

Capítulo 5

Creando Aplicaciones ASP.NET MVC 6

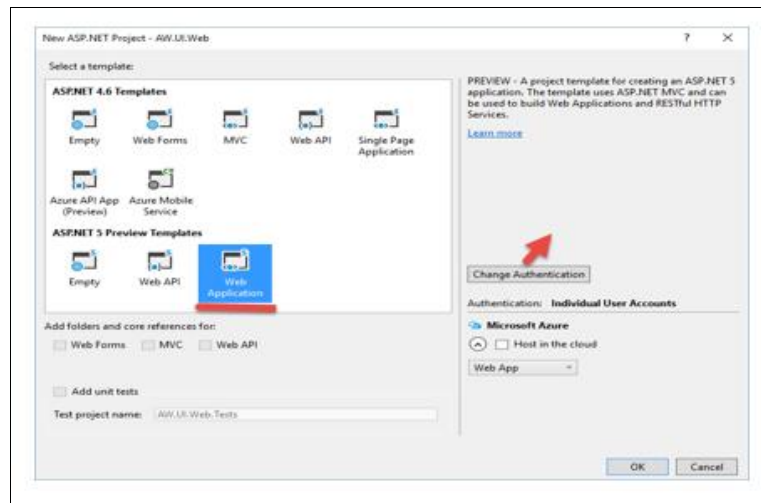
Objetivo

- Al finalizar el capítulo, el alumno: Identifica y aplica correctamente las plantillas que proporciona el Visual Studio 2017 para la creación de aplicaciones ASP.NET MVC 6.
- Implementa Modelos.
- Implementa Controladores.
- Implementa Vistas.

Temas

1. Exploración de las plantillas del Visual Studio 2017.
2. Revisar convenciones de nombres en MVC.
3. Introducción e implementación de Modelos.
4. Personalizar los modelos usando atributos y validaciones.
5. Introducción e implementación de Controladores, revisar los diferentes tipos de Actions.
6. Introducción e implementación de Vistas, verificar los diferentes tipos de Views.

1. Exploración de las plantillas del Visual Studio 2017

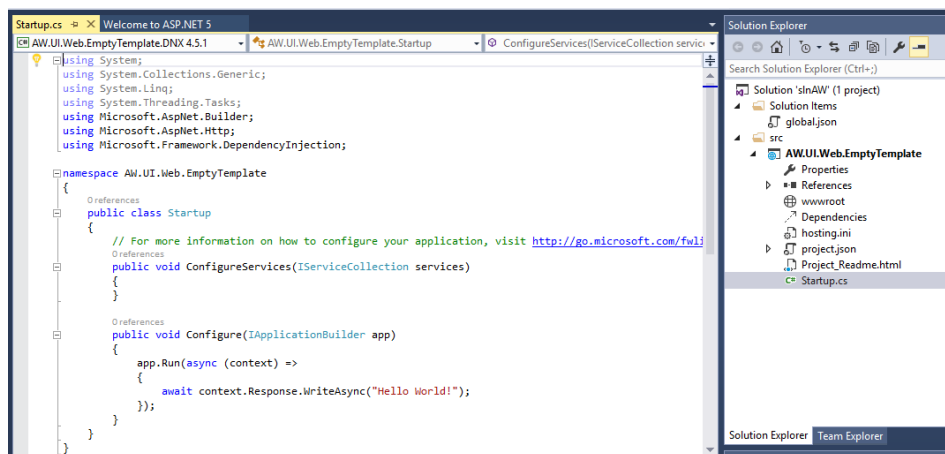
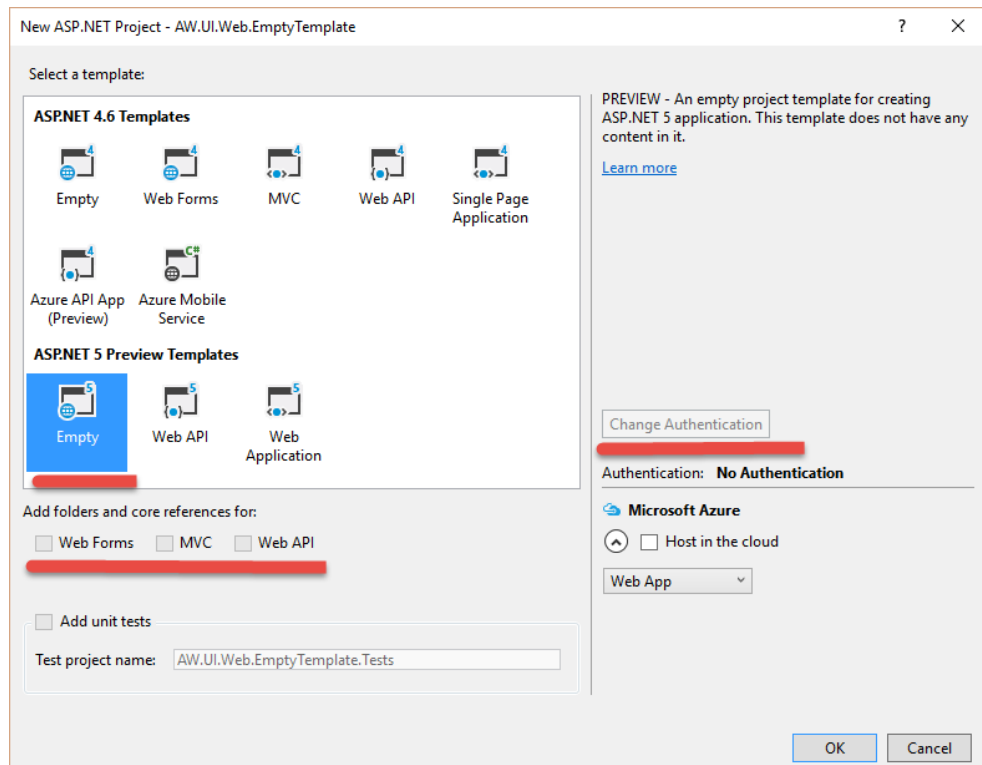


Visual Studio utiliza plantillas para crear proyectos web. Una plantilla de proyecto puede crear archivos y carpetas en el nuevo proyecto, instalar paquetes de NuGet y proporciona código de ejemplo para una aplicación. Plantillas que implementan los últimos estándares web y pretenden demostrar las mejores prácticas de cómo utilizar las tecnologías ASP.NET así como darle un salto de inicio en la creación de su propia aplicación.

Se tiene plantillas para crear aplicaciones web utilizando Asp.Net 4.6 Templates o la versión más reciente como lo es Asp.Net 5.0.

En la versión más reciente de .net tenemos tres plantillas que nos permiten crear aplicaciones web y son las siguientes:

Empty Template: proporciona una plantilla con lo mínimo necesario para crear una aplicación web, teniendo que agregar luego los servicios y componentes requeridos para crear una aplicación web real.



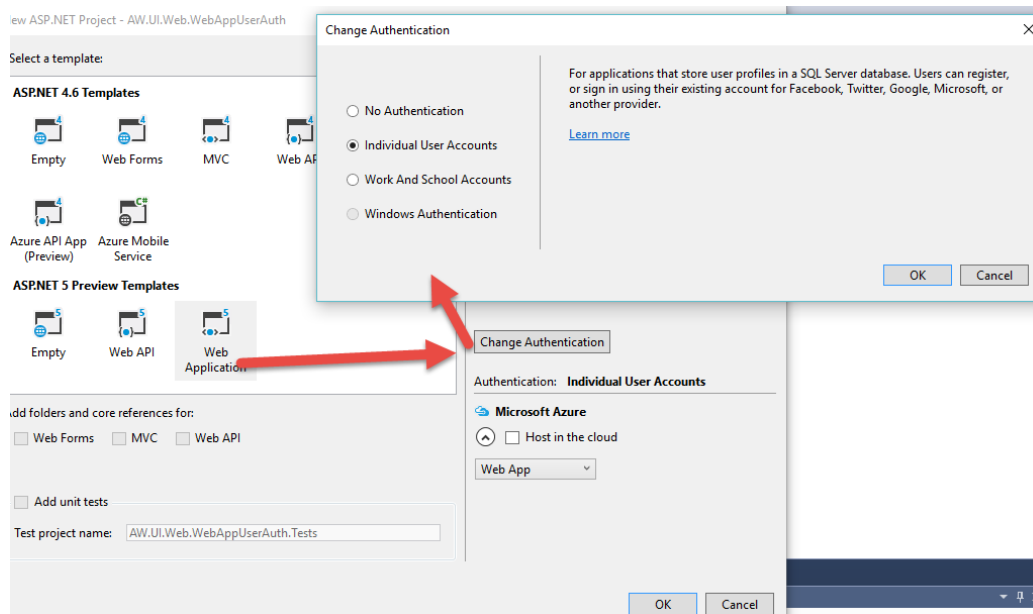
Más adelante se muestra el detalle de cada archivo.

Web Api Template: crear la estructura de carpetas y archivos para implementar servicios web basados en RESTful.

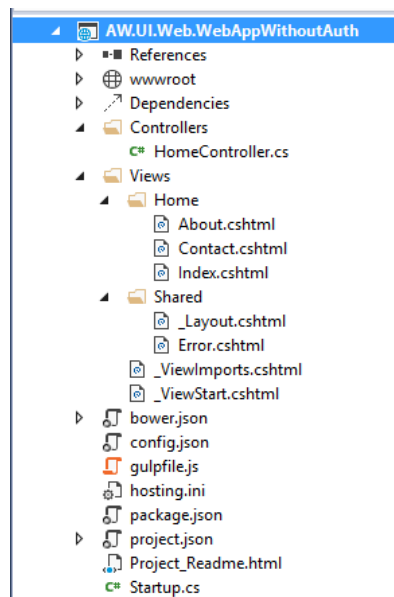


Web Application: crear la estructura de carpetas y archivos para implementar aplicaciones web basadas en MVC. Crea un proyecto de ejemplo y en el cual se hace uso de Bootstrap que es un framework que proporciona los estilos (colores, tipo de letra, distribución de los componentes html, responsive design, etc) para construir aplicaciones web profesionales.

Mediante este template se puede indicar el tipo de autenticación que va a soportar la aplicación. Por defecto se crea con **Individual User Accounts** (Cuentas de usuarios individuales).



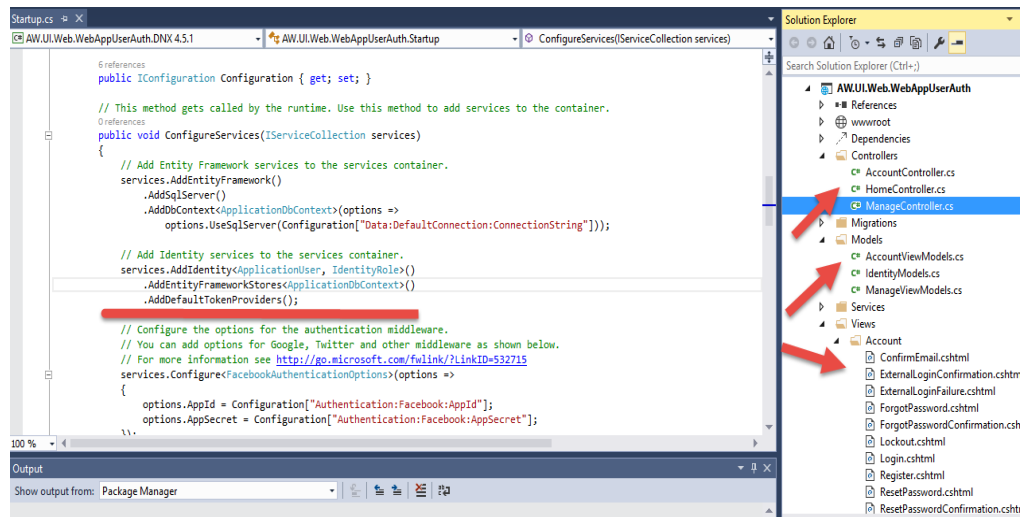
- **No Authentication:** pensando para aplicaciones que no requieren soportar autenticación de usuario, mucho menos autorización.



- **Individual User Accounts:** 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 se puede 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

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:

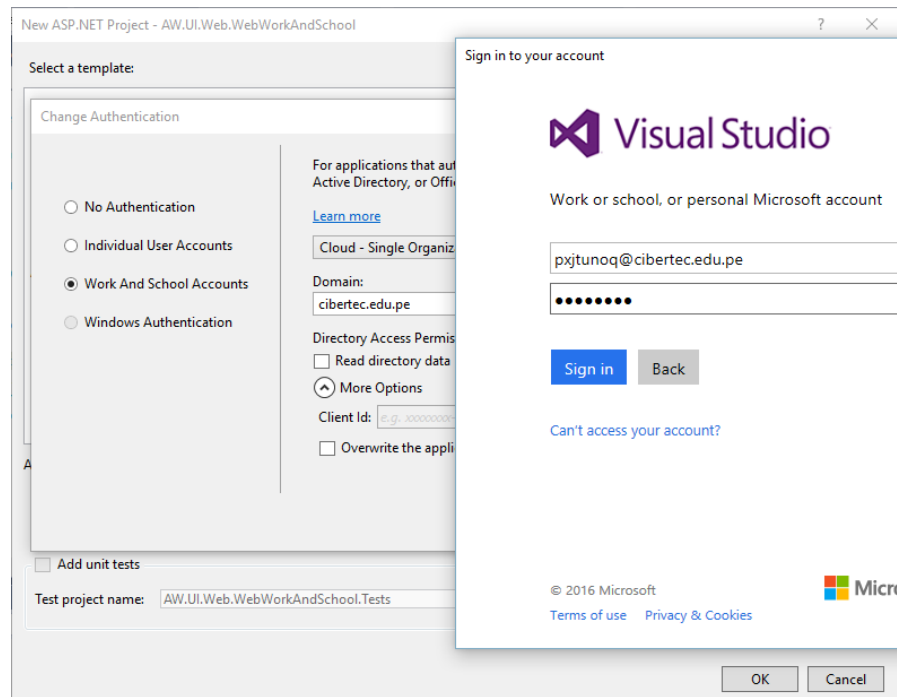
☐ Read directory data ⓘ

☒ More Options

Client Id: e.g., xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx

☐ Overwrite the application entry if one with same id exists

OK Cancel

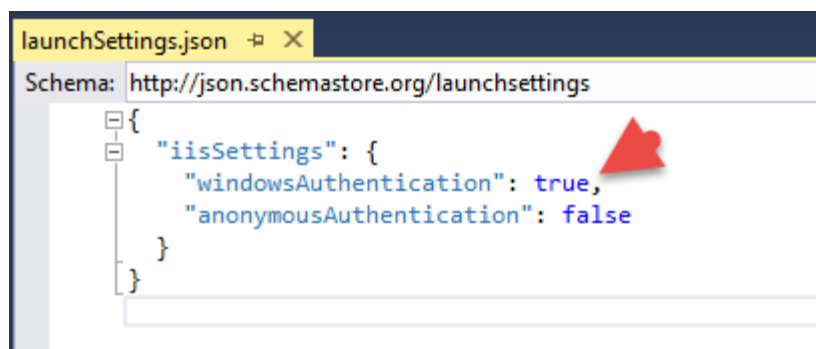


```
// This method gets called by the runtime. Use this method to add services to the container.
// References
public void ConfigureServices(IServiceCollection services)
{
    services.Configure<CookieAuthenticationOptions>(options =>
    {
        (parameter) IServiceCollection services true;
    });

    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.



Archivos Importantes

project.json	En este archivo se configuran las dependencias del Proyecto, información para la compilación y comando a ejecutar.
global.json	Se definen dependencias para todos los proyectos de una solución.
config.json o appsettings.json	En este archivo se define la información de configuración de la aplicación, como por ejemplo la cadena de conexión a la base de datos.
package.json	Este archivo permite gestionar las dependencias del proyecto mediante Node Package Manager (NPM)
bower.json	En este archivo se gestionan los componentes de terceros que contienen html, css, javascript, fonts e images.
gulpfile.js	<p>Archivo que permite automatizar algunas tareas relacionadas a la minificación (Minification) y agrupamiento (Bundles) de archivos javascript y css.</p> <p>Para automatizar las tareas se hace uso de Grunt que es una herramienta open-source.</p>
Startup.cs	<p>Mediante esta clase se agregan los componentes que va usar la aplicación al momento de ejecutarse.</p> <ul style="list-style-type: none">- Se define el archivo de configuración que va usar.- Variables de entorno.- Agrega el componente de MVC. <p>Se define el HTTP Request pipeline (respuesta a solicitud de archivos estáticos, acciones MVC, etc.).</p>

Archivo Startup.cs

Este componente que se ejecuta al iniciar una aplicación web en MVC. Consta de un constructor y 2 métodos:

- Startup: en el constructor de la clase Startup se indica el archivo de configuración que se va a usar en la aplicación.

```
public Startup(IHostingEnvironment env, IApplicationEnvironment appEnv)
{
    // Setup configuration sources.
    var builder = new ConfigurationBuilder(appEnv.ApplicationBasePath)
        .AddJsonFile("config.json")
        .AddEnvironmentVariables();
    Configuration = builder.Build();
}
```

- ConfigureServices: en este método se agregan los servicios que se van a usar en la aplicación web, tales como EntityFramework, Asp.Net Identity, configuración de autenticación, MVC o Inyección de Dependencia (DI).

```
public void ConfigureServices(IServiceCollection services)
{
    // Add Entity Framework services to the services container.
    services.AddEntityFramework()
        .AddSqlServer()
        .AddDbContext<ApplicationDbContext>(options =>
            options.UseSqlServer(Configuration["Data:DefaultConnection:ConnectionString"]));

    // Add Identity services to the services container.
    services.AddIdentity<ApplicationUser, IdentityRole>()
        .AddEntityFrameworkStores<ApplicationDbContext>()
        .AddDefaultTokenProviders();

    // Configure the options for the authentication middleware.
    // You can add options for Google, Twitter and other middleware as shown below.
    // For more information see http://go.microsoft.com/fwlink/?LinkID=532715
    services.Configure<FacebookAuthenticationOptions>(options =>
    {
        options.AppId = Configuration["Authentication:Facebook:AppId"];
        options.AppSecret = Configuration["Authentication:Facebook:AppSecret"];
    });

    services.Configure<MicrosoftAccountAuthenticationOptions>(options =>
    {
        options.ClientId = Configuration["Authentication:MicrosoftAccount:ClientId"];
        options.ClientSecret = Configuration["Authentication:MicrosoftAccount:ClientSecret"];
    });

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

- Configure: si bien es cierto en el método **ConfigureServices** se indica que servicios va a usar la aplicación, en el método Configure se indica o define el orden de ejecución de los servicios (se define el HTTP Request pipeline)

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env, ILoggerFactory loggerFactory)
{
    loggerFactory.MinimumLevel = LogLevel.Information;
    loggerFactory.AddConsole();

    // Configure the HTTP request pipeline.

    // Add the following to the request pipeline only in development environment.
    if (env.IsDevelopment())
    {
        app.UseBrowserLink();
        app.UseErrorPage(ErrorPageOptions.ShowAll);
    }
    else
    {
        // Add Error handling middleware which catches all application specific errors and
        // send the request to the following path or controller action.
        app.UseExceptionHandler("/Home/Error");
    }

    // Add static files to the request pipeline.
    app.UseStaticFiles();

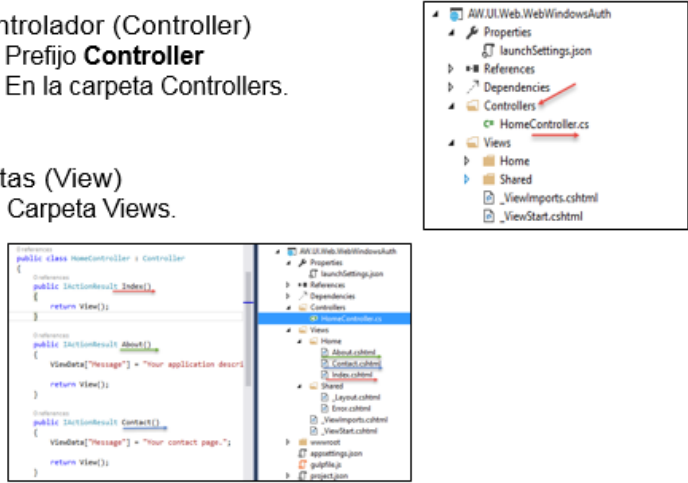
    // Add MVC to the request pipeline.
    app.UseMvc(routes =>
    {
        routes.MapRoute(
            name: "default",
            template: "{controller=Home}/{action=Index}/{id?}");

        // Uncomment the following line to add a route for porting Web API 2 controllers.
        // routes.MapWebApiRoute("DefaultApi", "api/{controller}/{id?}");
    });
}
```

2. Revisar convenciones de nombres en MVC

Revisar convenciones de nombres en MVC

- Controlador (Controller)
 - Prefijo **Controller**
 - En la carpeta Controllers.
- Vistas (View)
 - Carpeta Views.

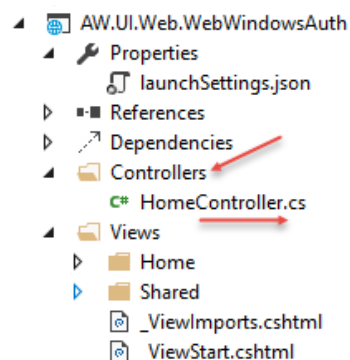


The screenshot shows a Visual Studio project named 'AW.UI.Web.WebWindowsAuth'. The 'Controllers' folder contains 'HomeController.cs'. The 'Views' folder contains 'Home' and 'Shared' subfolders, with files like '_ViewImports.cshtml' and '_ViewStart.cshtml'. The code editor shows the 'HomeController' class with methods 'Index', 'About', and 'Contact', each returning a view. The 'Index' method returns 'View()', 'About' returns 'View()' with a message, and 'Contact' returns 'View()' with a message.

En MVC se tiene que tener en cuenta ciertas convenciones de nombres referidos principalmente al controlador y vistas.

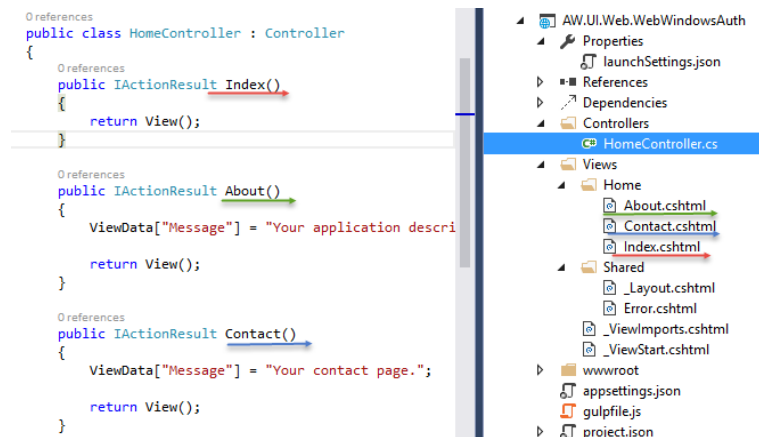
Controlador

- Todos los controladores se ubican en la carpeta Controllers debajo de la raíz del proyecto.
- Todos los controladores deben tener la palabra **Controller** después del nombre del controlador. Ejemplo si se desea implementar el controlador para gestionar la información de productos, entonces el controlador tendría que tener el nombre ProductoController.



Vistas

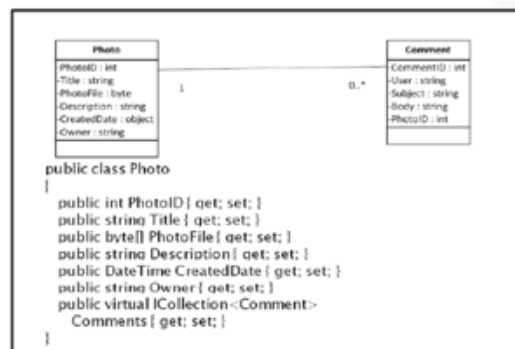
- Todas las vistas se ubican en la carpeta Views debajo de la raíz del proyecto.
- Las sub-carpetas que se crean en la carpeta Views deben estar directamente relacionadas a los controladores. Por ejemplo, si se tiene el controlador HomeController, este debe crear una sub-carpeta llamada Home.
- Dentro de las sub-carpetas se encuentran las vistas, que por default están directamente relacionadas a cada una de las acciones de un controlador.



3. Introducción e implementación de Modelos (Model)

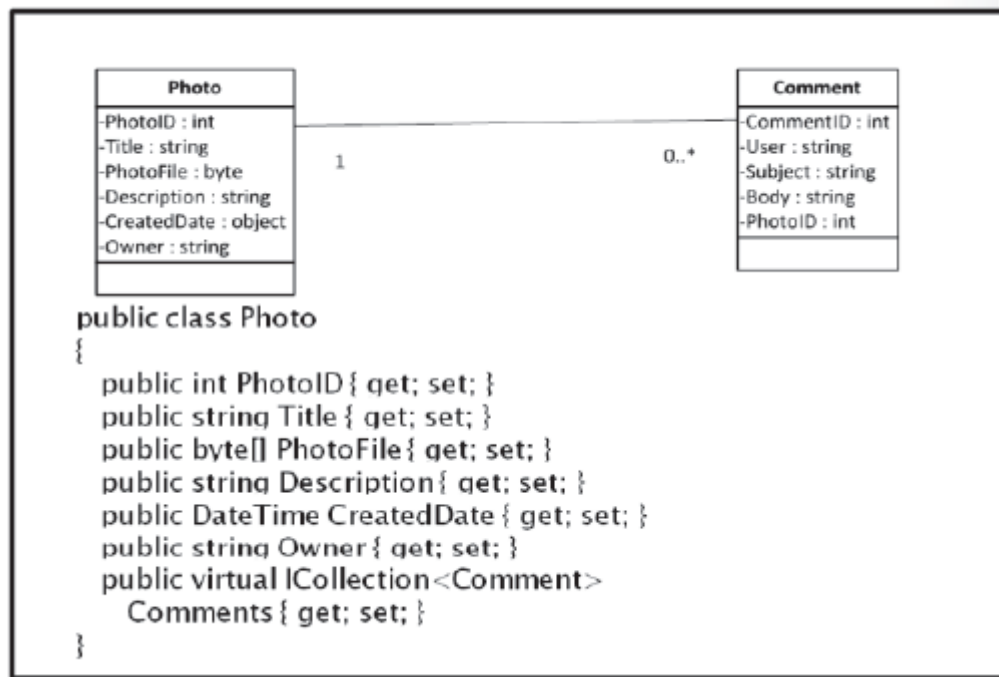
Introducción e implementación de Modelos

- Los modelos son un conjunto de clases (entidades) que representan el dominio de la aplicación. Un modelo debe representar lo siguiente: Lógica de negocio, validaciones y acceso a datos.



Los modelos es un conjunto de clases (entidades) que representan el dominio de la aplicación. Un modelo debe representar lo siguiente: Lógica de negocio, validaciones y acceso a datos.

Los modelos se pueden colocar en la carpeta Models o pueden estar ubicados en tres proyectos de librería de clases. Uno que represente las entidades, el segundo que represente la capa de acceso a datos y el último la capa de negocio.



4. Personalizar los modelos usando atributos y validaciones.

Personalizar los modelos usando atributos y validaciones

- Se debe hacer uso del namespace `System.ComponentModel.DataAnnotations`.

```
public class Movie {  
    public int ID { get; set; }  
  
    [Required]  
    public string Title { get; set; }  
  
    [DataType(DataType.Date)]  
    public DateTime ReleaseDate { get; set; }  
  
    [Required]  
    public string Genre { get; set; }  
  
    [Range(1, 100)]  
    [DataType(DataType.Currency)]  
    public decimal Price { get; set; }  
  
    [StringLength(5)]  
    public string Rating { get; set; }  
}
```

A los modelos se puede agregar atributos que van a permitir validar la información de la entidad.

Se debe hacer uso del siguiente namespace en el modelo:

```
using System.ComponentModel.DataAnnotations;
```

Un atributo de validación se debe agregar por cada propiedad de la clase.

```
public class Movie {
    public int ID { get; set; }

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

    [DataType(DataType.Date)]
    public DateTime ReleaseDate { get; set; }

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

    [Range(1, 100)]
    [DataType(DataType.Currency)]
    public decimal Price { get; set; }

    [StringLength(5)]
    public string Rating { get; set; }
}
```

Lista de atributos importantes a configurar en el modelo:

Atributo	Descripción
Display	Es el texto o etiqueta que aparece en la a lado de control. Ejemplo: [Display(Name = "Email")]
DataType	El tipo de dato se utiliza para renderizar el control adecuado en la vista. Ejemplo control de tipo número, teléfono, password, etc. [DataType(DataType.Password)] [DataType(DataType.DateTime)]
Required	Es la etiqueta que indica que el valor de campo es obligatorio. [Required]
EmailAddress	Valida un dirección de email. [EmailAddress]
StringLength	Indica el tamaño máximo y mínimo de caracteres en un campo. [StringLength(100, ErrorMessage = "The {0} must be at least {2} characters long.", MinimumLength = 6)]
Compare	Permite comparar dos cadenas de datos. [Compare("Password", ErrorMessage = "The password and confirmation password do not match.")] El primer parámetro de la anotación Password representa la propiedad de la clase con la cual se desea comparar.
Range	Permite indica el mínimo y máximo valor para un tipo de dato numérico. [Range(0, 99999)]
RegularExpression	Permite configurar una expresión regular para una propiedad de la clase. [RegularExpression(@"^[a-zA-Z"]-\s){1,40}\$", ErrorMessage = "Numbers and special characters are not allowed in the name.")]

5. Introducción e implementación de Controladores (Controller)

Introducción e implementación de Controladores

- El controlador entonces responde a las acciones del usuario, carga la información de un modelo y lo pasa a la vista que finalmente es renderizada como una página web.
- Se hereda de una clase base Controller se encuentra en el namespace Microsoft.AspNet.Mvc

```
5 references
public class HomeController : Controller
{
    4 references
    public IActionResult Index()
    {
        return View();
    }

    0 references
    public IActionResult About()
    {
        ViewData["Message"] = "Your application description page.";
        return View();
    }
}
```

En las aplicaciones web el usuario interactúa con la aplicación haciendo click en botones o enlaces (links). El controlador entonces responde a las acciones del usuario, carga la información de un modelo y lo pasa a la vista que finalmente es renderizada como una página web.

Los controladores se ubican en la carpeta Controllers.

Todos los controladores heredan de una clase base **Controller** que se encuentra en el siguiente namespace:

Microsoft.AspNet.Mvc

```
5 references
public class HomeController : Controller
{
    4 references
    public IActionResult Index()
    {
        return View();
    }

    0 references
    public IActionResult About()
    {
        ViewData["Message"] = "Your application description page.";

        return View();
    }
}
```

Los controladores representa una clase, esta clase tiene métodos y en MVC se le conoce con el nombre Acciones (Actions).

De esa manera, se crea un extremo (endpoint web) que puede ser consultado a través de un cliente que pueda realizar solicitudes HTTP (un navegador, un robot, un dispositivo). Por ejemplo según la imagen anterior en HomeController existen las siguientes 2 rutas que responden solicitudes, acorde a la siguiente convención:

<http://localhost:<puerto>/<CONTROLADOR>/<ACCION>>

```
http://localhost:<puerto>/Home/Index
http://localhost:<puerto>/Home/About
```

(*) nota: La palabra “controller” en la clase es removida del nombre.

Tipos de Acciones en controladores

El tipo por defecto y genérico de las acciones es el IActionResult, pero se pueden usar otros tipos específicos como:

Tipo de acción	Descripción
PartialViewResult	Son usadas para mostrar partes de la aplicación web, como por ejemplo mostrar el nombre del usuario que ingreso al sistema.
RedirectResult	Se usa esta acción para redireccionar a una url específica.
RedirectToAction	Se usa para redireccionar a una acción específica.
JsonResult	Se usa para devolver información al browser en formato JSON.

6. Introducción e implementación de Vistas (Views)

Introducción e implementación de Vistas

- La vista es la respuesta que da el controlador en forma de HTML y que contiene los datos del modelo.
- Están ubicadas en el folder Views.
- ¿Qué extensiones tienen? .cshtml o .vbhtml dependiendo del lenguaje de programación, si es C# la extensión es cshtml, si es Visual Basic la extensión es vbhtml.
- ¿Qué lenguaje se utiliza? HTML a través del pre compilador de HTML llamado Razor.

La vista es la respuesta que da el controlador en forma de HTML y que contiene los datos del modelo.

¿Dónde están ubicadas? En el folder **Views**.

¿Qué extensiones tienen? .cshtml o .vbhtml dependiendo del lenguaje de programación, si es C# la extensión es cshtml, si es Visual Basic la extensión es vbhtml.

¿Qué lenguaje se utiliza? HTML a través del pre compilador de HTML llamado Razor. Toda aplicación web tiene ciertos componentes que se repiten entre páginas, tipo “marco”. En Web Forms se le conocía como MasterPage, en MVC se le conoce como **_Layout**. Usualmente estas secciones incluyen partes como encabezados, pie de página y menús de navegación. Por ejemplo, en la imagen de la derecha lo que está en celeste sería el layout.

layout.cshtml

Este archivo incluye todo el marco y plantilla HTML, es decir, tags como <head>, Al abrir este archivo se puede apreciar algo nuevo en esta versión de ASP.NET, el tag “environment” el cual permite, según el despliegue que se realice, agregar unos u otros archivos de scripts y hojas de estilos. Por ejemplo, para el momento de desarrollo sirven los archivos fuentes iniciales, sin embargo, en producción usualmente los archivos que se envían a un usuario son los archivos de hojas de estilo y scripts minificados y obfuscados.

Esto para evitar el uso innecesario de ancho de banda y complicar la lectura del código fuente que se descarga del lado cliente y que durante desarrollo se pueda realizar depuración de código JavaScript.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"] - WebApplication2</title>

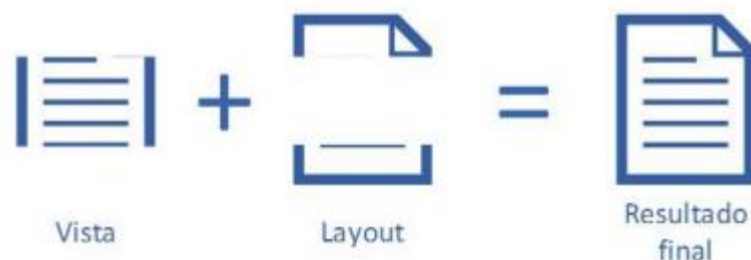
    <environment>...</environment>
    <environment names="Staging,Production">
      <link rel="stylesheet" href="https://ajax.aspnetcdn.com/ajax/bootstrap/
        asp-fallback-href="~/lib/bootstrap/dist/css/bootstrap.min.css"
        asp-fallback-test-class="sr-only" asp-fallback-test-property="
      <link rel="stylesheet" href="~/css/site.min.css" asp-append-version=
    </environment>
    @Html.ApplicationInsightsJavaScript(TelemetryConfiguration)
  </head>
  <body>
    <div class="navbar navbar-inverse navbar-fixed-top">...</div>
    <div class="container body-content">
      @RenderBody()
      <hr />
      <footer>
        <p>&copy; 2016 - WebApplication2</p>
      </footer>
    </div>

    <environment>...</environment>
    <environment>...</environment>

    @RenderSection("scripts", required: false)
  </body>
</html>
```

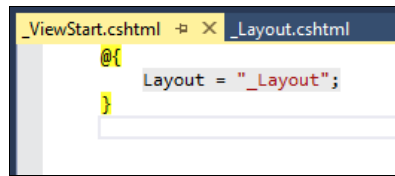
En esta vista se puede encontrar la sentencia `@RenderBody()` en esta parte del HTML se renderizará el resto de páginas que utilicen este archivo de layout. La manera de asignar este layout a diferentes páginas es a través de una variable llamada Layout.

Es decir si en alguna vista se asigna algún valor a esa variable global, esa página utilizará ese archivo de layout. Con el siguiente diagrama se intentará explicar un poco más claro esto.



ViewStart.cshtml

Mediante esta vista se puede indicar cuál es el Layout por defecto en la aplicación.



La siguiente sentencia permite incrustar código C#, se conocen como bloques de código (code blocks):

```
@{  
}
```

Permiten ejecutar lógica, por ejemplo:



@RenderSection("scripts", required: false): Permite renderizar scripts que las vista secundarias puedan usar.