Module 12: **Building a Resilient ASP.NET MVC 5 Web Application**

# ****Contents:****

|  |  |
| --- | --- |
|  | [Module Overview](https://www.skillpipe.com/api/2.1/content/urn:uuid:4ed7876c-8864-4cf5-91ee-491726dfe322@2020-12-11T17:05:09Z/OPS/html/20486C12.html#P1) |
| **Lesson 1:** | [Developing Secure Sites](https://www.skillpipe.com/api/2.1/content/urn:uuid:4ed7876c-8864-4cf5-91ee-491726dfe322@2020-12-11T17:05:09Z/OPS/html/20486C12.html#P2) |
| **Lesson 2:** | [State Management](https://www.skillpipe.com/api/2.1/content/urn:uuid:4ed7876c-8864-4cf5-91ee-491726dfe322@2020-12-11T17:05:09Z/OPS/html/20486C12.html#P3) |
| **Lab:** | [Building a Resilient ASP.NET MVC 5 Web Application](https://www.skillpipe.com/api/2.1/content/urn:uuid:4ed7876c-8864-4cf5-91ee-491726dfe322@2020-12-11T17:05:09Z/OPS/html/20486C12.html#P4) |
|  | [Module Review and Takeaways](https://www.skillpipe.com/api/2.1/content/urn:uuid:4ed7876c-8864-4cf5-91ee-491726dfe322@2020-12-11T17:05:09Z/OPS/html/20486C12.html#P5) |

# ****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

**Accessing a Query String Parameter**

public ActionResult Index(string msg)

{

ViewBag.Msg = msg;

return 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

**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.

**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.

**Using an Anti-Forgery Token**

@using (Html.BeginForm())

{

@Html.AntiForgeryToken();

@Html.EditorForModel();

<input type="submit" value="Submit" />

}

**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.

**Using the ValidateInput Attribute**

[HttpPost]

[ValidateInput(false)]

public ActionResult Edit(string comment)

{

return View(comment);

}

**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. |

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 5 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.

**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 5 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-S** **ide 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 **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**

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. |

**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. |

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

**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.

**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****

|  |  |
| --- | --- |
| • | You will find the steps in the “Lesson 2: State Management“ section on the following page: <https://github.com/MicrosoftLearning/20486-DevelopingASPNETMVCWebApplications/blob/master/Instructions/20486C/20486C_MOD12_DEMO.md>. |

# ****Lab: Building a Resilient ASP.NET MVC 5 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: 35 minutes

You will find the high-level steps on the following page: <https://github.com/MicrosoftLearning/20486-DevelopingASPNETMVCWebApplications/blob/master/Instructions/20486C/20486C_MOD12_LAB_MANUAL.md>.

You will find the detailed steps on the following page: <https://github.com/MicrosoftLearning/20486-DevelopingASPNETMVCWebApplications/blob/master/Instructions/20486C/20486C_MOD12_LAK.md>.

### ****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. |

### ****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. |

### ****Review Question(s)****

**Check Your Knowledge**

**Discovery**

**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?**

Show solution Reset

**Check Your Knowledge**

**Discovery**

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

Show solution Reset

# ****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.

### ****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.

### ****Review Question(s)****