**OBJETIVO:**

Crear una aplicación sencilla de registro en el sitio que incluya más de una página y valide las entradas.

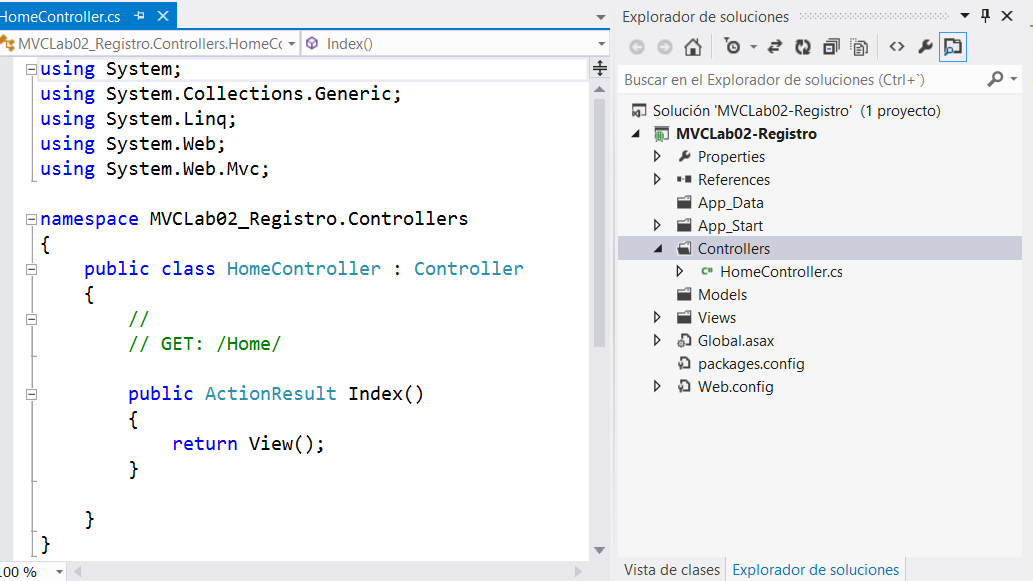
La primera página solicitará el alias, mail y contraseña.  
La segunda página será una página de confirmación e incluirá datos ingresados en la primera página.

**CREACIÓN DEL PROYECTO APLICACIÓN AS.NET MVC 4**

1. Crear una Aplicación de ASP.NET MVC 4 y denominarla MVCLab02-Registro.
2. Seleccionar la opción sin autenticación de usuario como se hizo en el LAB001
3. Desmarcar la opción para Tests Unitarios

**AGREGAR EL CONTROLADOR PARA INDEX**

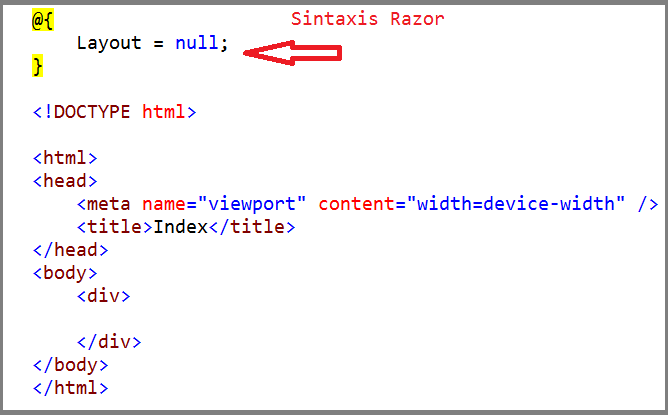
1. Agregar un controlador llamado ***HomeController*** en la carpeta ***Controllers.*** Desde el explorador de soluciones, dar clic derecho sobre la carpeta ***Controllers*** y seleccionar ***Agregar > Controlador.*** Elegir la plantilla de controlador vacío. Ponerle como nombre ***HomeController.*** En el cuadro de diálogo dar clic en aceptar para sobreescribir el controlador HomeController existente. Se genera una clase ***HomeController*** que deriva de ***Controller*** e incluye por defecto un método públic  ***ActionResult Index***.



**AGREGAR LA VISTA INDEX**

1. En el código, sobre el ***ActionResult Index()*** hacer clic derecho y seleccionar ***Agregar Vista***.
2. En el cuadro de diálogo, mantener el nombre de la vista ***Index*** y la plantilla por defecto y desmarcar ***Usar una página de diseño o maestra***.
3. Observar:

* Debajo de la carpeta ***Views > Home*** se creó un archivo de código ***Index.cshtml***. Por convención tiene el nombre de la vista, y la extensión de Razor: cshtml.
* En la carpeta ***Views*** se incluye un archivo ***Web.config***.
* Se generó la vista incluyendo el código HTML y Razor:



Por el momento dejaremos pendiente la vista. Si se visualiza en el navegador (F5) veremos una página vacía cuyo título es "Index".

Eliminaremos las otras vistas que se encuentran en la carpeta Views (About y Contact)

**CREAR EL MODELO**

1. Sobre la carpeta ***Models***, clic derecho, agregar Clase. Ponerle como nombre ***Usuario***.

Incluirle tres properties de tipo String:

* + Nombre
  + Email
  + Password

**CREAR UN FORMULARIO HTML UTILIZANDO HTMLHELPERS Y RAZOR.**

Abrir la vista Index.cshtml e incluir el siguiente código dentro de body.

El uso del “***HtmlHelper”*** ***@Html.BeginForm()*** da inicio al formulario y se cierra automáticamente al final del ámbito ( “{“ y “}”). Como puede verse, cada elemento o input se genera a través de un textbox y tiene un elemento de validación que se explicará más adelante.

|  |
| --- |
| @model MVCLab02\_Registro.Models.Usuario    @Html.ValidationSummary()  @using (Html.BeginForm())  {  <p>  @Html.LabelFor(m => m.Nombre)  @Html.TextBoxFor(m => m.Nombre)  @Html.ValidationMessageFor(m => m.Nombre)  </p>  <p>  @Html.LabelFor(m => m.Email)  @Html.TextBoxFor(m => m.Email)  @Html.ValidationMessageFor(m => m.Email)  </p>  <p>  <label for="Password">Password:</label>  @Html.PasswordFor(m=>m.Password )  @Html.ValidationMessageFor(m=>m.Password)  </p>      <p>  <input type="submit" value="Registrarse" />  </p>  } |

Si se visualiza en el navegador, veremos las casillas de texto correspondientes y el botón para el envío del form.

**ASOCIAR EL MODELO A LA VISTA.**

Para referenciar nuestra clase desde la vista utilizaremos la etiqueta Razor ***@model***:

@model MVCLab02\_Registro .Models.Usuario

Esta etiqueta, como se ve en el código anterior, debe quedar por encima de la primera sentencia del formulario correspondiente al modelo a validar, en este caso encima de @Html.ValidationSummary("Se produjo un error. Puede proseguir luego de corregirlos") .[[1]](#footnote-1)

**IMPLEMENTAR ACCIONES EN RESPUESTA A LOS MÉTODOS HTTP QUE SE UTILICEN: POST y/o GET.**

Estas validaciones se realizarán del lado del servidor, por lo que será necesario diferenciar los dos casos posibles:

* 1. Cuando el formulario se despliega por primera vez, al estar vacíos los datos, se deberá responder a una solicitud HTTP *GET.*
  2. Cuando se reenvía el formulario con datos, se responderá a una petición *POST*.

Para proveer código según el tipo de petición HTTP, se deben escribir dos métodos ActionResult: uno decorado con [HttpGet] que no reciba el modelo como parámetro para desplegar el formulario por primera vez, y otro decorado con [HttpPost] que sí reciba el modelo, para trabajar con él (por ejemplo, para tomar las reglas de validación incluidas en los Data Annotations)

Para usar el modelo, no olvide incluir el using:

using MVCLab02\_Registro.Models;

Nuestro controlador deberá quedar así:

|  |
| --- |
| public class HomeController : Controller  {  //  // GET: /Home/  [HttpGet]  public ActionResult Index()  {  return View();  }  // POST: /Home/  [HttpPost]  public ActionResult Index(Usuario u)  {  return View(u)  } |

Colocar un punto de parada en cada método ActionResult, ejecutarlo y verificar que la primera vez se detiene en el Get y y al continuar despliega el formulario, hacer clic sin llenar los datos, y verificar que al enviar el formulario nuevamente se detiene en el ActionResult decorado con [HttpPost].

Inspeccionar la instancia del modelo (u) y verificar que sus atributos están en null

Probar ahora rellenando los datos correctamente, e inspeccionar el objeto en el ActionResult correspondiente al post. El objeto en este caso tiene los datos ingresados.

**REFINAR LAS VALIDACIONES UTILIZANDO DATA ANNOTATIONS EN EL MODELO.**

Las anotaciones las realizaremos sobre la clase Usuario.

Recordar que para utilizarlas es necesario incluir una sentencia using:

using System.ComponentModel ;

using System.ComponentModel.DataAnnotations;

En este ejemplo utilizaremos algunas de ellas, se puede consultar la lista completa en la url:

https://msdn.microsoft.com/en-us/library/system.componentmodel.dataannotations%28v=vs.110%29.aspx

Modifique la clase Usuario para que concuerde con el siguiente código. Posteriormente probarla dejando los campos vacíos, ingresando datos no válidos, y finalmente ingresando datos corectos.

|  |
| --- |
| using System.ComponentModel;  using System.ComponentModel.DataAnnotations;  namespace MVCLab02\_Registro.Models  {  public class Usuario  {  /\* A cada atributo se le agregará la anotación [Required] a la que opcionalmente  \* se le puede agregar un parámetro con nombre ErrorMessage para especificar el  \* mensaje de error en caso de no ingresar nada.  \*/  [Required(ErrorMessage = "El nombre no se puede dejar vacío")]  //Cambiar el nombre que se desplegará en el formulario  [DisplayName("Nombre de usuario")]  //Indicar largo mínimo  [StringLength(10)]  public string Nombre { get; set; }  //Indicar que se trata de una dirección de mail para que realice automáticamente la validación  [Required]  [EmailAddress]  public string Email { get; set; }  //Varias validaciones  [Required(ErrorMessage = "La contraseña no se puede dejar vacía")]  [DataType(DataType.Password)]  [MinLength(8, ErrorMessage = "Largo mínimo de la contraseña: 8"),  MaxLength(12, ErrorMessage = "Largo máximo de la contraseña:12")]  public string Password { get; set; }  }  } |

**DEFINIR ACCIONES DIFERENTES SEGÚN LA VALIDEZ DEL MODELO**

Es posible verificar la validez del modelo mediante la propiedad *isValid* de ModelState. De esa forma, podremos realizar distintas acciones según su valor (true/false). Incluir el siguiente código y probarlo:

|  |
| --- |
| public class HomeController : Controller  {  //  // GET: /Home/  [HttpGet]  public ActionResult Index()  {  return View();  }  // POST: /Home/  [HttpPost]  public ActionResult Index(Usuario u)  {  if (ModelState.IsValid)  return (RedirectToAction("RegistroOk"));  else  return View(u);  }  public ActionResult RegistroOk()  {  ViewBag.Mensaje = "Gracias por registrarse";  return View();  }  } |

Para probarlo, agregar una View para la acción *RegistroOk* e incluirle un mensaje:

|  |
| --- |
| @{  Layout = null;  }  <!DOCTYPE html>  <html>  <head>  <meta name="viewport" content="width=device-width" />  <title>RegistroOk</title>  </head>  <body>  <div>  @ViewBag.Mensaje  </div>  </body>  </html> |

Ejecutarlo, y verificar que al ingresar valores no válidos se retorna a Index, y cuando todos los valores son válidos se despliega la vista RegistroOk.

RedirectToAction se comporta análogamente a Response.Redirect, salvo que despliega la vista indicada como parámetro, que previamente debe haber sido creada. No es el único comportamiento posible, se sugiere buscar información en el MSDN o similar.

**iNCLUIR DATOS AL REDIRECCIONAR A OTRA ACCIÓN**

Si deseamos compartir datos entre los controladores podemos pasar un parámetro al método RedirectToAction y declararlo en el método ActionResult correspondiente.

Sustituir el método RegistroOk como sigue:

public ActionResult RegistroOk(Usuario usr)

{

@ViewBag.Mensaje = "Gracias por registrarse " + usr.Nombre + " su email es: " + usr.Email ;

return View();

}

Modificar la invocación desde el método Index (HttpPost) de modo que le envíe el usuario válido a RegistroOk:

[HttpPost]

public ActionResult Index(Usuario u)

{

if (ModelState.IsValid)

return (RedirectToAction("RegistroOk", u));

else

return View(u);

}

Probar ingresando todos los datos correctos, y verificar que en RegistroOk se accede a los valores ingresados en Index.

1. Para usar otro modelo en la misma vista, se debe declarar nuevamente @model , con lo que se pierde la referencia al anterior. Secuencialmente, se puede trabajar con tantos modelos como se quiera..   
    [↑](#footnote-ref-1)