pane, under Views, right-click **Account**, point to **Add**, and then click **Existing Item**.
2. In the **Add Existing Item - PhotoSharingApplication** dialog box, navigate to **Allfiles (D):\Labfiles\Mod11\Account Views**, click **ResetPassword.cshtml**, and then click **Add**.

Note: In the Solution Explorer pane, note that the **ResetPassword.cshtml** file is added to the **Account** folder.

▶ Task 4: Add a link to the reset password view.

1. In the Solution Explorer pane, under Shared, click **_MainLayout.cshtml**.
2. In the _MainLayout.cshtml code window, locate the following code.

```
@Html.ActionLink("Log Off", "LogOff", "Account")
```

3. Place the mouse cursor at the end of the located code, press Enter, and then type the following code.

```
@Html.ActionLink("Reset", "ResetPassword", "Account")
```

4. On the **FILE** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Save All**.

▶ Task 5: Test password reset.

1. On the **DEBUG** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Start Debugging**.
2. On the Welcome to Adventure Works Photo Sharing page, click the **Log In** link.
3. On the logon page, in the **User name** box, type **David Johnson**, in the **Password** box, type **Pa$$w0rd**, and then click **Log in**.
4. On the Welcome to Adventure Works Photo Sharing page, click the **Reset** link.
5. On the ResetPassword page, in the **Current password** box, type **Pa$$w0rd**.

6. In the **New password** box, type **Pa$$w0rd2**, in the **Confirm new password** box, type **Pa$$w0rd2**, and then click **Change Password**.

Note: The message, "Your password has been changed" is displayed.

7. In the Windows Internet Explorer window, click the **Close** button.
8. In the **PhotoSharingApplication - Microsoft Visual Studio** window, click the **Close** button.
9. In the **SQL Databases – Windows Azure** window, click the **Close** button.

Results: At the end of this exercise, you will build a Photo Sharing application in which registered users can reset their own password.

Module 12: Building a Resilient ASP.NET MVC 4 Web Application

Lab: Building a Resilient ASP.NET MVC 4 Web Application

Exercise 1: Creating Favorites Controller Actions

▶ Task 1: Create the Favorites Slideshow action.

1. In the result pane of the Hyper-V Manager console, in the **Name** list of the **Virtual Machines** area, right-click **20486B-SEA-DEV11**, and then click **Connect**.
2. To log on to **20486B-SEA-DEV11**, click the **Ctrl+Alt+Delete** button.
3. In the **User name** box, type **Admin**, in the **Password** box, type **Pa$$w0rd**, and then click the **Forward** button.
4. On the Start screen, click **Desktop**.
5. On the taskbar, click the **File Explorer** icon.
6. In the Libraries window, navigate to Allfiles (D:)\Labfiles\Mod12\Starter \PhotoSharingApplication, and then double-click PhotoSharingApplication.sln.
7. In the Solution Explorer pane of the **PhotoSharingApplication - Microsoft Visual Studio** window, expand **PhotoSharingApplication**, expand **Controllers**, and then click **PhotoController.cs**.
8. In the PhotoController.cs code window, locate the following code.

```
public ActionResult SlideShow()
{
   return View("SlideShow", context.Photos.ToList());
}
```

9. Place the mouse cursor at the end of the located code, press Enter twice, and then type the following code.

```
public ActionResult FavoritesSlideShow()
{
}
```

10. Place the cursor within the **FavoritesSlideShow** action code block you just created, and then type the following code.

```
List<Photo> favPhotos = new List<Photo>();
```

11. Place the mouse cursor at the end of the **favPhotos** list code block you just created, press Enter, and then type the following code.

```
List<int> favoriteIds = Session["Favorites"] as List<int>;
```

12. Place the mouse cursor at the end of the **favoriteIds** list code block you just created, press Enter, and then type the following code.

```
if (favoriteIds == null)
{
   favoriteIds = new List<int>();
```

```
}
```

13. Place the mouse cursor at the end of the **if** statement code block you just created, press Enter, and then type the following code.

```
Photo currentPhoto;
```

14. Place the mouse cursor at the of the **currentPhoto** object code block you just created, press Enter, and then type the following code.

```
foreach (int currentId in favoriteIds)
{
}
```

15. Place the mouse cursor within the **foreach** code block you just created, and then type the following code.

```
currentPhoto = context.FindPhotoById(currentId);
```

16. Place the mouse cursor at the end of the code you just created, press Enter, and then type the following code.

```
if (currentPhoto != null)
{
   favPhotos.Add(currentPhoto);
}
```

17. Place the mouse cursor at the end of the **foreach** code block you created earlier, press Enter, and then type the following code.

```
return View("SlideShow", favPhotos);
```

Note: You do not need to create a new view for the **FavoritesSlideShow** action. Instead, you can re-use the **SlideShow.cshtml** view by passing a different list of **Photo** objects.

18. On the **FILE** menu of the **PhotoSharingApplication – Microsoft Visual Studio** window, click **Save All**.

▶ Task 2: Create the Add Favorite action.

1. Place the mouse cursor immediately after the **FavoritesSlideShow** action code block, but inside the **PhotoController** class code block, and then type the following code.

```
public ContentResult AddFavorite(int PhotoId)
{
}
```

2. Place the cursor within the **AddFavorite** action code block you just created, and then type the following code.

```
List<int> favoriteIds = Session["Favorites"] as List<int>;
```

3. Place the mouse cursor at the end of the **favoriteIds** list code block you just created, press Enter, and then type the following code.

```
if (favoriteIds == null)
{
```

```
    favoriteIds = new List<int>();
}
```

4. Place the mouse cursor at the end of the code block you just created, press Enter, and then type the following code.

```
favoriteIds.Add(PhotoId);
```

5. Place the mouse cursor at the end of the code you just created, press Enter, and then type the following code.

```
Session["Favorites"] = favoriteIds;
```

6. Place the mouse cursor at the end of the code you just created, press Enter, and then type the following code.

```
return Content("The picture has been added to your favorites", "text/plain",
System.Text.Encoding.Default);
```

7. On the **FILE** menu of the **PhotoSharingApplication – Microsoft Visual Studio** window, click **Save All**.

Results: After completing this exercise, you will be able to create controller actions that store values in the session state of the web application, and retrieve values from the same session state.

Exercise 2: Implementing Favorites in Views

▶ Task 1: Add an AJAX action link in the Photo Display view.

1. In the Solution Explorer pane of the **PhotoSharingApplication - Microsoft Visual Studio** window, expand **Views**, expand **Photo**, and then click **Display.cshtml**.
2. In the Display.cshtml code window, locate the following code.

```
<div>
   <span class="display-label">
      @Html.DisplayNameFor(model => model.CreatedDate):
   </span>
   <span class="display-field">
      @Html.DisplayFor(model => model.CreatedDate)
   </span>
</div>
```

3. Place the mouse cursor at the end of the located code, press Enter, and then type the following code.

```
<div id="addtofavorites">
</div>
```

4. In the **DIV** element you just created, type the following code.

```
@Ajax.ActionLink("Add this photo to your favorites",
   "AddFavorite",
   "Photo",
   new { PhotoId = Model.PhotoID },
   new AjaxOptions {
      UpdateTargetId = "addtofavorites",
      HttpMethod = "GET",
      InsertionMode = InsertionMode.Replace
   })
```

5. On the **FILE** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Save All**.

▶ Task 2: Add a link and update the site map.

1. In the Solution Explorer pane, under Views, expand **Shared**, and then click **_MainLayout.cshtml**.

Note: In the **<script>** tag of the _MainLayout.cshtml code window, ensure that the version number for jquery.ui is same as the version number noted in the Manage NuGet package. If not, update the version number provided in the **<script>** tag.

2. In the _MainLayout.cshtml code window, locate the following code.

```
<div id="breadcrumb">
   Current Location: @Html.MvcSiteMap().SiteMapPath()
</div>
```

3. Place the mouse cursor at the end of the located code, press Enter, and then type the following code.

```
@if (Session["Favorites"] != null) {
}
```

4. In the **@if** statement code block, type the following code.

```
<div id="favorites-link">
```

```
</div>
```

5. In the **DIV** element you just created, type the following code.

```
@Html.ActionLink("Favorite Photos", "FavoritesSlideShow", "Photo")
```

6. On the **FILE** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Save All**.
7. In the Solution Explorer pane, click **Mvc.sitemap**.
8. In the Mvc.sitemap code window, locate the following code.

```
<mvcSiteMapNode title="Slideshow" visibility="*" controller="Photo"
action="SlideShow" />
```

9. Place the mouse cursor at the end of the located code, press Enter, and then type the following code.

```
<mvcSiteMapNode title="Favorites" visibility="SiteMapPathHelper,!*"
controller="Photo" action="FavoritesSlideShow" />
```

10. On the **FILE** menu of the **PhotoSharingApplication – Microsoft Visual Studio** window, click **Save All**.

▶ Task 3: Test favorites.

1. On the **DEBUG** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Start Debugging**.

Note: On the Welcome to Adventure Works Photo Sharing page, note that there is no link to the **FavoritesSlideShow** action.

2. On the Welcome to Adventure Works Photo Sharing page, click **All Photos**.
3. On the All Photos page, click the **Display** link corresponding to any photo of your choice.

Note: Ensure that the photo you have selected is not already added to the favorites list.

4. On the result page, click the **Add this photo to your favorites** link.

Note: The message, "The picture has been added to your favorites" is displayed. This indicates that the AJAX action displays a confirmation message without a full-page reload.

5. On the result page, click the **Back to List** link.
6. On the All Photos page, click the **Display** link corresponding to any photo of your choice.

Note: Ensure that the photo you have selected is not already added to the favorites list.

7. On the result page, click the **Add this photo to your favorites** link, and then click the **Back to List** link.
8. On the All Photos page, click the **Display** link corresponding to any photo of your choice.

Note: Ensure that the photo you selected is not already added to the favorites list.

9. On the result page, click the **Add this photo to your favorites** link, and then click the **Back to List** link.
10. On the All Photos page, click **Home**, and then click the **Favorite Photos** link.

Note: The slideshow displays the photos that were marked as favorites.
On the Home page, if you are not able to view the **Favorite Photos** link, click the **Refresh** button.

11. In the Windows Internet Explorer window, click the **Close** button.
12. In the **PhotoSharingApplication - Microsoft Visual Studio** window, click the **Close** button.

Results: After completing this exercise, you will be able to :

Create the user interface components for the favorite photos functionality.

Test the functionality of the user interface components.

Module 13: Using Windows Azure Web Services in ASP.NET MVC 4 Web Applications

Lab: Using Windows Azure Web Services in ASP.NET MVC 4 Web Applications

Exercise 1: Accessing Windows Azure and Bing Maps

▶ Task 1: Install the Windows Azure SDK.

1. In the result pane of the Hyper-V Manager console, in the **Name** list of the **Virtual Machines** area, right-click **20486B-SEA-DEV11**, and then click **Connect**.
2. To log on to **20486B-SEA-DEV11**, click the **Ctrl+Alt+Delete** button.
3. In the **User name** box, type **Admin**, in the **Password** box, type **Pa$$w0rd**, and then click the **Forward** button.
4. On the Windows 8 Start screen, click **Desktop**.
5. On the taskbar, click the **Windows Internet Explorer** icon.
6. In the Address bar, type **http://www.microsoft.com/web/downloads/platform.aspx**, and then click **Go to**.
7. On the Download the Microsoft Web Platform page, click **Free Download**.
8. In the Navigation bar, click **Run**.
9. In the **Search** box of the **Web Platform Installer 4.5** window, type **Windows Azure SDK**, it will search automatically.
10. In the **Web Platform Installer 4.5** window, click **Windows Azure SDK for .NET (VS 2012)** <*latest version*>, click **Add**, and then click **Install**.
11. In the **PREREQUISITES** page of the Web Platform Installer 4.5 wizard, click **I Accept**.
12. On the **FINISH** page, click **Finish**.
13. In the **Web Platform Installer 4.5** window, click **Exit**.

▶ Task 2: Create a Bing Maps developer account.

1. Switch to Windows Internet Explorer.
2. In the Address bar, type **https://www.bingmapsportal.com**, and then click the **Go to** button.
3. On the Home – Bing Maps Account Center page, click the **Create a Bing Maps Account** link.
4. On the Sign in to your Microsoft account page, in the **someone@example.com** box, type <*Your Windows Live account name*>, and then, in the **Password** box, type <*Your Windows Live account password*>.
5. On the Sign in to your Microsoft account page, select the **Keep me signed in** check box, and then click **Sign in**.
6. In the Maps Account Center page, click **Continue**.
7. On the Account Details page, in the **Account Name** box, type <*Your account name*>, and then in the **Contact Name** box, type <*Your name*>.

8. In the **Email Address** box, ensure that the text, <*Your Windows Live account name*> appears, select the **I agree to the Bing Maps** check box, and then click **Save**.
9. In the result window, under Account, click **Update or view account details**.

▶ Task 3: Create a Bing Maps key.

1. In the My Account section of the Create key - Bing Maps Account Center page, click **Create or view keys**.
2. In the **Application Name** box of the Create key - Bing Maps Account Center page, type **Photo Sharing Application**, in the **Key Type** box, ensure that the value is **Trial**, and then, in the **Application type** box, ensure that the value is **Public website**.
3. In the **Enter the characters you see** box, type <*the characters seen in the image*>, and then click **Submit**.

Note: On the Create Key - Bing Maps Account Center page, the message, "Key created successfully" is displayed.

Results: After completing this exercise, you will be able to register an application to access the Bing Maps APIs.

Exercise 2: Creating a WCF Service for Windows Azure

▶ Task 1: Add a new Windows Azure Cloud Service project to the web application.

1. On the taskbar, click the **File Explorer** icon.
2. In the **Libraries** window, navigate to **Allfiles (D):\Labfiles\Mod13\Starter\PhotoSharingApplication**, and then double-click **PhotoSharingApplication.sln**.
3. On the **FILE** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, point to **Add**, and then click **New Project**.
4. In the navigation pane of the **Add New Project** dialog box, under Installed, expand **Visual C#**, and then click **Cloud**.
5. In the **Add New Project** dialog box, click **Windows Azure Cloud Service**, in the **Name** box, type **LocationChecker**, and then click **OK**.
6. In the **.NET Framework 4.5 roles** box of the **New Windows Azure Cloud Service** dialog box, click **WCF Service Web Role**, click the **Right Arrow** button, and then click **OK**.
7. In the Solution Explorer pane of the **PhotoSharingApplication - Microsoft Visual Studio** window, right-click **WCFServiceWebRole1**, and then click **Rename**.
8. In the Solution Explorer pane, rename **WCFServiceWebRole1** as **LocationCheckerWebRole**, and then press Enter.
9. In the Solution Explorer pane, right-click **IService1.cs**, and then click **Rename**.
10. In the Solution Explorer pane, rename **IService1.cs** as **ILocationCheckerService.cs**, and then press Enter.
11. In the **Microsoft Visual Studio** dialog box, click **Yes**.
12. In the Solution Explorer pane, right-click **Service1.svc**, and then click **Rename**.
13. In the Solution Explorer pane, rename **Service1.svc** as **LocationCheckerService.svc**, and then press Enter.
14. In the Solution Explorer pane, under LocationCheckerService.svc, double-click **LocationCheckerService.svc.cs**.
15. In the LocationCheckerService.svc.cs code window, locate the following code.

```
namespace WCFServiceWebRole1
```

16. In the LocationCheckerService.svc.cs code window, right-click the located code, point to **Refactor**, and then click **Rename**.
17. In the **New name** box of the **Rename** dialog box, type **LocationCheckerWebRole**, ensure that the **Preview reference changes** check box is selected, and then click **OK**.
18. In the **Preview Changes – Rename** dialog box, click **Apply**.
19. In the **Microsoft Visual Studio** dialog box, click **Yes**.
20. In the LocationCheckerService.svc.cs code window, locate the following code.

```
public class Service1 : ILocationCheckerService
```

21. In the located code, select the text **Service 1**, right-click **Service 1**, point to **Refactor**, and then click **Rename**.

22. In the **New name** box of the **Rename** dialog box, type **LocationCheckerService**, ensure that the **Preview reference changes** check box is selected, and then click **OK**
23. In the **Preview Changes – Rename** dialog box, click **Apply**.
24. In the Solution Explorer pane, under LocationChecker, double-click **ServiceDefinition.csdef**.
25. In the ServiceDefinition.csdef code window, locate the following code.

```
<WebRole name="LocationCheckerWebRole" vmsize="Small">
```

26. In the ServiceDefinition.csdef code window, replace the located code with the following code.

```
<WebRole name="LocationCheckerWebRole" vmsize="ExtraSmall">
```

27. On the **FILE** menu of the **PhotoSharingApplication - Micorosft Visual Studio** window, click **Save All**.

▶ Task 2: Create the Location Checker Service interface.

1. In the Solution Explorer pane, under LocationCheckerWebRole, double-click **ILocationCheckerService.cs**
2. In the ILocationCheckerService.cs code window, locate the following code, select the code, and then press Delete.

```
[OperationContract]
string GetData(int value);
 [OperationContract]
CompositeType GetDataUsingDataContract(CompositeType composite);
```

3. Place the mouse cursor in the **ILocationCheckerService** interface code block, and then type the following code.

```
[OperationContract]
string GetLocation(string address);
```

4. On the **FILE** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Save All**.

▶ Task 3: Add a service reference to the Bing Maps Geocode service.

1. In the Solution Explorer pane, under **LocationCheckerWebRole**, right-click **References**, and then click **Add Service Reference**.
2. In the **Address** box of the **Add Service Reference** dialog box, type **http://dev.virtualearth.net/webservices/v1/geocodeservice/geocodeservice.svc?wsdl**, and then click **Go**.
3. In the **Namespace** box of the **Add Service Reference** dialog box, type **GeocodeService**, and then click **OK**.
4. In the Solution Explorer pane, under **LocationCheckerWebRole**, double-click **LocationCheckerService.svc**.
5. In the LocationCheckerService.svc.cs code window, locate the following code.

```
using System.Text;
```

6. Place the mouse cursor at the end of the located code, press Enter, and then type the following code.

```
using LocationCheckerWebRole.GeocodeService;
```

7. On the **FILE** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Save All**.

▶ Task 4: Write the Location Checker service.

1. In the LocationCheckerService.svc.cs code window, locate the following code, select the code, and then press Delete.

```
public string GetData(int value)
{
   return string.Format("You entered: {0}", value);
}
```

2. In the LocationCheckerService.svc.cs code window, locate the following code, select the code, and then press Delete.

```
public CompositeType GetDataUsingDataContract(CompositeType composite)
{
   if (composite == null)
   {
      throw new ArgumentNullException("composite");
   }
   if (composite.BoolValue)
   {
      composite.StringValue += "Suffix";
   }
   return composite;
}
```

3. Place the mouse cursor in the **LocationCheckService** class code block, and then type the following code.

```
public string GetLocation(string address)
{
}
```

4. In the **GetLocation** method you just created, type the following code.

```
string results = "";
```

5. On the taskbar, click the **Windows Internet Explorer** icon.
6. In the lower part of the Create Key - Bing Maps Account Center page, select the key corresponding to **Photo Sharing Application**, right-click the key, and then click **Copy**.
7. Switch to the **PhotoSharingApplication - Microsoft Visual Studio** window.
8. Place the mouse cursor at the end of the **results** variable, press Enter, and then type the following code.

```
string key = "";
```

9. Place the mouse cursor within the key variable, and then, on the **EDIT** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Paste**.
10. On the taskbar, click the **File Explorer** icon.
11. In the **PhotoSharingApplication** window, navigate to **Allfiles (D):\Labfiles\Mod13\Service Code**, and then double-click **Geocoding Code.txt**.

12. On the **Edit** menu of the **Geocoding Code.txt - Notepad** window, click **Select All**.
13. On the **Edit** menu of the **Geocoding Code.txt- Notepad** window, click **Copy**.
14. Switch to the **PhotoSharingApplication - Microsoft Visual Studio** window.
15. Place the mouse cursor at the end of the **key** variable, and then press Enter.
16. On the **EDIT** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Paste**.
17. Place the mouse cursor at the end of the code you just pasted, press Enter, and then type the following code.

```
return results;
```

18. On the **FILE** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Save All**.

▶ Task 5: Publish the Service in Windows Azure.

1. In the Solution Explorer pane, collapse **LocationCheckerWebRole**, right-click **LocationChecker**, and then click **Package**.
2. In the **Service Configuration** box of the **Package Windows Azure Application** dialog box, ensure that **Cloud** is selected, and then, in the **Build Configuration** box, click **Debug**, and then click **Package**.

 **Note:** Visual Studio creates the **.cspkg** file and the **.cscfg** file for the service and displays them in the File Explorer window.

3. On the taskbar, click the **Windows Internet Explorer** icon.
4. In the Windows Internet Explorer window, click the **New Tab** button.
5. In the Address bar of the Windows Internet Explorer window, type **http://www.windowsazure.com**, and then click **Go to**.
6. On the Home page of the **Windows Azure** window, click the **PORTAL** link.
7. In the left pane of the **Windows Azure** window, click **CLOUD SERVICES**.
8. In the lower pane of the **Cloud services – Windows Azure** window, click **NEW**, and then click **CUSTOM CREATE**.
9. In the **URL** box of the **Create a cloud service** dialog box, type <*Your Windows Live Account Name*>**LocationService**, in the **REGION/AFFINITY GROUP** box, click <*Choose a location near you*>, and then click **Complete**.
10. In the **NAME** column of the **Cloud services – Windows Azure** window, click <*Your Live Account Name*>**LocationService**.
11. On the <*Your Live Account Name*>**locationservice** page, click **INSTANCES**, click **STAGING**, and then click **UPLOAD A New staging deployment**.
12. In the **DEPLOYMENT NAME** box of the **Upload a package** dialog box, type **LocationChecker**, and then, in the **PACKAGE** box, click **FROM LOCAL**.
13. In the **Choose File to Upload** dialog box, navigate to **Allfiles (D):\Labfiles\Mod13\Starter\PhotoSharingApplication\LocationChecker\bin\Debug\app.publish**, click **LocationChecker.cspkg**, and then click **Open**.
14. In the **CONFIGURATION** box of the **Upload a package.** dialog box, click **FROM LOCAL**.

15. In the **Choose File to Upload** dialog box, click **ServiceConfiguration.Cloud.cscfg**, and then click **Open**.
16. In the **Upload a package.** dialog box, select the **Deploy even if one or more roles contain a single instance** check box, and then click **OK**.

 **Note:** It may take several minutes to fully create and deploy the service. Ensure the process is complete before you proceed to Exercise 3.

Results: After completing this exercise, you will be able to create a WCF cloud service. You will also be able to publish the WCF cloud service in Windows Azure.

Exercise 3: Calling a Web Service from Controller Action

▶ Task 1: Add a service reference to the Photo Sharing application.

1. On the taskbar, click the **Microsoft Visual Studio** icon.
2. In the Solution Explorer pane of the **PhotoSharingApplication - Microsoft Visual Studio** window, collapse **LocationChecker**, and then expand **PhotoSharingApplication**.
3. In the Solution Explorer pane, under PhotoSharingApplication, right-click **References**, and then click **Add Service Reference**.
4. On the taskbar, point to the Windows Internet Explorer icon, and then click the **Cloud services – Windows Azure** window.

 **Note:** If the **Cloud services – Windows Azure** window is not displayed, click **DASHBOARD**.

5. In the SITE URL section of the **Cloud services – Windows Azure** window, click the link provided.
6. In the result window, click the **LocationCheckerService.svc** link.
7. In the Address bar of the result window, right-click the URL, and then click **Copy**.
8. Switch to Microsoft Visual Studio.
9. In the **Address** box of the **Add Service Reference** dialog box, right-click anywhere, and then click **Paste**.
10. In the **Add Service Reference** dialog box, click **Go**.
11. In the **Namespace** box of the **Add Service Reference** dialog box, type **GeocodeService**, and then click **OK**.
12. In the Solution Explorer pane of the **PhotoSharingApplication - Microsoft Visual Studio** window, expand **Controllers**, and then double-click **PhotoController.cs**.
13. In the PhotoController.cs code window, locate the following code.

```
using PhotoSharingApplication.Models;
```

14. Place the mouse cursor at the end of the located code, press Enter, and then type the following code.

```
using PhotoSharingApplication.GeocodeService;
```

15. On the **FILE** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Save All**.

▶ Task 2: Call the WCF service from the photo create action.

1. In the PhotoController.cs code window, place the mouse cursor at the end of the **PhotoController** class but outside any action code block, and then type the following code.

```
private string CheckLocation(string location)
{
}
```

2. In the **CheckLocation** method code block, type the following code.

```
LocationCheckerServiceClient client = null;
```

3. Place the mouse cursor at the end of the **client** variable code block, press Enter, and then type the following code.

```
string response = "";
```

4. Place the mouse cursor at the end of the response variable code block, press Enter, and then type the following code.

```
try
{
}
catch (Exception e)
{
}
```

5. In the **try** code block, type the following code.

```
client = new LocationCheckerServiceClient();
```

6. Place the mouse cursor at the end of the **client** variable you just entered, press Enter, and then type the following code.

```
response = client.GetLocation(location);
```

7. In the **catch** code block, type the following code.

```
response = "Error: " + e.Message;
```

8. Place the mouse cursor after the **catch** code block but in the **CheckLocation** method, press Enter, and then type the following code.

```
return response;
```

9. In the PhotoController.cs code window, locate the following code.

```
[Authorize]
[HttpPost]
public ActionResult Create(Photo photo, HttpPostedFileBase image)
{
```

10. Place the mouse cursor at the end of the located code, press Enter, and then type the following code.

```
if (photo.Location != "")
{
}
```

11. In the **if** code block, type the following code.

```
string stringLongLat = CheckLocation(photo.Location);
```

12. Place the mouse cursor at the end of the code you just entered, press Enter, and then type the following code.

```
if (stringLongLat.StartsWith("Success"))
{
}
```

13. In the **if** statement code block you just created, type the following code.

```
char[] splitChars = {':'};
```

14. Place the mouse cursor at the end of the code you just entered, press Enter, and then type the following code.

```
string[] coordinates = stringLongLat.Split(splitChars);
```

15. Place the mouse cursor at the end of the code you just entered, press Enter, and then type the following code.

```
photo.Latitude = coordinates[1];
photo.Longitude = coordinates[2];
```

Note: Remember that the array is zero-based.

16. On the **FILE** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Save All**.

▶ Task 3: Configure the Services Database connection string.

1. On the taskbar, click the **Internet Explorer** icon.
2. In the left pane of the Windows Azure Portal window, click **SQL Databases**.
3. In the **NAME** column of the **SQL Databases – Windows Azure** window, click **PhotoAppServices**.
4. In the result window, click **DASHBOARD**.
5. Under quick glance, click **Show connection strings**.
6. Select all the text in the **ADO.NET** box.
7. Right-click the selected code, and then click **Copy**.
8. On the taskbar, click the **PhotoSharingApplication - Microsoft Visual Studio** icon.
9. In the Solution Explorer pane, double-click **web.config**.
10. In the Web.config code window, select the following code, which is the connection string value for the **PhotoSharingDB** connection string.

```
Server=tcp:{Your Azure Database URL},1433;Database=PhotoSharingDB;User ID={Your ID
Here};Password=Pa$$w0rd;Trusted_Connection=False;Encrypt=True;Connection Timeout=30;
PersistSecurityInfo=true
```

11. Right-click the selected code, and then click **Paste**.
12. In the pasted content, locate the text **{your_password_here}** and then replace it with **Pa$$w0rd**
13. On the **FILE** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Save All**.

▶ Task 4: Test the Location Checker service.

1. In the Solution Explorer pane, right-click **PhotoSharingApplication**, and then click **Set as Startup Project**.
2. On the **DEBUG** menu of the **PhotoSharingApplication – Microsoft Visual Studio** window, click **Start Debugging**.

Note: If you get error while debugging, reopen the PhotoSharingApplication.sln file, and run the web application in debugging mode.

3. On the Welcome to Adventure Works page, click **Log In**.
4. In the **User name** box of the Login page, type **David Johnson**, in the **Password** box, type **Pa$$w0rd2**, and then click **Log In**.
5. On the Welcome to Adventure Works page, click **Add a Photo**.
6. In the **Title** box of the Create New Photo page, type **Testing Locations**, and then, in the **Picture** box, click **Browse**.
7. In the **Choose File to Upload** dialog box, navigate to **Allfiles (D):\Labfiles\Mod13\Sample Photos**, click **Beach.jpg**, and then click **Open**.
8. In the **Location** box of the Create New Photo page, type **Florence, OR**, and then click **Create**.
9. On the All Photos page, click the **Display** link corresponding to the "Testing Locations" photo.
10. On the Testing Locations page, note that the Latitude and Longitude values are displayed.

Note: The latitude and longitude values are taken from the Location Checker WCF service.

11. In the Windows Internet Explorer window, click the **Close** button.
12. In the PhotoSharingApplication – Microsoft Visual Studio window, click the **Close** button.

Results: After completing this exercise, you will be able to add a service reference for a Windows Azure WCF service to a .NET Framework application. You will also be able to call a WCF service from an MVC controller action.

Module 14: Implementing Web APIs in ASP.NET MVC 4 Web Applications

Lab: Implementing APIs in ASP.NET MVC 4 Web Applications

Exercise 1: Adding a Web API to the Photo Sharing Application

▶ Task 1: Add a Photo API controller.

1. In the result pane of the Hyper-V Manager console, in the **Name** list of the **Virtual Machines** area, right-click **20486B-SEA-DEV11**, and then click **Connect**.
2. To log on to **20486B-SEA-DEV11**, click the **Ctrl+Alt+Delete** button.
3. In the **User name** box, type **Admin**, in the **Password** box, type **Pa$$w0rd**, and then click the **Forward** button.
4. On the Windows 8 Start screen, click **Desktop**.
5. On the taskbar, click the **File Explorer** icon.
6. In the **Libraries** window, navigate to **Allfiles (D):\Labfiles\Mod14\Starter\PhotoSharingApplication**, and then double-click **PhotoSharingApplication.sln**.

 **Note:** If the Windows Azure Tools dialog box appears, click **Convert the project to target Windows Azure Tools – v2.0.**, and then click **OK**. In the result Windows Internet Explorer window, click the **Close** button, and switch to the Microsoft Visual Studio window.

7. In the Solution Explorer pane of the **PhotoSharingApplication – Microsoft Visual Studio** window, under **PhotoSharingApplication**, right-click **Controllers**, point to **Add**, and then click **Controller**.
8. In the **Controller name** box of the **Add Controller** dialog box, type **PhotoApiController**.
9. In the **Template** list, click **Empty API controller**, and then click **Add**.
10. In the PhotoApiController.cs code window, locate the following code.

```
using System.Web.Http;
```

11. Place the mouse cursor at the end of the located code, press Enter, and then type the following code.

```
using PhotoSharingApplication.Models;
```

12. Place the mouse cursor within the **PhotoApiController** class code block, and then type the following code.

```
private IPhotoSharingContext context = new PhotoSharingContext();
```

13. Place the mouse cursor at the end of the code you just added, press Enter twice, and then type the following code.

```
public IEnumerable<Photo> GetAllPhotos()
{
}
```

14. Place the mouse cursor within the **GetAllPhotos** action code block you just created, and then type the following code.

```
return context.Photos.AsEnumerable();
```

15. Place the mouse cursor in the **PhotoApiController** class, but outside any action code block, and then type the following code.

```
public Photo GetPhotoById(int id)
{
}
```

16. Place the mouse cursor within the **GetPhotoById** action code block you just created, and then type the following code.

```
Photo photo = context.FindPhotoById(id);
```

17. Place the mouse cursor at the end of the code you just added, press Enter, and then type the following code.

```
if (photo == null)
{
   throw new HttpResponseException(HttpStatusCode.NotFound);
}
```

18. Place the mouse cursor at the end of the **if** statement code block you just created, press Enter, and then type the following code.

```
return photo;
```

19. Place the mouse cursor in the **PhotoApiController** class, but outside any action code block, and then type the following code.

```
public Photo GetPhotoByTitle(string title)
{
}
```

20. Place the mouse cursor within the **GetPhotoByTitle** action code block you just created, and then type the following code.

```
Photo photo = context.FindPhotoByTitle(title);
```

21. Place the mouse cursor at the end of the code you just added, press Enter, and then type the following code.

```
if (photo == null)
{
   throw new HttpResponseException(HttpStatusCode.NotFound);
}
```

22. Place the mouse cursor at the end of the **if** statement code block you just created, press Enter, and then type the following code.

```
return photo;
```

23. On the **FILE** menu of the **PhotoSharingApplication – Microsoft Visual Studio** window, click **Save All**.

▶ Task 2: Configure API routes.

1. In the Solution Explorer pane, expand **App_Start**, and then double-click **WebApiConfig.cs**.
2. In the WebApiConfig.cs code window, locate the following code, select the code, and then press Delete.

```
config.Routes.MapHttpRoute(
   name: "DefaultApi",
   routeTemplate: "api/{controller}/{id}",
   defaults: new { id = RouteParameter.Optional }
);
```

3. Place the mouse cursor within the **Register** method code block, and then type the following code.

```
config.Routes.MapHttpRoute(
   name: "PhotoApi",
   routeTemplate: "api/photos/{id}",
   defaults: new { controller = "PhotoApi", action = "GetPhotoById" },
   constraints: new { id = "[0-9]+" }
);
```

4. Place the mouse cursor at the end of the PhotoApi route code block you just added, press Enter twice, and then type the following code.

```
config.Routes.MapHttpRoute(
   name: "PhotoTitleApi",
   routeTemplate: "api/photos/{title}",
   defaults: new { controller = "PhotoApi", action = "GetPhotoByTitle" }
);
```

5. Place the mouse cursor at the end of the **PhotoTitleApi** route code block you just added, press Enter twice, and then type the following code.

```
config.Routes.MapHttpRoute(
   name: "PhotosApi",
   routeTemplate: "api/photos",
   defaults: new { controller = "PhotoApi", action = "GetAllPhotos" }
);
```

6. On the **FILE** menu of the **PhotoSharingApplication – Microsoft Visual Studio** window, click **Save All**.

▶ Task 3: Configure media-type formatters.

1. In the WebApiConfig.cs code window, place the mouse cursor at the end of the you just typed, press Enter twice, and then type the following code.

```
var json = config.Formatters.JsonFormatter;
```

2. Place the mouse cursor at the end of the **json** variable code block you just added, press Enter, and then type the following code.

```
json.SerializerSettings.PreserveReferencesHandling =
Newtonsoft.Json.PreserveReferencesHandling.Objects;
```

3. Place the mouse cursor at the end the code you just added, press Enter, and then type the following code.

```
config.Formatters.Remove(config.Formatters.XmlFormatter);
```

4. On the **FILE** menu of the **PhotoSharingApplication – Microsoft Visual Studio** window, click **Save All**.

▶ Task 4: Test the Web API with Internet Explorer.

1. On the **DEBUG** menu of the **PhotoSharingApplication – Microsoft Visual Studio** window, click **Start Debugging**.

Note: The Adventure Works Photo Sharing page is displayed.

2. In the Address bar of the Internet Explorer window, type **http://localhost:<***yourportnumber***>/api/photos/4**, and then press Enter.
3. In the lower-end of the Adventure Works Photo Sharing page, a message stating "Do you want to open or save 4.json (712 KB) from localhost?" is displayed. Click **Open** to view the file.
4. If the "How do you want to open this type of file (.json)?" message is displayed, click **More options**, and then click **Microsoft Visual Studio Version Selector**.
5. In the 4.json code window, note that the **Title** property is **Sample Photo 4**.
6. In the **4.json – Microsoft Visual Studio** window, click the **Close** button.
7. In the Address bar of the Internet Explorer window, type **http://localhost:<***yourportnumber***>/api/photos/sample photo 5**, and then press Enter.
8. In the lower-end of the Adventure Works Photo Sharing page, a message stating "Do you want to open or save sample photo 5.json (750 KB) from localhost?" is displayed. Click **Open** to view the file.
9. If the "How do you want to open this type of file (.json)?" message is displayed, click **Microsoft Visual Studio Version Selector**.
10. In the sample photo 5.json code window, note that the **Title** property is **Sample Photo 5**.
11. In the **sample photo 5.json – Microsoft Visual Studio** window, click the **Close** button.
12. In the Address bar of the Internet Explorer window, type **http://localhost:<***yourportnumber***>/api/photos**, and then press Enter.
13. In the lower-end of the Adventure Works Photo Sharing page, a message stating "Do you want to open or save photos.json (15.0 MB) from localhost?" is displayed. Click **Open** to view the file.
14. On the **EDIT** menu of the **photos.json – Microsoft Visual Studio** window, click **Find and Replace**, and then click **Quick Find**.
15. In the **Search Item** box of the **Quick Find** dialog box, type **Sample Photo 9**, click the **Find Next** button, and then click the **Close** button.

Note: In the photos.json code window, note that Visual Studio locates the text, **Sample Photo 9**. This implies that the API has included this photo in the returned photos.

16. On the **EDIT** menu of the **photos.json – Microsoft Visual Studio** window, click **Find and Replace**, and then click **Quick Find**.
17. In the **Search Item** box of the **Quick Find** dialog box, type **Sample Photo 13**, click the **Find Next** button, and then click the **Close** button.

Note: In the photos.json code window, note that Visual Studio locates the text, **Sample Photo 13**. This implies that the API has included this photo in the returned photos.

18. In the **photos.json – Microsoft Visual Studio** window, click the **Close** button.
19. In the Internet Explorer window, click the **Close** button.

Results: At the end of this exercise, you will be able to create a simple Web API for an ASP.NET MVC 4 web application.

Exercise 2: Using the Web API for a Bing Maps Display

▶ Task 1: Create a new template view.

1. In the Solution Explorer pane, expand **Views**, and then expand **Shared**.
2. In the Solution Explorer pane, under Shared, right-click **_MainLayout.cshtml**, and then click **Copy**.
3. In the Solution Explorer pane, right-click **Shared**, and then click **Paste**.
4. In the Solution Explorer pane, under Shared, right-click **Copy of _MainLayout.cshtml**, and then click **Rename**.
5. In the Solution Explorer pane, rename **Copy of _MainLayout.cshtml** as **_MapLayout.cshtml**, and then press Enter.
6. In the Solution Explorer pane, double-click **_MapLayout.cshtml**.
7. In the _MapLayout.cshtml code window, locate the following code.

```
<!DOCTYPE html>
```

8. Replace the located code with the following code.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

9. In the _MapLayout.cshtml code window, locate the following code.

```
<body>
```

10. Replace the located code with the following code.

```
<body onload="GetMap();">
```

11. In the _MapLayout.cshtml code window, locate the following code, select the code, and then press Delete.

```
<script src="http://ajax.aspnetcdn.com/ajax/jquery.ui/1.9.0/jquery-ui.min.js"
type="text/javascript"></script>
```

12. In the _MapLayout.cshtml code window, locate the following code, select the code, and then press Delete.

```
<script src="http://ajax.aspnetcdn.com/ajax/mvc/3.0/jquery.unobtrusive-ajax.js"
type="text/javascript"></script>
```

13. In the _MapLayout.cshtml code window, locate the following code.

```
</head>
```

14. Place the mouse cursor before the located code, type the following code, and then press Enter.

```
<script charset="UTF-8" type="text/javascript"
src="http://ecn.dev.virtualearth.net/mapcontrol/mapcontrol.ashx?v=7.0"></script>
```

15. On the **FILE** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Save All**.

▶ Task 2: Create a map action, view, and script file.

1. In the Solution Explorer pane, under Controllers, double-click **PhotoController.cs**.

2. In the PhotoController.cs code window, place the mouse cursor in the **PhotoController** class code block, but outside any action code block, press Enter, and then type the following code.

```
public ViewResult Map()
{
}
```

3. In the **Map** action code block you just created, type the following code.

```
return View("Map");
```

4. In the PhotoController.cs code window, locate the following code, right-click the code, and then click **Add View**.

```
public ViewResult Map()
```

5. In the **View name** box of the **Add View** dialog box, ensure that the text is **Map**, and then ensure that the **Create a Strongly-Typed View** check box is cleared.
6. In the **Add View** dialog box, ensure that the **Use a layout or master page** check box is selected, and then click the **Browse** button.
7. In the **Content of the folder** box of the **Select a Layout Page** dialog box, click **_MapLayout.cshtml**, and then click **OK**.
8. In the **Add View** dialog box, click **Add**.
9. In the Map.cshtml code window, locate the following code, select the code, and then press Delete.

```
<h2>Map</h2>
```

10. In the Map.cshtml code window, place the mouse cursor outside the Razor code block, press Enter, and then type the following code.

```
<script type="text/javascript">
</script>
```

11. Place the mouse cursor in the **SCRIPT** element, and then type the following code.

```
var webApiUrl = '@Url.Content("~/api/photos")';
```

12. Place the mouse cursor at the end of the code you just typed, press Enter, and then type the following code.

```
var pictureUrl = '@Url.Action("GetImage", "Photo")/';
```

13. Place the mouse cursor at the end of the code you just typed, press Enter, and then type the following code.

```
var displayUrl = '@Url.Action("Display", "Photo")/';
```

14. Place the mouse cursor at the end of the **</script>** tag, press Enter twice, and then type the following code.

```
<script src="" type="text/javascript"></script>
```

15. Place the mouse cursor in the empty **src** attribute you just created, and then type the following code.

```
@Url.Content("~/Scripts/MapDisplay.js")
```

16. Place the mouse cursor at the end of the JavaScript code you just created, press Enter twice, and then type the following code.

```
<div id="mapDiv" style="position: absolute; width: 650px; height: 400px;"></div>
```

17. In the Solution Explorer pane, right-click **Scripts**, point to **Add**, and then click **Existing Item**.
18. In the **Add Existing Item - PhotoSharingApplication** dialog box, navigate to **Allfiles (D):\Labfiles\Mod14\Bing Maps Script**, click **MapDisplay.js** and then click **Add**.
19. In the Solution Explorer pane, double-click **Mvc.sitemap**.
20. In the Mvc.sitemap code window, locate the following code.

```
<mvcSiteMapNode title="Add a Photo" visibility="*" controller="Photo" action="Create"
/>
```

21. Place the mouse cursor before the located code, type the following code, and then press Enter.

```
<mvcSiteMapNode title="Map" visibility="*" controller="Photo" action="Map" />
```

22. On the **FILE** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Save All**.
23. On the **DEBUG** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Start Debugging**.
24. On the Welcome to Adventure Works page, click **Map**.

Note: On the Map page, the Bing Map AJAX control is displayed. However, there are no push pins on the map.

25. In the Windows Internet Explorer window, click the **Close** button.

▶ Task 3: Obtain and display photos.

1. In the Solution Explorer pane, under **Scripts**, and then double-click **MapDisplay.js**.
2. At the end of the file, insert the following code.

```
function GetPhotos(serviceUrl) {
}
```

3. In the **GetPhotos** function code block, type the following code.

```
$.support.cors = true;
```

4. Place the mouse cursor at the end of the code you just typed, press Enter, and then type the following code.

```
$.ajax({
   url: serviceUrl,
   type: 'GET',
   dataType: 'json',
   success: DisplayPics,
   error: OnError
});
```

5. Place the mouse cursor at the end of the **GetPhotos** function code block, press Enter twice, and then type the following code.

```
function DisplayPics(response) {
}
```

6. In the **DisplayPics** function code block, type the following code.

```
var location;
var pin;
```

7. Place the mouse cursor at the end of the code you just typed, press Enter, and then type the following code.

```
$.each(response, function(index, photo) {
});
```

8. In the **$.each** function code block, type the following code.

```
location = new Microsoft.Maps.Location(photo.Latitude, photo.Longitude);
```

9. Place the mouse cursor at the end of the code you just typed, press Enter, and then type the following code.

```
pin = new Microsoft.Maps.Pushpin(location);
```

10. Place the mouse cursor at the end of the code you just typed, press Enter, and then type the following code.

```
pin.Title = photo.Title;
pin.ID = photo.PhotoID;
```

11. Place the mouse cursor at the end of the code you just typed, press Enter, and then type the following code.

```
Microsoft.Maps.Events.addHandler(pin, 'click', DisplayInfoBox);
```

12. Place the mouse cursor at the end of the code you just typed, press Enter, and then type the following code.

```
dataLayer.push(pin);
```

13. Place the mouse cursor at the end of the **DisplayPics** function code block, press Enter twice, and then type the following code.

```
function OnError(response) {
}
```

14. In the **OnError** function code block, type the following code.

```
alert("Could not obtain the picture coordinates");
```

15. In the **GetMap** function code block, locate the following code.

```
infoboxLayer.push(infoBox);
```

16. Place the mouse cursor at the end of the located code, press Enter twice, and then type the following code.

```
GetPhotos(webApiUrl);
```

17. On the **FILE** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Save All**.

▶ Task 4: Test the Bing Maps control.

1. On the **DEBUG** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Start Debugging**.
2. On the Welcome to Adventure Works page, click **Map**.

Note: On the Map of All Photos page, if the message **The specified credentials are invalid. You can sign up for a free developer account at http://www.bingmapsportal.com** is displayed, then click the **http://www.bingmapsportal.com** link, and then log on to the page by using your Windows Live credentials. Then, in the **PhotoSharingApplication - Microsoft Visual Studio** window, you should re-start the web application in debugging mode and navigate to the Map page.

Note: On the Map page, the Bing Map AJAX control is displayed along with push pins on the map.

3. On the Map page, click a pin of your choice.

Note: The info box appears, displaying the photo as a thumbnail with the title.

4. In the info box of the Map page, click the thumbnail.

Note: The web application redirects you to the **Display** view, which displays the full size photo.

5. In the Windows Internet Explorer window, click the **Close** button.
6. In the **PhotoSharingApplication - Microsoft Visual Studio** window, click the **Close** button.

Results: After completing this exercise, you will be able to create a template view to display a Bing Map AJAX control, and create a view and script file to display a Bing Map. You will also use jQuery to call a Web API and obtain details of photos. You will then mash up the data from a web API with Bing Maps data.

Module 15: Handling Requests in ASP.NET MVC 4 Web Applications

Lab: Handling Requests in ASP.NET MVC 4 Web Applications

Exercise 1: Creating a SignalR Hub

▶ Task 1: Install SignalR.

1. In the result pane of the Hyper-V Manager console, in the **Name** list of the **Virtual Machines** area, right-click **20486B-SEA-DEV11**, and then click **Connect**.
2. To log on to **20486B-SEA-DEV11**, click the **Ctrl+Alt+Delete** button.
3. In the **User name** box, type **Admin**, in the **Password** box, type **Pa$$w0rd**, and then click the **Forward** button.
4. On the Windows 8 Start screen, click **Desktop**.
5. On the taskbar, click the **File Explorer** icon.
6. In the **Libraries** window, navigate to **Allfiles (D):\Labfiles\Mod15\Starter\PhotoSharingApplication**, and then double-click **PhotoSharingApplication.sln**.

 **Note:** If the Windows Azure Tools dialog box appears, click **Convert the project to target Windows Azure Tools – v2.0.**, and then click **OK**. In the result Windows Internet Explorer window, click the **Close** button, and switch to the Microsoft Visual Studio window.

7. In the Solution Explorer pane of the **PhotoSharingApplication - Microsoft Visual Studio** window, right-click **Solution 'PhotoSharingApplication'**, and then click **Manage NuGet Packages for Solutions**.
8. In the navigation pane of the **PhotoSharingApplication.sln - Manage NuGet Packages** dialog box, under Online, click **NuGet official package source**.
9. In the **PhotoSharingApplication.sln - Manage NuGet Packages** dialog box, click inside the **Stable Only** box, and then click **Include Prerelease**.
10. In the **Search Online** box of the **PhotoSharingApplication.sln - Manage NuGet Packages** dialog box, type **Microsoft.aspnet.signalr**.
11. In the list of packages, click **Microsoft ASP.NET SignalR**, and then click **Install**.
12. If the **Select Projects** dialog box appears, clear all projects except **PhotoSharingApplication**, and then click **OK**.
13. In the **License Acceptance** dialog, click **I Accept**.
14. When the installation is complete, click **Close**.
15. In the Solution Explorer pane of the **PhotoSharingApplication - Microsoft Visual Studio** window, under **PhotoSharingApplication**, and then expand **References**.

Note: The NuGet package has added three **Microsoft.aspnet.SignalR** namespaces to the project references.

16. In the Solution Explorer pane, under PhotoSharingApplication, expand **Scripts**.

Note: The NuGet package has added five **jquery** scripts, including two **signalR** scripts, to the **Scripts** folder.

▶ Task 2: Create a Hub class.

1. In the Solution Explorer pane, right-click **PhotoSharingApplication**, point to **Add**, and then click **Class**.
2. In the **Name** box of the **Add New Item - PhotoSharingApplication** dialog box, type **ChatHub.cs**, and then click **Add**.
3. In the ChatHub.cs code window, locate the following code, select the code, and then press Delete.

```
using System.Collections.Generic;
using System.Linq;
```

4. In the ChatHub.cs code window, locate the following code.

```
using System.Web;
```

5. Place the mouse cursor at the end of the located code, press Enter, and then type the following code.

```
using System.Threading.Tasks;
using Microsoft.AspNet.SignalR;
```

6. In the ChatHub.cs code window, locate the following code.

```
public class ChatHub
```

7. In the ChatHub.cs code window, replace the located code with the following code.

```
public class ChatHub : Hub
```

8. In the **ChatHub** class code block, type the following code.

```
public Task Join(int photoId)
{
}
```

9. In the **Join** method code block, type the following code.

```
return Groups.Add(Context.ConnectionId, "Photo" + photoId);
```

10. Place the mouse cursor at the end of the **Join** method code block, press Enter twice, and then type the following code.

```
public Task Send(string username, int photoId, string message)
{
}
```

11. In the **Send** method code block, type the following code.

```
string groupName = "Photo" + photoId;
```

12. Place the mouse cursor at the end of the code you just typed, press Enter, and then type the following code.

```
return Clients.Group(groupName).addMessage(username, message);
```

13. On the **FILE** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Save All**.

▶ Task 3: Configure SignalR routes.

1. In the Solution Explorer pane, click **Global.asax**.
2. In the Global.asax code window, locate the following code.

```
using System.Data.Entity;
```

3. Place the mouse cursor at the end of the located code, press Enter, and then type the following code.

```
using Microsoft.AspNet.SignalR;
```

4. In the Global.asax code window, locate the following code.

```
AreaRegistration.RegisterAllAreas();
```

5. Place the mouse cursor at the end of the located code, press Enter twice, and then type the following code.

```
RouteTable.Routes.MapHubs();
```

6. On the **FILE** menu of the **PhotSharingApplication - Microsoft Visual Studio** window, click **Save All**.

Results: After completing this exercise, you will be able to install SignalR in an MVC 4 web application, configure routes for SignalR, and create a hub to receive and forward simple text messages.

Exercise 2: Creating a Photo Chat View

▶ Task 1: Create a chat action and view.

1. In the Solution Explore pane, expand **Controllers**, and then click **PhotoController.cs**.
2. In the PhotoController.cs code window, place the mouse cursor in the **PhotoController** class, but outside any action method code block, and then type the following code.

```
[Authorize]
public ActionResult Chat(int id)
{
}
```

3. Place the mouse cursor within the **Chat** action code block, type the following code, and then press Enter.

```
Photo photo = context.FindPhotoById(id);
```

4. In the **Chat** action code block, place the mouse cursor at the end of the code you just added, type the following code, and then press Enter.

```
if (photo == null) {
   return HttpNotFound();
}
```

5. In the **Chat** action code block, place the mouse cursor at the end of the code you just added, type the following code, and then press Enter.

```
return View("Chat", photo);
```

6. In the Solution Explorer pane, expand **Views**, right-click **Photo**, point to **Add**, and then click **Existing Item**.
7. In the **Add Existing Item – PhotoSharingApplication** dialog box, navigate to **Allfiles (D):\Labfiles\Mod15\Chat View\Chat.cshtml**, click **Chat.cshtml**, and then click **Add**.
8. On the **FILE** menu of the **PhotoSharingApplication – Microsoft Visual Studio** window, click **Save All**.

▶ Task 2: Link to the chat view.

1. In the Solution Explorer pane, under **Photo**, click **Display.cshtml**.
2. In the Display.cshtml code window, locate the following code.

```
<div id="addtofavorites">
   @Ajax.ActionLink("Add this photo to your favorites",
      "AddFavorite",
      "Photo",
      new { PhotoID = Model.PhotoID },
      new AjaxOptions {
         UpdateTargetId = "addtofavorites",
         HttpMethod = "GET",
         InsertionMode = InsertionMode.Replace
      })
</div>
```

3. Place the mouse cursor at the end of the located code, press Enter, and then type the following code.

```
<div id="chataboutthisphoto">
</div>
```

4. Place the mouse cursor within the **DIV** element code block you just added, and then type the following code.

```
@Html.ActionLink("Chat about this photo", "Chat", new { id = Model.PhotoID })
```

5. On the **DEBUG** menu of the **PhotoSharingApplication – Microsoft Visual Studio** window, click **Start Debugging**.
6. On the Welcome to Adventure Works Photo Sharing page, in the Latest Photos section, click the **Display** link corresponding to the image of your choice.
7. On the Sample Photo page, click the **Chat about this photo** link, and then note that the logon page appears.
8. In the **User name** box of the Login page, type **David Johnson**.
9. In the **Password** box, type **Pa$$w0rd2**, and then click **Log in**.

Note: If an exception appears when you log on, you need to replace the connection string in the Web.config file from the PhotoAppServices project. Also, in the connection string, you need to replace **{your_password_here}** with **Pa$$w0rd**.

10. On the Chat page, in the message box, type **Test message**, and then click **Send**.

Note: The message is not sent because the SignalR scripts are not yet written.

11. In the Internet Explorer window, click the **Close** button.

▶ Task 3: Link to JScript files.

1. In the Solution Explorer pane, under **Photo**, click **Chat.cshtml**.
2. In the Chat.cshtml code window, place the mouse cursor at the end of the view file, press Enter, and then type the following code.

```
<script type="text/javascript">
</script>
```

3. Place the mouse cursor in the **SCRIPT** element code block you just added, type the following code, and then press Enter.

```
var username = '@User.Identity.Name';
```

4. In the **SCRIPT** element, place the mouse cursor at the end of the **username** variable, press Enter, and then type the following code.

```
var photoid = @Model.PhotoID
```

5. Place the mouse cursor at the end of the **SCRIPT** element code block, press Enter, and then type the following code.

```
<script type="text/javascript" src=""></script>
```

6. Place the mouse cursor within the **src** attribute, and then type the following code.

```
@Url.Content("~/Scripts/jquery.signalR-1.0.0.js")
```

Note: NuGet installs the latest version of SignalR. Ensure that the name of the script file you enter matches the name of the file in the Scripts folder.

7. Place the mouse cursor at the end of the **SCRIPT** element code block that you just added, press Enter, and then type the following code.

```
<script type="text/javascript" src=""></script>
```

8. Place the mouse cursor within the **src** attribute, and then type the following code.

```
@Url.Content("~/signalr/hubs")
```

9. Place the mouse cursor at the end of the **SCRIPT** element code block that you just added, press Enter, and then type the following code.

```
<script type="text/javascript" src=""></script>
```

10. Place the mouse cursor within the **src** attribute, and then type the following code.

```
@Url.Content("~/Scripts/ChatRoom.js")
```

11. On the **FILE** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Save All**.

▶ Task 4: Script SignalR connections.

1. In the Solution Explorer pane, right-click **Scripts**, point to **Add**, and then click **New Item**.
2. In the **Search Installed Templates** box of the **Add New Item – PhotoSharingApplication** dialog box, type **JavaScript**.
3. In the **Add New Item – PhotoSharingApplication** dialog box, click **JavaScript File**, and then click **Add**.
4. In the Solution Explorer pane, under Scripts, right-click **JavaScript1.js**, and then click **Rename**.
5. In the Solution Explorer pane, under Scripts, rename **JavaScript1.js** as **ChatRoom**, and then press Enter.
6. In the ChatRoom.js code window, type the following code.

```
$(function() {
});
```

7. Place the mouse cursor within the anonymous function code block you just added, and then type the following code.

```
var chat = $.connection.chatHub;
```

8. Place the mouse cursor at the end of the **chat** variable code block, press Enter, and then type the following code.

```
$('#chat-message').focus();
```

9. Place the mouse cursor at the end of the code you just typed, press Enter, and then type the following code.

```
$.connection.hub.start().done(function(){
});
```

10. Place the mouse cursor within the anonymous function you just created, and then type the following code.

```
chat.server.join(photoid).done(function() {
});
```

11. On the **FILE** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Save All**.

▶ Task 5: Script SignalR messages.

1. In the **ChatRoom.js** code window, locate the following code.

```
var chat = $.connection.chatHub;
```

2. Place the mouse cursor at the end of the located code, press Enter, and then type the following code.

```
chat.client.addMessage = function (name, message) {
};
```

3. Place the mouse cursor within the anonymous function code block you just created, and then type the following code.

```
var encodedName = $('<div />').text(name).html();
```

4. Place the mouse cursor at the end of the **encodedName** variable code block, press Enter, and then type the following code.

```
var encodedMessage = $('<div />').text(message).html();
```

5. Place the mouse cursor at the end of the **encodedMessage** variable code block, press Enter, and then type the following code.

```
var listItem = '<li>' + encodedName + ': ' + encodedMessage + '</li>';
```

6. Place the mouse cursor at the end of the **listItem** variable code block, press Enter, and then type the following code.

```
$('#discussion').append(listItem);
```

7. In the ChatRoom.js code window, locate the following code.

```
chat.server.join(photoid).done(function() {
});
```

8. Place the mouse cursor within the located code, and then type the following code.

```
$('#sendmessage').click(function() {
});
```

9. Place the mouse cursor within the anonymous function you just created, and then type the following code.

```
chat.server.send(username, photoid, $('#chat-message').val());
```

10. Place the mouse cursor at the end of the code you just typed, press Enter, and then type the following code.

```
$('#chat-message').val('').focus();
```

11. On the **FILE** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Save All**.

▶ Task 6: Test the chat room.

1. On the **DEBUG** menu of the **PhotoSharingApplication - Microsoft Visual Studio** window, click **Start Debugging**.
2. On the Welcome to Adventure Works Photo Sharing page, click the **Log In** link.
3. In the **User name** box of the Login page, type **David Johnson**.
4. In the **Password** box, type **Pa$$w0rd2**, and then click **Log in**.
5. On the Welcome to Adventure Works Photo Sharing page, click the **Display** link corresponding to **Sample Photo 1**
6. On the Sample Photo 1 page, click the **Chat about this photo** link.
7. On the Chat page, in the message box, type **Test message**, and then click **Send**.

Note: The script sends the message to the hub that returns the message to all browsers currently connected to the group of this photo.

8. On the taskbar, right-click the **Internet Explorer** icon, and then click **Internet Explorer**.
9. In the Address bar of the Internet Explorer window, type **http://localhost:**<*portnumber*>**/**, and then press Enter.
10. On the Welcome to Adventure Works Photo Sharing page, click the **Log off** link.
11. On the Welcome to Adventure Works Photo Sharing page, click the **Register** link.
12. In the **User name** box of the Register page, type **Mark Steele**, in the **Password** box, type **Pa$$w0rd**.
13. In the **Confirm password** box, type **Pa$$w0rd**, and then click **Register**.
14. On the Welcome to Adventure Works Photo Sharing page, click the **Display** link corresponding to **Sample Photo 1**.
15. On the Sample Photo 1 page, click the **Chat about this photo** link.
16. On the Chat page, in the message box, type **Second test message**, and then click **Send**.

Note: On the Welcome to Adventure Works Photo Sharing page, if the user name shown is **David Johnson** and not **Mark Steele**, click the **Refresh** button before proceeding with the steps.

17. On the taskbar, point to the **Internet Explorer** icon, and then click the **David Johnson Chat – Windows Internet Explorer** icon.

Note: The new message from **Mark Steele** is displayed because both users joined the chat for **Sample Photo 1**.

18. On the Chat page, in the message box, type **Third test message**, and then click **Send**.
19. On the taskbar, point to the **Internet Explorer** icon, and then click the **Mark Steele Chat** - **Windows Internet Explorer**.

Note: The new message is displayed in Internet Explorer.

20. In the Internet Explorer windows, click the **Close** button.
21. In the **PhotoSharingApplication - Microsoft Visual Studio** window, click the **Close** button.

Results: After completing this exercise, you will be able to create MVC controller actions and views to display a user interface for SignalR functionality, link to the JScript libraries that SignalR requires, and use JScript to connect to a SignalR hub and send messages.

Module 16: Deploying ASP.NET MVC 4 Web Applications

Lab: Deploying ASP.NET MVC 4 Web Applications

Exercise 1: Deploying a Web Application to Windows Azure

▶ Task 1: Prepare the Photo Sharing application project for deployment.

1. In the result pane of the Hyper-V Manager console, in the **Name** list of the Virtual Machines area, right-click **20486B-SEA-DEV11**, and then click **Connect**.
2. To log on to **20486B-SEA-DEV11**, click the **Ctrl+Alt+Delete** button.
3. In the **User name** box, type **Admin**, in the **Password** box, type **Pa$$w0rd**, and then click the **Forward** button.
4. On the Windows 8 Start screen, click **Desktop**.
5. On the taskbar, click the **File Explorer** icon.
6. In the **Libraries** window, navigate to **Allfiles (D):\Labfiles\Mod16\Starter\PhotoSharingApplication**, and then double-click **PhotoSharingApplication.sln**.

 **Note:** If the Windows Azure Tools dialog box appears, click **Convert the project to target Windows Azure Tools – v2.0.**, and then click **OK**. In the result Windows Internet Explorer window, click the **Close** button, and switch to the Microsoft Visual Studio window.

7. In the Solution Explorer pane of the PhotoSharingApplication – Microsoft Visual Studio window, right-click **PhotoSharingApplication**, and then click **Properties**.
8. In the PhotoSharingApplication code window, ensure that the **Build** tab is selected.
9. In the **Configuration** box, click **Release**.
10. On the taskbar, click the **Internet Explorer** icon.
11. In the Address bar of the Internet Explorer window, type **http://www.windowsazure.com**, and click the **Go to** button.
12. In the upper-right corner of the Windows Azure: Microsoft's Cloud Platform | Cloud Hosting | Cloud Services page, click **PORTAL**.
13. In the **Microsoft account** box of the Sign in to your Microsoft account page, type <*Your Windows Live account name*>, in the **Password** box, type <*your password*>, and then click **Sign in**.
14. In the left pane of the Windows Azure page, click **SQL Databases**.
15. In the **Name** column, click **PhotoSharingDB**.
16. On the photosharingdb page, click **DASHBOARD**.
17. In the quick glance section of the photosharingdb page, click **Show connection strings**.
18. In the **ADO.NET** box of the **Connection Strings** dialog box, select all the text, right-click the selected text, and then click **Copy**.
19. On the taskbar, click the **PhotoSharingApplication – Microsoft Visual Studio** icon.

20. In the Solution Explorer pane, expand **PhotoSharingApplication**, and then click **Web.config**.
21. In the Web.config code window, locate the following code.

```
<add name="PhotoSharingDB" connectionString="Server=tcp:{Your Azure Database
URL},1433;Database=PhotoSharingDB;User ID={Your ID
Here};Password=Pa$$w0rd;Trusted_Connection=False;Encrypt=True;Connection
Timeout=30;PersistSecurityInfo=true" providerName="System.Data.SqlClient" />
```

22. In the located code, select the contents of the **connectionString** attribute, right-click the selected text, and then click **Paste**.
23. Within the text you just pasted, select the following text.

```
{your_password_here}
```

24. Replace the selected text with the word, **Pa$$w0rd**.
25. On the taskbar, click the **SQL Databases – Windows Azure – Windows Internet Explorer** icon.
26. In the Internet Explorer window, click the **New Tab** button.
27. In the Address bar of the Internet Explorer window, type **https://www.bingmapsportal.com**, and then click the **Go to** button.
28. On the Home – Bing Maps Account Center page, click the **Sign In** link.
29. If the Sign in to your Microsoft account page appears, in the Microsoft account box, type <*Your Windows Live account name*>, in the **Password** box, type <*Your Windows Live account password*>, and then click Sign in.
30. In the left pane of the Create Key - Bing Maps Account Center page, under My Account, click the **Create or view keys** link.
31. In the Key/URL section of the Create Key - Bing Maps Account Center page, select the key corresponding to **Photo Sharing Application**, right-click the key, and then click **Copy**.
32. On the taskbar, click the **PhotoSharingApplication – Microsoft Visual Studio** icon.
33. In the Solution Explorer pane of the PhotoSharingApplication – Microsoft Visual Studio window, expand Scripts, and then click **MapDisplay.js**.
34. In the MapDisplay.js code window, locate the following code, right-click the selected code, and then click Paste.

```
{Your Bing Key}
```

35. On the **FILE** menu of the PhotoSharingApplication – Microsoft Visual Studio window, click **Save All**.

▶ Task 2: Create a new website in Windows Azure.

1. On the taskbar, point to the **Internet Explorer** icon, and then click the **SQL Databases – Windows Azure – Windows Internet Explorer** icon.
2. In the **Connection Strings** dialog box, click **Close**.
3. In the left pane of the Windows Azure page, click the **Web Sites** icon.
4. In the lower-left pane of the Windows Azure page, click **New**, and then click **Custom Create**.
5. In the **URL** box of the **Create Web Site** dialog box, type <*your username*>**PhotoSharing**.
6. In the **REGION** box, click <*a region near you*>.
7. In the **DATABASE** box, click **PhotoSharingDB**.

8. In the **DB CONNECTION STRING NAME** box, type **PhotoSharingDB**.
9. In the **DB USER** box, type <*your first name*>.
10. In the **PASSWORD** box, type **Pa$$w0rd**, and then click the **Complete** icon.

Note: Windows Azure creates the new web application. You need to upload the Photo Sharing application project to this web application.

▶ Task 3: Deploy the Photo Sharing application.

1. On the Windows Azure page, in the **Name** column, click <*your username*>**PhotoSharing**, and then click **DASHBOARD**.
2. In the quick glance section of the <*your username*>PhotoSharing page, click **Download the publish profile**.
3. In the Navigation bar, click **Save**, and then click the **Close** button.
4. On the taskbar, click the **PhotoSharingApplication – Microsoft Visual Studio** icon.
5. In the Solution Explorer pane, right-click **PhotoSharingApplication**, and then click **Publish**.
6. On the **Profile** page of the Publish Web wizard, in the **Select or import a publish profile** box, click **Import**, and then click Browse.
7. In the **Import Publish Settings** dialog box, click <*your username*>**PhotoSharing.azurewebsites.net.PublishSettings**, and then click **Open**.
8. In the **Import Publish Profile** dialog box, click **OK**.
9. On the **Connection** page, click **Validate Connection**, and then click **Next**.
10. If the **Certification Error** dialog box appears, click **Accept**.

Note: If the **PhotoSharingContext**, the **PhotoSharingDB**, and the **UsersContext** boxes are not displayed, you need to publish the **PhotoSharingApplication** project again. Note that, at some instances the **PhotoSharingDB** box will be displayed with the 500 Error page. At these instances, you need to close the publish wizard and you need to publish the project again. At the second instance, you will be able to see all the three boxes-the **PhotoSharingContext** box, the **PhotoSharingDB** box, and the **UsersContext** box.

11. On the **Settings** page, in the **PhotoSharingContext** box, click **PhotoSharingDB** to get the following value.

    Data Source=tcp:*{Your Azure Database URL}*,1433;Initial Catalog=PhotoSharingDB; Integrated Security=False;User ID={Your ID Here};Password=Pa$$w0rd; Connection Timeout=30; Encrypt=True

12. In the **PhotoSharingDB** box, click **PhotoSharingDB** to get the following value.

    Data Source=tcp:*{Your Azure Database URL}*,1433;Initial Catalog=PhotoSharingDB; Integrated Security=False;User ID={Your ID Here};Password=Pa$$w0rd;Connection Timeout=30; Encrypt=True

13. In the **UsersContext** box, click **PhotoSharingDB** to get the following value.

Data Source=tcp:*{Your Azure Database URL}*,1433;Initial Catalog=PhotoSharingDB; Integrated Security=False;User ID={Your ID Here};Password=Pa$$w0rd; Connection Timeout=30; Encrypt=True

14. On the **Settings** page, click **Next**.
15. On the **Preview** page, click **Start Preview**.

Note: The files that require changes are displayed.

16. On the **Publish** page, click **Publish**.

Note: Visual Studio publishes the Photo Sharing web application to Windows Azure. This process may take several minutes. When the process is complete, a new tab in Windows Internet Explorer displays the Photo Sharing home page.

Results: After completing this exercise, you will be able to prepare an ASP.NET MVC web application for production deployment, create a web application in Windows Azure, and deploy an MVC web application to Windows Azure.

Exercise 2: Testing the Completed Application

▶ Task 1: Add a photo and a comment.

1. On the Welcome to Adventures Works Photo Sharing page, click the **Log In** link.
2. In the **User name** box of the Login page, type **David Johnson**, in the **Password** box, type **Pa$$w0rd2**, and then click **Log in**.
3. On the Create New Photo page, click **Add a Photo**.
4. In the **Title** box, type **Test New Photo**, and then click **Browse**.
5. In the **Choose File to Upload** dialog box, navigate to **Allfiles (D):\Labfiles\Mod16\Sample Photos**, click **track.JPG**, and then click **Open**.
6. In the **Description** box, type **This is the first photo added to the deployed web application**, and then click **Create**.
7. On the All Photos page, click the **Display** link corresponding to **Test New Photo**.
8. On the Test News Photo page, note that the metadata you entered is displayed correctly.
9. In the **Subject** box of the Comments section, type **Functionality Check**.
10. In the **Body** box, type **The photo metadata is displayed correctly**, and then click **Create**.

 **Note:** The new comment is displayed on the website.

▶ Task 2: Use the slideshow and favorites options.

1. On the Test New Photo page, click **Slideshow**, and then view a slideshow of the images.

 **Note:** The slideshow displays all the photos in the web application and includes the faded transition between photos and the jQueryUI progress bar widget.

2. On the Test New Photo page, click **All Photos**.
3. On the All Photos page, click the **Display** link corresponding to **Sample Photo 1**.
4. On the Sample Photo 1 page, click the **Add this photo to your favorites** link.
5. In the Internet Explorer window, click the **Back** button.
6. On the All Photos page, click the **Display** link corresponding to **Sample Photo 2**.
7. On the Sample Photo 2 page, click the **Add this photo to your favorites** link.
8. In the Internet Explorer window, click the **Back** button.
9. On the All Photos page, click the **Display** link corresponding to **Sample Photo 3**.
10. On the Sample Photo 3 page, click the **Add this photo to your favorites** link.
11. On the Sample Photo 3 page, click the **Favorite Photos** link, and then view a slideshow of the images that you added to the favorites list.

▶ Task 3: Test the chat room.

1. On the SlideShow page, click **All Photos**.
2. On the All Photos page, click the **Display** link corresponding to **Sample Photo 1**.

3. On the Sample Photo 1 page, click the **Chat about this photo** link.
4. On the Chat page, in the message box, type **A test message for Sample Photo 1**, and then click **Send**.

Note: Your message is displayed in the conversation.

5. On the taskbar, right-click the **Internet Explorer** icon, and then click **Internet Explorer**.
6. In the Address bar, type **http://**<*yourusername*>**PhotoSharing.azurewebsites.net**, and then click the **Go to** button.
7. On the Welcome to Adventure Works Photo Sharing page, click the **Log Off** link, and then click the **Refresh** button.
8. On the Welcome to Adventure Works Photo Sharing page, click the **Log In** link.
9. In the **User name** box of the Login page, type **Mark Steele**.
10. In the **Password** box, type **Pa$$w0rd**, and then click **Log In**.
11. On the Welcome to Adventure Works Photo Sharing page, click **All Photos**.
12. On the All Photos page, click the **Display** link corresponding to **Sample Photo 1**.

Note: Ensure that the user name, **Mark Steele** is displayed in the upper-right corner. Else, you need to refresh the page to get the correct user name.

13. On the Sample Photo 1 page, click the **Chat about this photo** link.

Note: Ensure that the user name, **Mark Steele** is displayed in the upper-right corner. Else, you need to refresh the page to get the correct user name.

14. On the Chat page, in the message box, type **This message is from Mark Steele**, and then click **Send**.
15. On the taskbar, point to the **Internet Explorer** icon, and then click the Chat window for David Johnson.

Note: The second message is displayed successfully.

16. On the Chat page, in the message box, type **This is the third test message**, and then click **Send**.
17. On the taskbar, point to the **Internet Explorer** icon, and then click the **Chat** window for Mark Steele.

Note: The third message is displayed successfully.

18. Close all the Internet Explorer windows.
19. In the **PhotoSharingApplication - Microsoft Visual Studio** window, click the **Close** button.

Results: After completing this exercise, you will be able to confirm that all the functionalities that you built in the Photo Sharing application run correctly in the Windows Azure deployment.