

Capítulo 6

Seguridad de Aplicaciones Web

Objetivo

Al finalizar el capítulo, el alumno:

- Implementa autenticación y autorización.
- Usa ASP.NET Identity en una aplicación MVC.
- Trabaja con Áreas.
- Evita ataques a las aplicaciones.

Temas

1. Introducción a la Seguridad en Aplicaciones
2. Autenticación y Autorización
3. ASP.NET Identity
4. Áreas
5. Cross-Site Scripting (XSS)
6. Cross-Site Request Forgery (CSRF)

1. Introducción a la Seguridad en Aplicaciones

Introducción a la Seguridad en Aplicaciones

La creación de una aplicación web segura no solo implica crear usuarios e implementar una página de login.

Es necesario familiarizarse con temas de seguridad que ofrece el sistema operativo, en este caso, Windows.



El diagrama ilustra una arquitectura de seguridad web. En el centro, un servidor web (etiquetado como 'WEB SERVER') está conectado a una base de datos ('DATABASE') y a un sistema de archivos ('FILE SYSTEM'). El servidor web también está conectado a un firewall ('FIREWALL') que protege el acceso desde Internet. El firewall está conectado a un router ('ROUTER') que gestiona el tráfico de red. El router está conectado a un switch ('SWITCH') que gestiona el tráfico local. El switch está conectado a un servidor de almacenamiento ('STORAGE SERVER'). El diagrama también muestra un servidor de correo ('MAIL SERVER') y un servidor de aplicaciones ('APPLICATION SERVER').

S - 4

Copyright © Todos los Derechos Reservados - Cibertec Perú S.A.C.

La creación de una aplicación web segura no solo implica solamente crear usuarios e implementar una página de login para ingresar a la aplicación con una contraseña; si no que, adicionalmente, requiere realizar un estudio para comprender los puntos vulnerables de la aplicación.

Además de conocer los temas de seguridad relacionados con ASP.NET y .NET Framework, es necesario familiarizarse con temas de seguridad que ofrece el sistema operativo, en este caso Windows.

La siguiente lista muestra un conjunto de medidas básicas que se deben adoptar para proteger cualquier aplicación web:

- Tener Windows, IIS, SQL Server, etc. actualizados.
- Ejecutar las aplicaciones con los privilegios mínimos necesarios (por ejemplo, nuestra aplicación web no debería conectarse con el usuario administrador a la base de datos).
- Conocer a los usuarios y manejar perfiles de privilegios.
- Tener acceso seguro a bases de datos.
- Crear mensajes de error adecuados, sin dar detalles de la implementación interna de la aplicación (por ejemplo, no se debe mostrar el stack trace de las excepciones).
- Proteger la información confidencial.
- Usar cookies de forma segura.
- Implementar técnicas para protegerse contra ataques o amenazas.

Una fase importante en el proceso de programación de aplicaciones seguras, consiste en ser capaz de anticipar las amenazas que se pueden sufrir. Microsoft ha elaborado una clasificación de las amenazas en distintas categorías:

Suplantación

Suplantar (spoof) es utilizar los datos de identificación de un usuario o proceso de forma no autorizada. En su versión más simple, la suplantación consistiría en introducir las credenciales de un usuario diferente. Un usuario malintencionado podría también cambiar el contenido de una cookie para fingir que es otra persona, o que la cookie proviene de un servidor diferente.

Es posible contribuir a evitar la suplantación mediante una autenticación estricta.

Siempre que alguien solicita acceso a información privada, es preciso asegurarse de que es quien dice ser. También, se puede contribuir a la defensa contra la suplantación manteniendo la información de credenciales a salvo. Por ejemplo, no se debe guardar nunca una contraseña ni otro tipo de datos confidenciales o privados en una cookie, donde un usuario malintencionado podría encontrarlos y modificarlos fácilmente.

Manipulación

Manipular significa cambiar o eliminar un recurso sin autorización. El ejemplo típico consiste en cambiar la configuración de una página web, para lo cual, el usuario malintencionado logra acceso al sitio y cambia algunos archivos. Un modo indirecto de manipulación son los ataques mediante secuencias de comandos. Se consigue al ejecutar código (secuencia de comandos) enmascarándolo como la entrada de datos de un usuario en una página o como un vínculo.

Una defensa fundamental contra la manipulación consiste en usar la seguridad para bloquear los archivos, directorios y otros recursos de Windows. La aplicación también debería ejecutarse con privilegios mínimos.

Repudio

Una amenaza de repudio implica llevar a cabo una transacción de manera que no haya pruebas fehacientes de los actores principales de la transacción. En una aplicación web, esto puede significar que se está suplantando a un usuario usando sus credenciales.

Para evitar este ataque se debe aplicar una autenticación estricta. Además, se deben usar las funciones de inicio de sesión de Windows para mantener un registro de auditoría de cualquier actividad en el servidor.

Revelación de información

Significa robar o desvelar información, que se supone es confidencial. Un ejemplo típico es el robo de contraseñas, pero la revelación de información, también incluye el acceso a cualquier archivo o recurso del servidor.

La mejor protección contra la revelación de información es no tener información que revelar. Por ejemplo, si se evita el almacenamiento de contraseñas, ningún usuario malintencionado podrá robarlas. Una alternativa al almacenamiento de las contraseñas consiste en guardar solo un valor hash de estas. De este modo, cuando un usuario presenta sus credenciales, se puede extraer el valor hash de su contraseña y

compararlo con el almacenado, pero si aun así, se almacena información confidencial, se debe utilizar la seguridad de Windows para ayudar a protegerla.

Como en todos los casos anteriores, se debería utilizar la autenticación para contribuir a garantizar que solo los usuarios autorizados pueden tener acceso a la información restringida.

Si se tiene que exponer información confidencial es recomendable cifrarla cuando se almacene y que se utilice SSL (Secure Sockets Layer) para cifrar la información cuando se envía al explorador o se recibe de este.

Denegación de servicio

Un ataque de denegación de servicio consiste en hacer, deliberadamente, que una aplicación esté menos disponible. Un ejemplo típico es sobrecargar una aplicación web de forma que no pueda servir a los usuarios normales. Como alternativa, los usuarios malintencionados pueden intentar bloquear el servidor.

Los servicios IIS permiten limitar a las aplicaciones de forma que solo respondan a un número determinado de solicitudes a ciertas direcciones IP, de tal manera que se pueda evitar una intención de saturar el servidor.

Concesión de privilegio

Un ataque de concesión de privilegio consiste en usar medios malintencionados para obtener más permisos de los asignados normalmente.

Por ejemplo, en un ataque de concesión de privilegio que tenga éxito, un usuario malintencionado consigue obtener privilegios administrativos para el servidor web, lo que le proporciona acceso a todos los datos del servidor, así como, el control de las funciones de este.

Como ayuda para protegerse contra ataques de concesión de privilegio, se debe ejecutar la aplicación en un contexto con los permisos mínimos, si resulta factible.

Por ejemplo, se recomienda no ejecutar las aplicaciones de ASP.NET con la cuenta de usuario SYSTEM (administrador).

2. Autenticación y Autorización

Autenticación y Autorización

En la seguridad de las aplicaciones de cualquier tipo y cualquier tecnología siempre se debe tener en cuenta dos conceptos claves, **autenticación** y **autorización**.

Autenticación

Es el proceso de identificar quién está accediendo a la aplicación. En Visual Studio 2017 se permite elegir entre:

- Individual User Accounts.
- Work And School Accounts.
- Windows Authentication



5 - 7

Copyright © Todos los Derechos Reservados - Cibertec Perú S.A.C.



En la seguridad de las aplicaciones de cualquier tipo y cualquier tecnología siempre se debe tener en cuenta dos conceptos claves, autenticación y autorización.

Autenticación

Es el proceso de identificar quién está accediendo a la aplicación. En visual Studio 2017 se permite elegir entre:

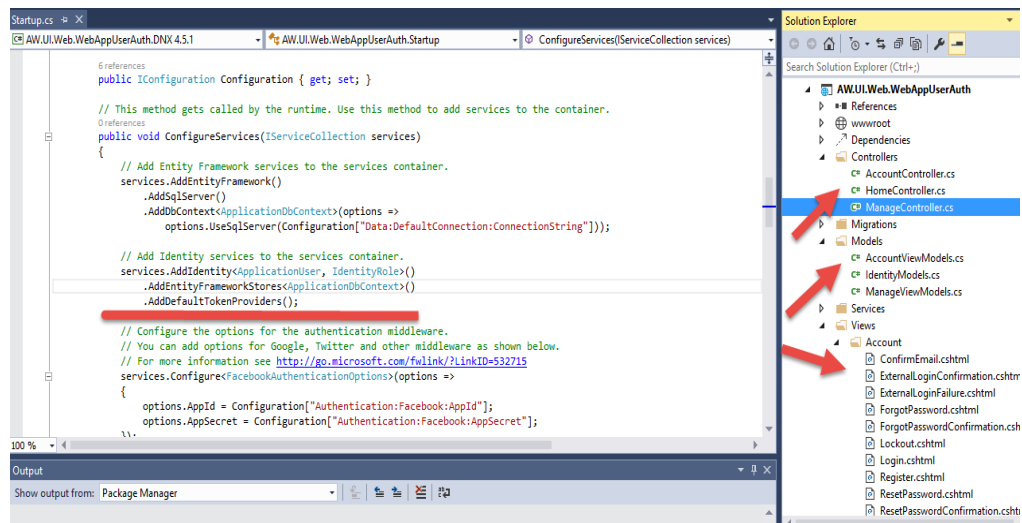
- **Individual User Accounts:** Es necesario implementar un formulario de login que solicite el usuario y contraseña, como parte de la aplicación, para aplicaciones que soporten autenticación basada en perfiles de usuarios cuya información se guarda en una base de datos de SQL Server. Mediante esta plantilla se crean todos los componentes necesarios para registrar nuevos usuarios, ingresar (Login) a la aplicación web, cambiar password, etc.

Adicional al registro de usuarios, esta plantilla permite la autenticación utilizando Facebook, Google, Microsoft Account y Twitter.

El componente que usa la plantilla para implementar la autenticación de usuarios es ASP.Net Identity (antes conocido como Membership) está basado en OWIN, lo que significa que se puede usar el mismo mecanismo en aplicaciones Web Form, MVC, WebApi, o SignalR.

En ASP.Net Identity la base de datos es administrada por Entity Framework Code First. Todas las tablas son representadas por clases que pueden ser

personalizadas para agregar campos adicionales que pueden ser considerados importantes en el perfil del usuario.



- **Work And School Accounts:** template para construir aplicaciones web basadas en autenticación con Active Directory, Microsoft Azure Active Directory y Office 365.

Change Authentication

No Authentication

Individual User Accounts

Work And School Accounts

Windows Authentication

For applications that authenticate users with Active Directory, Microsoft Azure Active Directory, or Office 365.

[Learn more](#)

Cloud - Single Organization

Domain:

cibertec.edu.pe

Directory Access Permissions:

☐ Read directory data

☒ More Options

Client Id:

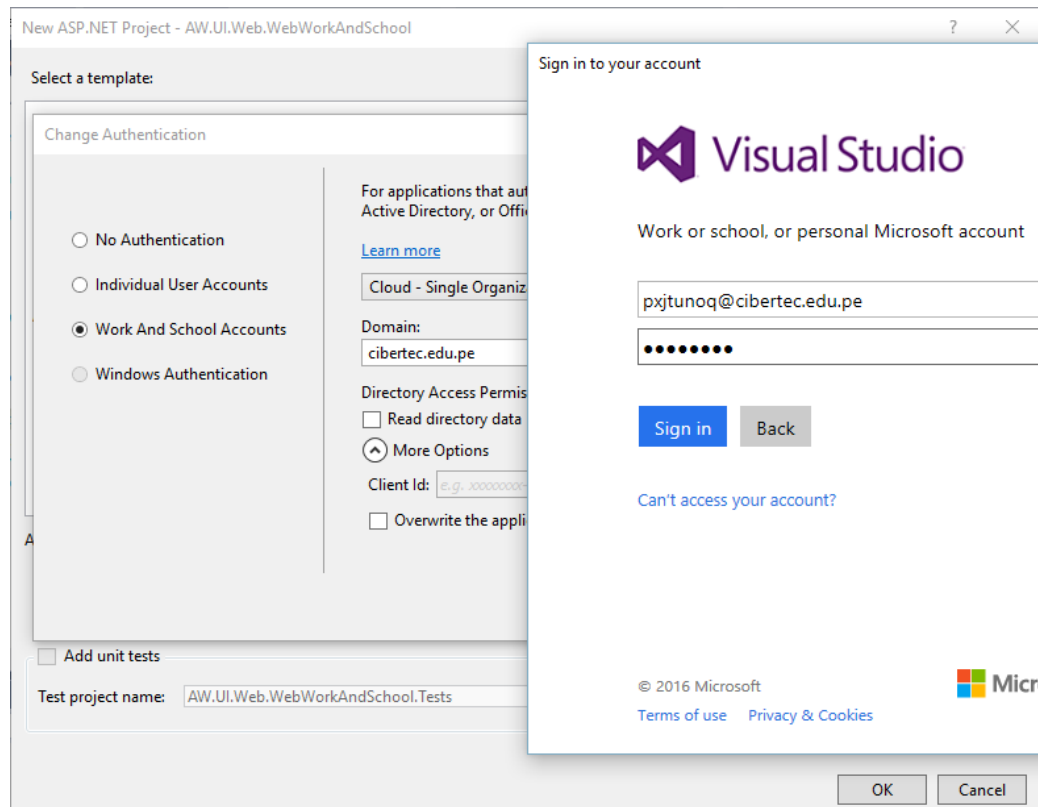
e.g. xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx

☐ Overwrite the application entry if one with same id exists

OK

Cancel

Cibertec Perú S.A.C – Visual Studio 2017 Web Developer

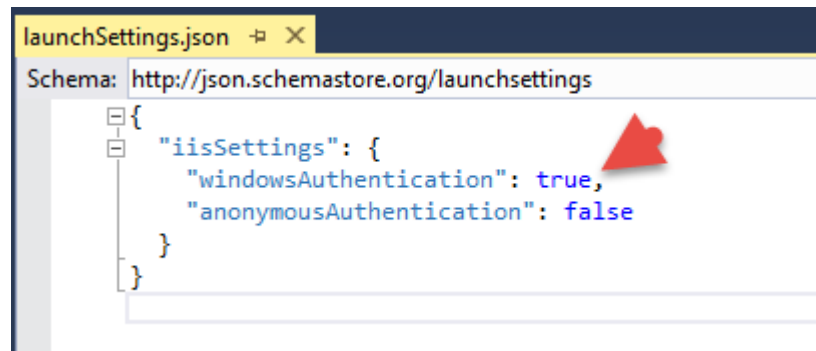


```
// This method gets called by the runtime. Use this method to add services to the container.
// References
public void ConfigureServices(IServiceCollection services)
{
    services.Configure<CookieAuthenticationOptions>(options =>
    {
        (parameter) ServiceCollection services true;
    });

    services.Configure<OpenIdConnectAuthenticationOptions>(options =>
    {
        options.AutomaticAuthentication = true;
        options.ClientId = Configuration["Authentication:AzureAd:ClientId"];
        options.Authority = Configuration["Authentication:AzureAd:AADInstance"] + Configuration["Authentication:AzureAd:TenantId"];
        options.PostLogoutRedirectUri = Configuration["Authentication:AzureAd:PostLogoutRedirectUri"];
        options.SignInScheme = CookieAuthenticationDefaults.AuthenticationScheme;
    });

    // Add MVC services to the services container.
    services.AddMvc();
}
```

- **Windows Authentication:** template para construir aplicaciones web de intranet. No solicita a los usuarios un nombre de usuario y contraseña. La información de usuario de Windows en el equipo cliente es suministrada por el navegador web a través de un intercambio criptográfico con el servidor Web. Si el intercambio de autenticación inicialmente no identifica al usuario, el navegador web le pedirá una cuenta de usuario, el Nombre de usuario y la contraseña de Windows.



Autorización

Es el proceso de determinar qué acciones puede realizar un usuario autenticado, de acuerdo a los permisos que tiene asignado.

3. ASP.NET Identity

Autenticación y Autorización

ASP.NET Identity

Para manejar de manera más rápida y fácil el aspecto de Autenticación y Autorización en aplicaciones Web, ASP.NET incluía ASP.NET Membership (hasta ASP.NET 3.5), esto incluía un panel de administración de permisos en el cual registrábamos usuarios, roles, esto se registraba en un esquema de datos ya definido. Posteriormente, se desarrolló ASP.NET Simple Membership (ASP.NET 4) y ahora se tiene **ASP.NET Identity** (ASP.NET 4.5 y 5).

ASP.NET Identity también incluye un esquema de base de datos predefinido que podemos utilizar inmediatamente con la configuración predeterminada de nuestros proyectos. Utiliza **Entity Framework Code First** para crear la base datos.

S - 9 Copyright © Todos los Derechos Reservados - Cibertec Perú SAC.

Para manejar de manera más rápida y fácil el aspecto de Autenticación y Autorización en aplicaciones Web, ASP.NET incluía ASP.NET Membership (hasta ASP.NET 3.5), esto incluía un panel de administración de permisos en el cual registrábamos usuarios, roles, esto se registraba en un esquema de datos ya definido. Posteriormente, se desarrolló ASP.NET Simple Membership (ASP.NET 4) y ahora se tiene ASP.NET Identity (ASP.NET 4.5 y 5).

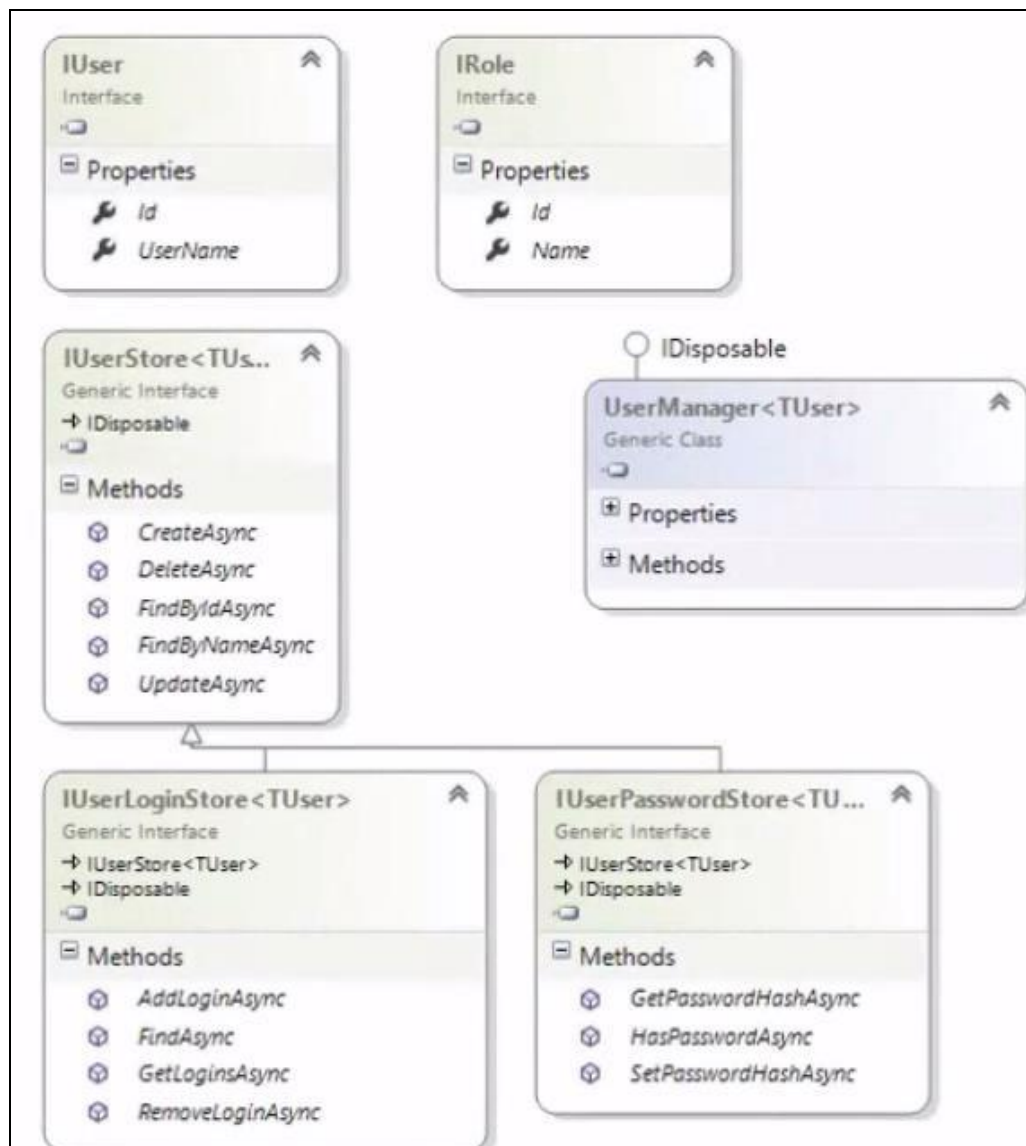
ASP.NET Identity también incluye un esquema de base de datos predefinido que podemos utilizar inmediatamente con la configuración predeterminada de nuestros proyectos. Podemos cambiar la ubicación por defecto en la que nuestra base de datos será generada (Entity Framework Code First) cambiando la cadena de conexión que viene por defecto en el archivo Web.config.

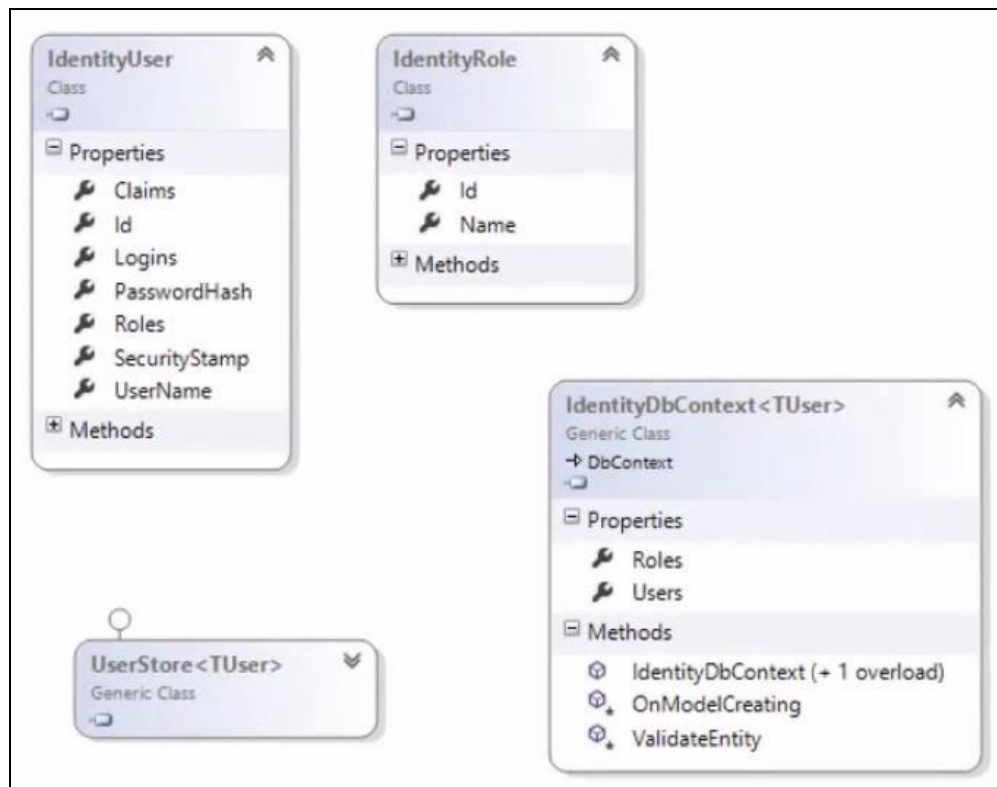
ASP.NET Identity permite realizar la autenticación no solo utilizando el mecanismo de usuario/password sino por medio de proveedores globales de identidad como es Facebook, Twitter, entre otros.

ASP.NET Identity fue desarrollado para brindar soporte a WebForms, ASP.NET MVC, Web API y SignalR.

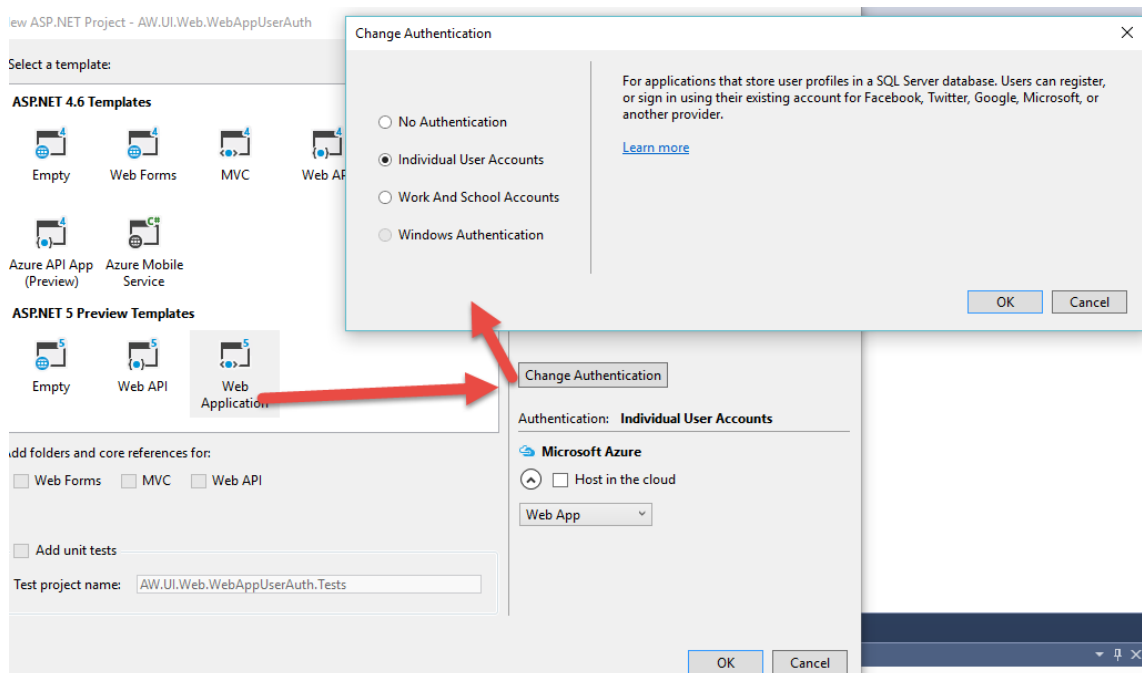
ASP.NET Identity, el cual provee los siguientes namespaces para trabajar con usuarios y roles:

Microsoft.AspNet.Identity.Core



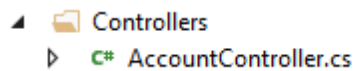
Microsoft.AspNet.Identity.EntityFramework:

Para aplicaciones que se usen en internet, debemos implementar la autenticación por formularios (Individual User Accounts).

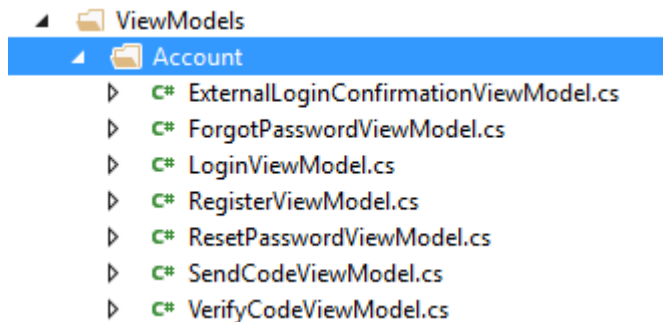


Con lo que se genera lo siguiente:

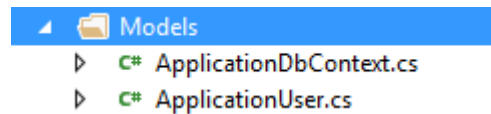
- AccountController: Contiene los action methods Login, Logoff, Register, entre otros.



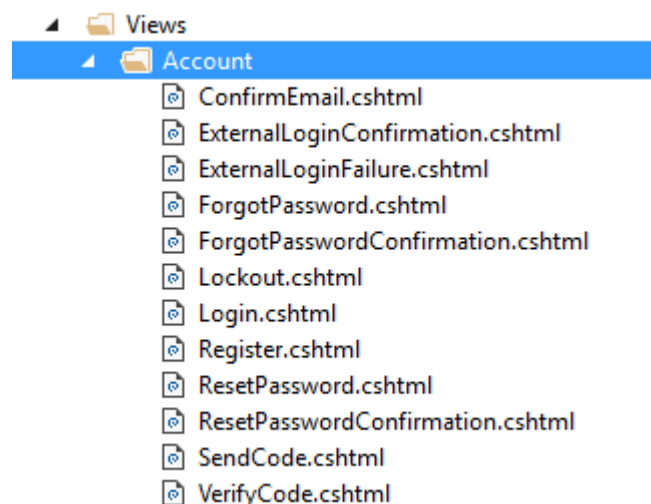
- AccountViewModels: Contiene los ViewModels a ser usados por las vistas generadas.



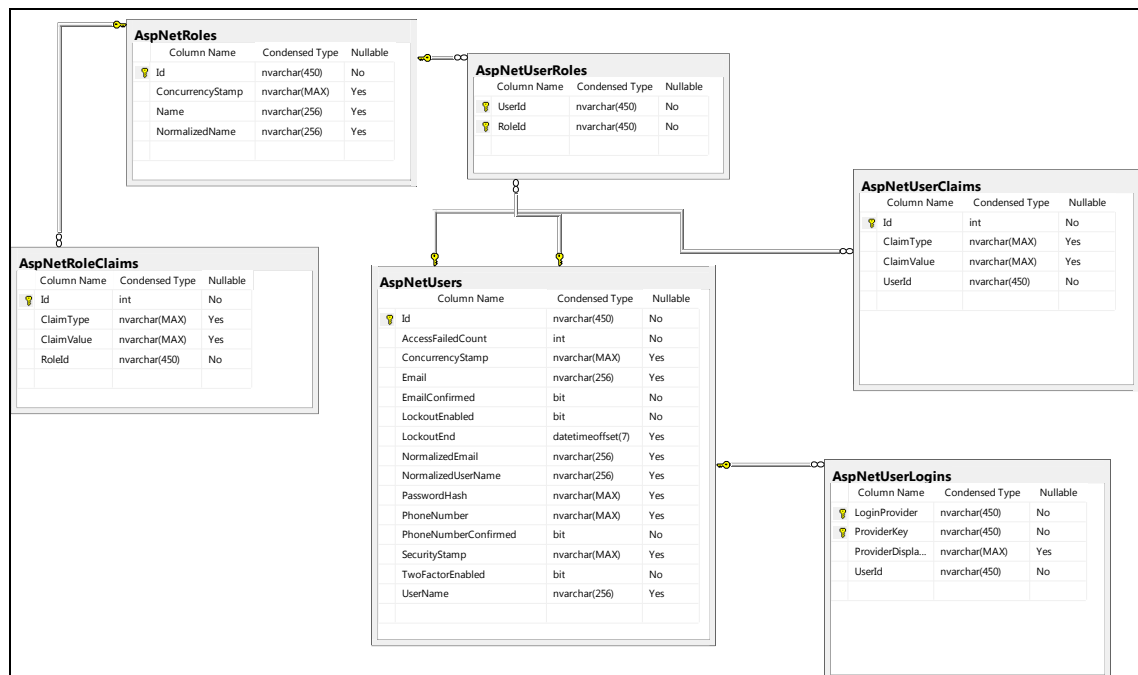
- IdentityModels: Contiene el IdentityDbContext (DbContext de entity framework especializado para ASP.NET Identity), en el cual se define la cadena de conexión que se usa para conectarse a la base de datos.



- Account Views: Contiene las vistas que se generan automáticamente, y ya traen la funcionalidad lista para usar.



La base de datos que se genera tiene las siguientes tablas:



Una vez que tenemos la aplicación con ASP.NET Identity, podemos manejar la autenticación y autorización:

- La autenticación ya viene con la vista Login.
- La autorización la podemos implementar utilizando el atributo `[Authorize]`. Decorando los controllers o action methods con este atributo, permite restringir el acceso a ellos.

Tiene las siguientes propiedades:

- Order: Define el orden en que el ActionFilter será ejecutado.
- Roles: Obtiene o asigna los nombres de los roles requeridos para un método de acción.
- User: Obtiene o asigna los nombres de usuarios requeridos para acceder al método de acción.

Por ejemplo, en la siguiente imagen, el action method "Index" solo puede ser accedido por usuarios con el rol "Administrators":

```
public class HomeController : Controller
{
    [Authorize(Roles="Administrators")]
    0 references
    public ActionResult Index()
    {
    }
```

En caso se desee especificar la autorización para todos los action methods de un controller, la forma de hacerlo es aplicando el atributo a nivel del controller, ya no de los action methods como en el ejemplo anterior.

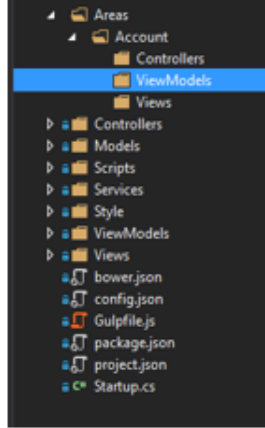
Si hemos aplicado el atributo [Authorize] a nivel de controller, pero se desea que uno o varios action methods no requieran autenticación, entonces se usa el atributo **[AllowAnonymous]**, tal como se muestra en el siguiente ejemplo, en donde, para ejecutar cualquier action method del HomeController, el usuario debe estar autenticado, menos para Index:

```
[Authorize]
0 references
public class HomeController : Controller
{
    [AllowAnonymous]
    0 references
    public ActionResult Index()
    {
```

4. Áreas

Áreas

- ASP.NET MVC le permite dividir las aplicaciones Web en unidades más pequeñas que se conocen como áreas.
- Las Áreas proporcionan una manera de separar una aplicación Web grande MVC en agrupaciones funcionales más pequeñas.
- Un área es una estructura MVC dentro de una aplicación. Una aplicación puede contener varias estructuras MVC (áreas).



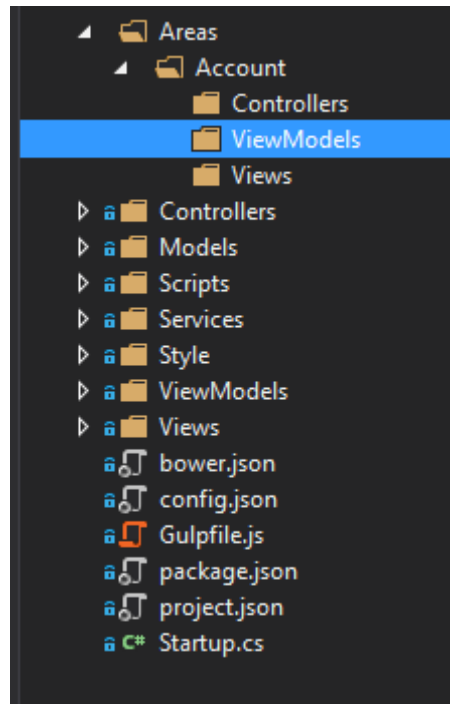
5 - 16 Copyright © Todos los Derechos Reservados - Cibertec Perú S.A.C.

El patrón MVC separa la lógica del modelo (datos) de una aplicación a partir de su lógica de presentación y la lógica de negocio. En ASP.NET MVC, esta separación lógica también se implementa físicamente en la estructura del proyecto, donde los controladores y las vistas se guardan en carpetas que utilizan convenciones de nomenclatura para definir las relaciones. Esta estructura es compatible con las necesidades de la mayoría de las aplicaciones web.

Sin embargo, algunas aplicaciones pueden tener un gran número de controladores, y cada controlador puede ser asociado con varias vistas. Para estos tipos de aplicaciones, la estructura del proyecto ASP.NET MVC predeterminado puede llegar a ser difícil de manejar.

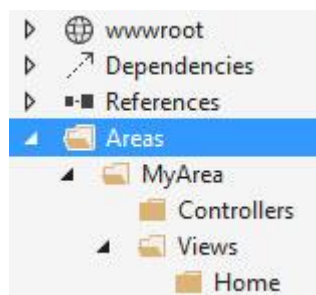
Para dar cabida a los grandes proyectos, ASP.NET MVC le permite dividir las aplicaciones Web en unidades más pequeñas que se conocen como áreas. Las Áreas proporcionan una manera de separar una aplicación Web grande MVC en Agrupaciones funcionales más pequeñas. Un área es una estructura MVC dentro de una aplicación. Una aplicación puede contener varias estructuras MVC (áreas).

En el siguiente ejemplo, se puede ver que se ha creado un área para cuentas de usuario:



Pasos para trabajar con áreas:

1. Crear un fólder en la carpeta raíz, con el nombre **Areas** y dentro de esta las áreas que desea implementar, como la imagen tenemos Areas->MyArea.



2. Los controladores a crear deben agregarse en la carpeta Controllers del área a trabajar. La diferencia con un controlador normal es que ahora estos controladores deben tener el atributo **[Area]**

```
[Area("MyArea")]
public class HomeController : Controller
{
    // GET: /<controller>/
    public IActionResult Index()
    {
        return View();
    }
}
```


3. Como último paso se tiene que configurar la ruta del área en el método **Configure** del archivo **Startup.cs**

```
// Add MVC to the request pipeline.
app.UseMvc(routes =>
{
    // add the new route here.
    routes.MapRoute(name: "areaRoute",
        template: "{area:exists}/{controller}/{action}",
        defaults: new { controller = "Home", action = "Index" });

    routes.MapRoute(
        name: "default",
        template: "{controller}/{action}/{id?}",
        defaults: new { controller = "Home", action = "Index" });
});
```

Ahora la nueva ruta debe trabajar exactamente como las rutas por defecto pero ahora se le tiene que adicionar el nombre del área. Así usted puede crear una vista Index para su controlador HomeController y navegar de la siguiente manera **/MyArea/Home** or **/MyArea/Home/Index**.

4. Cross-Site Scripting (XSS)

Uno de los aspectos que se deben controlar en el desarrollo de una aplicación es la validación de entrada de datos, y el Cross-Site Scripting (XSS) es una técnica utilizada para inyectar código malicioso ejecutable en las aplicaciones, del cual todo desarrollador se debe proteger.

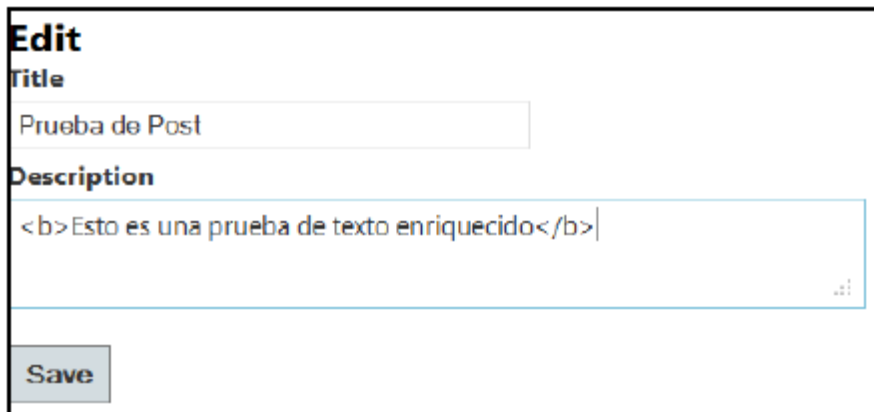
Para el siguiente ejemplo, se ha utilizado la clase Post:

```
public class Post
{
    [Key]
    public int Id { get; set; }

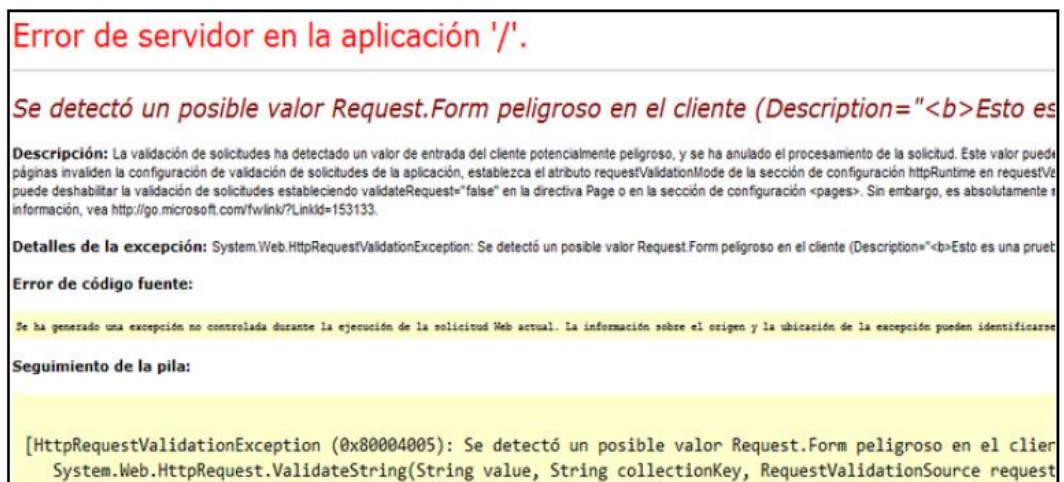
    [Required]
    public string Title { get; set; }

    [DataType(DataType.MultilineText)]
    [AllowHtml]
    public string Description { get; set; }
}
```

Observar qué es lo que sucede al intentar insertar código malicioso en la descripción de un post, en este ejemplo se quiso insertar un texto entre tags HTML del tipo ``:



Se mostrará un mensaje de error, indicando que se ha detectado un posible valor Request.Form peligroso en el cliente:



Es decir, ASP.NET MVC protege por defecto del XSS. Sin embargo, el problema se presenta si se desea que los usuarios puedan introducir código HTML en la aplicación.

Para que los usuarios puedan insertar texto enriquecido, como por ejemplo HTML, se deberá realizar lo siguiente:

- Decorar las propiedades de las clases del modelo con `[AllowHtml]`.
- Utilizar en las vistas el Helper `@Html.Raw(Propiedad)`.

Decorar las propiedades y utilizar el Helper mencionado permite la entrada de toda clase de texto, con lo que se debe prevenir la entrada de código malicioso de forma manual.

Es en este caso en donde se debe usar la librería AntiXSS que provee Microsoft y que se encuentra en NuGet, la cual provee de todos los elementos necesarios para proteger contra XSS.

AntiXSS añade dos referencias al proyecto: `AntiXSSLibrary` y `HtmlSanitizationLibrary`, que se utilizan en los controllers para sanear los

datos de entrada que puedan contener código malicioso. A continuación, se muestra un ejemplo que utiliza el método `GetSafeHtmlFragment` para sanear la propiedad `Description`. Este método elimina todo código malicioso de entrada:

```
public ActionResult Create([Bind(Include="Id,Title,Description")] Post post)
{
    if (ModelState.IsValid)
    {
        db.Posts.Add(post);
        post.Description = Sanitizer.GetSafeHtmlFragment(post.Description);
        db.SaveChanges();
        return RedirectToAction("Index");
    }

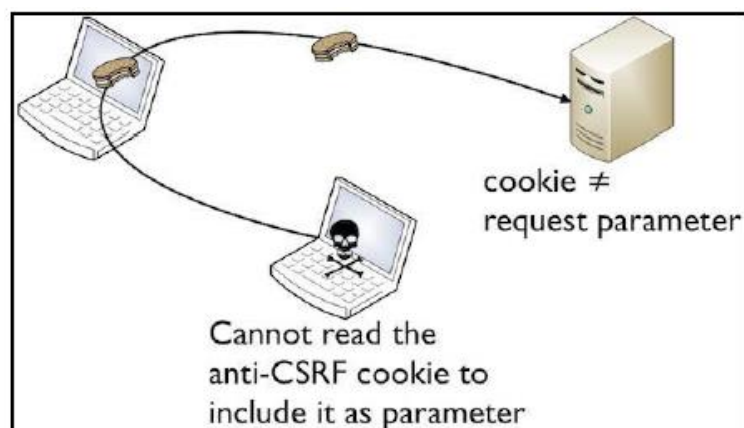
    return View(post);
}
```

5. Cross-Site Request Forgery (CSRF)

El Cross-Site Request Forgery (CSRF o XSRF), al contrario del XSS que explota la confianza del usuario en un sitio en particular, explota la confianza del sitio en un usuario en particular.

El CSRF aprovecha que un usuario está validado en una aplicación, para a través de esta, introducir solicitudes "válidas" que modifiquen el comportamiento de la aplicación a favor del atacante. Es decir, el atacante usa a la víctima para que sea ella misma quien realice la transacción dañina, cuando la víctima se encuentra validada en el servidor y en la aplicación específica.

El proceso es simple, muchos usuarios no finalizan correctamente (o no pueden hacerlo) sus sesiones en las aplicaciones bancarias o de otra índole que puedan ser afectadas por esta vulnerabilidad y las mantienen activas mientras navegan otros sitios, más aún, en tiempos en que las pestañas de los navegadores son muy utilizadas. Por tanto, desde cualquier otra ventana en el navegador, se pudiera inducir al usuario a pulsar un enlace a sitios en los que el usuario ya se ha autenticado, para que él mismo ejecute la acción de ataque sin darse cuenta.



En cuanto a código, se recomienda el control estricto del Timeout de la Session de la aplicación, de forma que si el usuario no se conecta nuevamente y olvida salir correctamente de la aplicación, el servidor deberá en un lapso corto de tiempo dar dicha sesión por finalizada.

Otra eficiente forma de controlar el CSRF es la introducción de un token dinámico adicional en las solicitudes del cliente que se asocia a la sesión de este en el servidor y se agrega a todos los enlaces y solicitudes. Este token será siempre diferente por sesión y por usuario, por lo que de esta forma, se le hace más difícil al atacante, tratar de emular el enlace necesario para efectuar el ataque debido a que el token es variable.

En el caso de ASP.NET MVC 6, se debe agregar el atributo **ValidateAntiForgeryToken** en la acción del controlador a validar CSRF, tal como se muestra en el siguiente ejemplo:

```
public class MyController : Controller
{
    [HttpPost]
    [ValidateAntiForgeryToken]
    public IActionResult Create(SomeModel model)
    {
        ...
    }
}
```

En la vista no se tiene que crear nada, por defecto en el formulario se crea el token de seguridad (forgerytoken).

El resultado del renderizado del html sería el siguiente:

```
<form action="/My/Create/" method="post">

<input name="__RequestVerificationToken" type="hidden"
value="CfDJ8DCDWZ4iOzZDmNKI5HFAUd2qQe4qwOhVP4znwDITDINNk_h-
1a2v0A1aDPCdb4IEgc9X_cTsjKkCDUCYh9EKr7HqXI3hvlRfnRltwfJwblmCZlx38
uDfwVu5jzdVZOcaXUUmopBDtG6-0__0FVezb-U" />
</form>
```

Si se desea desabilitar el token de seguridad, se tiene que indicar una propiedad **asp-antiforgery** en el TagHelper del form.

```
<form asp-action="Create" asp-antiforgery="false">

</form>
```