Realizado por Antonio Arcediano Consuegra
José Andrés Contioso Chaves

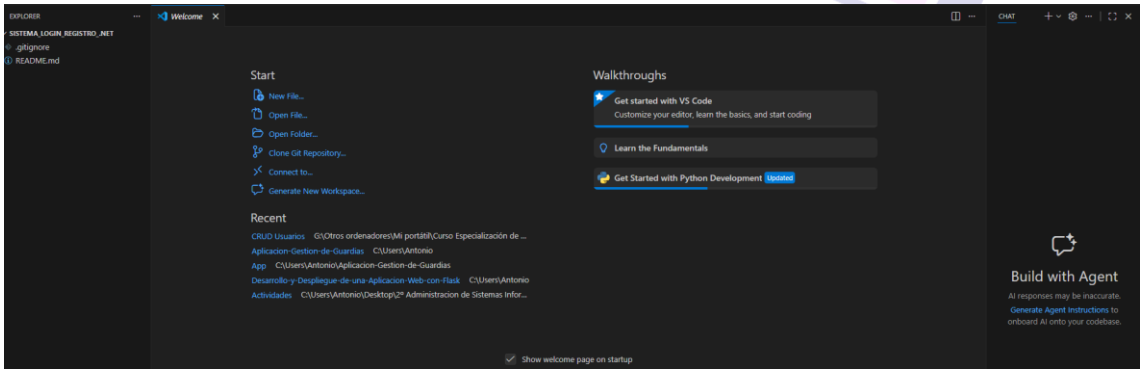
PASO A PASO SISTEMA DE LOGIN Y REGISTRO .NET

Ciberseguridad en Entornos de las Tecnologías de la Información
I.E.S. Martínez Montañés (Sevilla)
Modulo: Puesta en Producción Segura
Profesor: José Luis Fernandez Yagües

INDICE:

1. Creación del Proyecto ASP.NET Core MVC

Lo primero de todo que se va a realizar es abrir la carpeta en la se va almacenar el proyecto; en nuestro caso como va a ser almacenado en un repositorio; abriremos la carpeta que hemos creado para el repositorio.



Una vez que hemos abierto la carpeta vamos a proceder a crear el proyecto MVC ejecutando el siguiente comando: **"dotnet new mvc -n LoginRegistro";** en el cual este crea una aplicación web con el Patrón MVC que significa (Modelo de Datos, Páginas y Lógica y Rutas) y LoginRegistro es el nombre del proyecto.

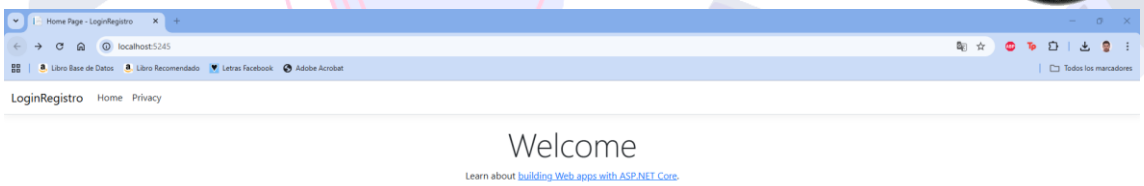
```
PS C:\Users\Antonio\Desktop\Sistema_Login_Registro_.NET> dotnet new mvc -n LoginRegistro
La plantilla "Aplicación web de ASP.NET Core (Modelo-Vista-Controlador)" se creó correctamente.
Esta plantilla contiene tecnologías de terceros ajenos a Microsoft, consulte https://aka.ms/aspnetcore/10.0-third-party-notices para obtener más información.

Procesando acciones posteriores a la creación...
Restaurando C:\Users\Antonio\Desktop\Sistema_Login_Registro_.NET\LoginRegistro\LoginRegistro.csproj:
Restauración realizada correctamente.
```

Tras realizar la creación del proyecto vamos a proceder a entrar en la carpeta de este para ello emplearemos el siguiente comando **"cd LoginRegistro"** y ejecutaremos la aplicación por primera vez para comprobar que funciona todo correctamente con el comando **"dotnet run".**

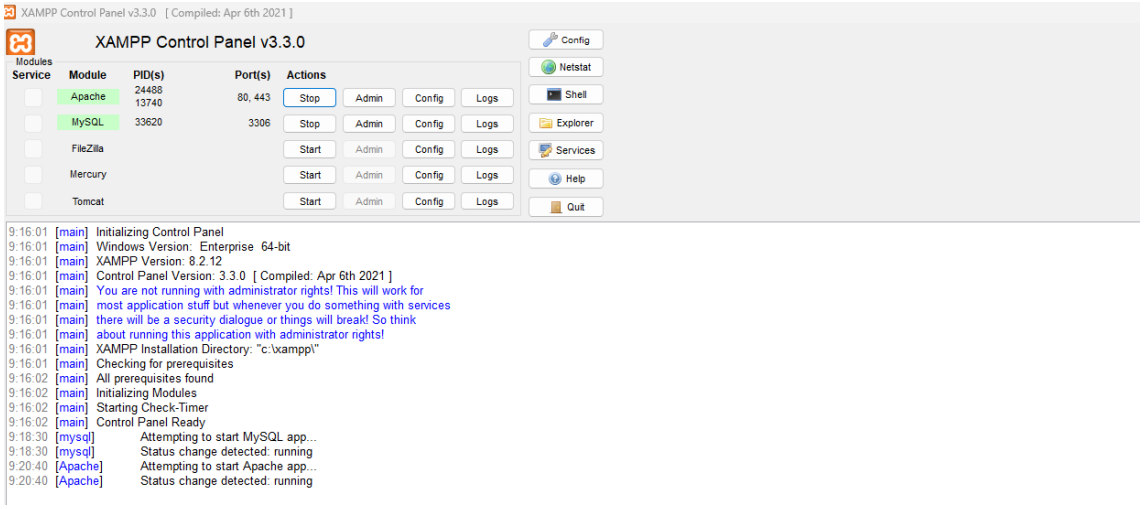
```
PS C:\Users\Antonio\Desktop\Sistema_Login_Registro_.NET> cd .\LoginRegistro\
PS C:\Users\Antonio\Desktop\Sistema_Login_Registro_.NET\LoginRegistro> dotnet run
Usando la configuración de inicio de C:\Users\Antonio\Desktop\Sistema_Login_Registro_.NET\LoginRegistro\Properties\launchSettings.json...
Compilando...
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5245
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: C:\Users\Antonio\Desktop\Sistema_Login_Registro_.NET\LoginRegistro
```

Si analizamos detenidamente el resultado de la ejecución del comando podemos ver como se nos presenta un enlace que pone http://localhost:5245 y nos lleva a la siguiente página web que es la página de bienvenida de ASP.NET

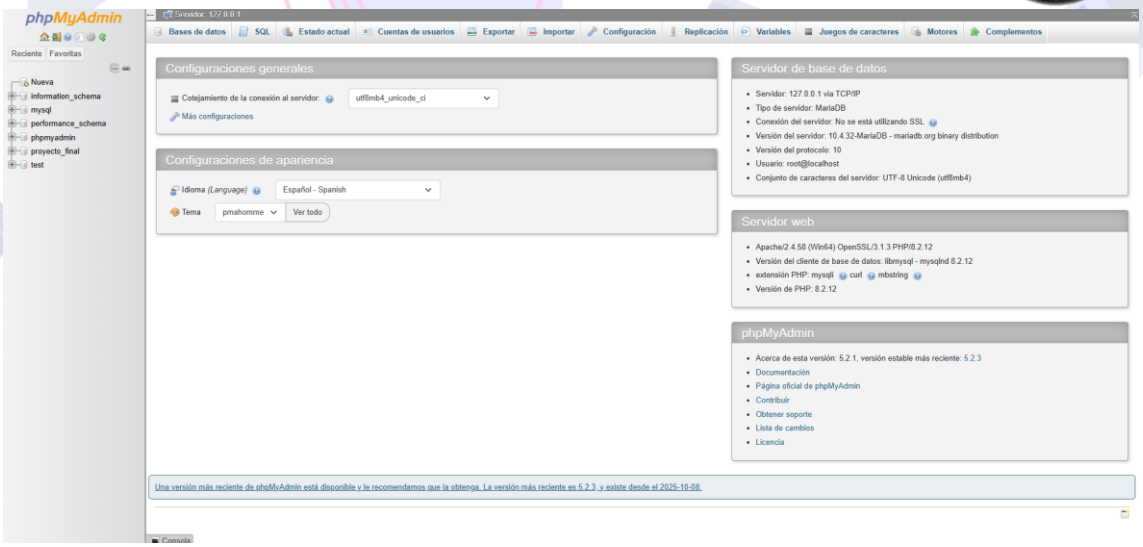


2. Añadir la Base de Datos.

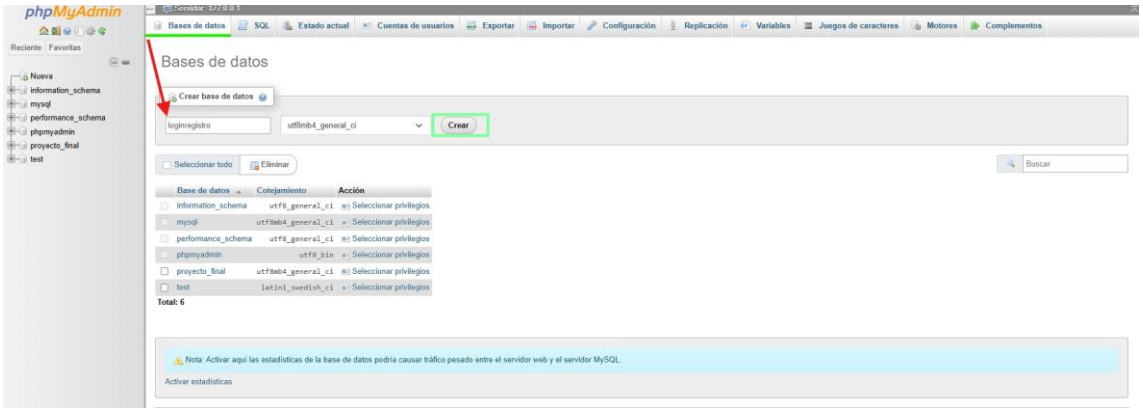
Para esta practica vamos a usar la Base de Datos que viene instalada en XAMPP para ello lo primero que debemos de realizar es abrir XAMPP y dale a Start a los servicios “Apche” y “MySQL”.



Una vez que ya están corriendo los servicios vamos a entrar en phpMyAdmin para ello indicaremos la siguiente URL en el navegador: “http://localhost/phpmyadmin” y como se puede comprobar en la siguiente imagen ya nos encontramos dentro del panel de Administración de phpMyAdmin.



Cuando ya nos encontremos dentro del panel de administración vamos a crear la base de datos en phpMyAdmin para ello nos tenemos que ir a la opción “Base de Datos” indicamos el nombre de la base de datos que en este caso es “loginregistro” y le damos a “Crear”



Una vez que hemos creado la base de datos vamos a proceder a instalar los paquetes necesarios en el proyecto .NET para ello vamos a ejecutar los siguientes comandos dentro de la carpeta del proyecto “dotnet add package Pomelo.EntityFrameworkCore.MySql”; este primer paquete es el conector que permite a Entity Framework Core trabajar con MySQL y el segundo paquete que vamos a instalar es “dotnet add package Microsoft.EntityFrameworkCore.Tools” este segundo paquete es el que nos da las herramientas para usar comandos de Entity Framework en la terminal.

```
PS C:\Users\Antonio\Desktop\Sistema_Login_Registro_.NET\LoginRegistro> dotnet add package Pomelo.EntityFrameworkCore.MySql
info : La validación de la cadena de certificados X.509 utilizará el almacén de confianza predeterminado seleccionado por .NET para la firma de código.
info : La validación de la cadena de certificados X.509 utilizará el almacén de confianza predeterminado seleccionado por .NET para la marca de tiempo.
info : Agregando PackageReference para el paquete "Pomelo.EntityFrameworkCore.MySql" al proyecto "C:\Users\Antonio\Desktop\Sistema_Login_Registro_.NET\LoginRegistro\LoginRegistro.csproj".
info :   GET https://api.nuget.org/v3/registration5-gz-semver2/pomelo.entityframeworkcore.mysql/index.json
info :   OK https://api.nuget.org/v3/registration5-gz-semver2/pomelo.entityframeworkcore.mysql/index.json 515 ms
info :   GET https://api.nuget.org/v3/registration5-gz-semver2/pomelo.entityframeworkcore.mysql/page/1.0.0-prerelease-20160722/1.1.1-prerelease-10021.json
info :   OK https://api.nuget.org/v3/registration5-gz-semver2/pomelo.entityframeworkcore.mysql/page/1.0.0-prerelease-20160722/1.1.1-prerelease-10021.json 152 ms
info :   GET https://api.nuget.org/v3/registration5-gz-semver2/pomelo.entityframeworkcore.mysql/page/1.1.1-prerelease-10022/3.1.0-rc1.final.json
info :   OK https://api.nuget.org/v3/registration5-gz-semver2/pomelo.entityframeworkcore.mysql/page/1.1.1-prerelease-10022/3.1.0-rc1.final.json 151 ms
info :   GET https://api.nuget.org/v3/registration5-gz-semver2/pomelo.entityframeworkcore.mysql/page/3.1.0/9.0.0.json
info :   OK https://api.nuget.org/v3/registration5-gz-semver2/pomelo.entityframeworkcore.mysql/page/3.1.0/9.0.0.json 471 ms
info : Restaurando paquetes para C:\Users\Antonio\Desktop\Sistema_Login_Registro_.NET\LoginRegistro\LoginRegistro.csproj...
info :   GET https://api.nuget.org/v3-flatcontainer/pomelo.entityframeworkcore.mysql/index.json
info :   OK https://api.nuget.org/v3-flatcontainer/pomelo.entityframeworkcore.mysql/index.json 211 ms
info :   GET https://api.nuget.org/v3-flatcontainer/pomelo.entityframeworkcore.mysql/9.0.0/pomelo.entityframeworkcore.mysql.9.0.0.nupkg
info :   OK https://api.nuget.org/v3-flatcontainer/pomelo.entityframeworkcore.mysql/9.0.0/pomelo.entityframeworkcore.mysql.9.0.0.nupkg 37 ms
info :   GET https://api.nuget.org/v3-flatcontainer/microsoft.entityframeworkcore.relational/index.json
info :   GET https://api.nuget.org/v3-flatcontainer/mysqlconnector/index.json
```

```
PS C:\Users\Antonio\Desktop\Sistema_Login_Registro_.NET\LoginRegistro> dotnet add package Microsoft.EntityFrameworkCore.Tools
info : La validación de la cadena de certificados X.509 utilizará el almacén de confianza predeterminado seleccionado por .NET para la firma de código.
info : La validación de la cadena de certificados X.509 utilizará el almacén de confianza predeterminado seleccionado por .NET para la marca de tiempo.
info : Agregando PackageReference para el paquete "Microsoft.EntityFrameworkCore.Tools" al proyecto "C:\Users\Antonio\Desktop\Sistema_Login_Registro_.NET\LoginRegistro\LoginRegistro.csproj"
info :   GET https://api.nuget.org/v3/registration5-gz-semver2/microsoft.entityframeworkcore.tools/index.json
info :   OK https://api.nuget.org/v3/registration5-gz-semver2/microsoft.entityframeworkcore.tools/index.json 350 ms
info :   GET https://api.nuget.org/v3/registration5-gz-semver2/microsoft.entityframeworkcore.tools/page/1.0.0-msbuild1-final/3.1.0.json
info :   OK https://api.nuget.org/v3/registration5-gz-semver2/microsoft.entityframeworkcore.tools/page/1.0.0-msbuild1-final/3.1.0.json 160 ms
info :   GET https://api.nuget.org/v3/registration5-gz-semver2/microsoft.entityframeworkcore.tools/page/3.1.1/6.0.0-preview.4.21253.1.json
info :   OK https://api.nuget.org/v3/registration5-gz-semver2/microsoft.entityframeworkcore.tools/page/3.1.1/6.0.0-preview.4.21253.1.json 152 ms
info :   GET https://api.nuget.org/v3/registration5-gz-semver2/microsoft.entityframeworkcore.tools/page/6.0.0-preview.5.21301.9/7.0.15.json
info :   OK https://api.nuget.org/v3/registration5-gz-semver2/microsoft.entityframeworkcore.tools/page/6.0.0-preview.5.21301.9/7.0.15.json 160 ms
info :   GET https://api.nuget.org/v3/registration5-gz-semver2/microsoft.entityframeworkcore.tools/page/7.0.16/10.0.0-preview.5.25277.114.json
info :   OK https://api.nuget.org/v3/registration5-gz-semver2/microsoft.entityframeworkcore.tools/page/7.0.16/10.0.0-preview.5.25277.114.json 154 ms
info :   GET https://api.nuget.org/v3/registration5-gz-semver2/microsoft.entityframeworkcore.tools/page/10.0.0-preview.6.25358.103/10.0.2.json
info :   OK https://api.nuget.org/v3/registration5-gz-semver2/microsoft.entityframeworkcore.tools/page/10.0.0-preview.6.25358.103/10.0.2.json 149 ms
info : Restaurando paquetes para C:\Users\Antonio\Desktop\Sistema_Login_Registro_.NET\LoginRegistro\LoginRegistro.csproj...
info :   GET https://api.nuget.org/v3-flatcontainer/microsoft.entityframeworkcore.tools/index.json
info :   OK https://api.nuget.org/v3-flatcontainer/microsoft.entityframeworkcore.tools/index.json 216 ms
info :   GET https://api.nuget.org/v3-flatcontainer/microsoft.entityframeworkcore.tools/10.0.2/microsoft.entityframeworkcore.tools.10.0.2.nupkg
```

Tras realizar la instalación de los paquetes necesarios vamos a proceder a ir a la carpeta Models que es la que va a contener las clases que representan los datos de la aplicación y crearemos el archivo AppUser.cs. y escribiremos el siguiente código.

LoginRegistro > Models > AppUser.cs

```cs
//Importamos el espacio de nombre en el cual contiene los atributos de validación.
using System.ComponentModel.DataAnnotations;

//Esta línea sirve para poder organizar el código dentro del poryecto LoginRegistro; evitando conflictos y poder mantener el orden del proyecto
namespace LoginRegistro.Models;

//Creamos las AppUser la cual va a presentar a un usuario dentro del sistema. Con esta clase hacemos que Entity Framework se convierta en una tabla My
public class AppUser
{
    //El ID será la clave primaria del usuario; en la base de datos este valor será autoincremental
    public int Id { get; set; }

    //El username (usuario) será un campo obligatorio y tendrá un máximo de 50 caracteres.
    [Required, StringLength(50)]
    public string Username { get; set; } = string.Empty;

    //El PasswordHash almacenara los hash de las contraseñas cifrado; es obligatorio que siempre debe de existir un hash y no se permitan contraseñas
    [Required]
    public string PasswordHash { get; set; } = string.Empty;

    //El DateTime almacenara la fecha de la creación de los usuarios; este dato se pondrá de manera automatica para crear el usuario
    public DateTime CreatedAtUtc { get; set; } = DateTime.UtcNow;
}
```

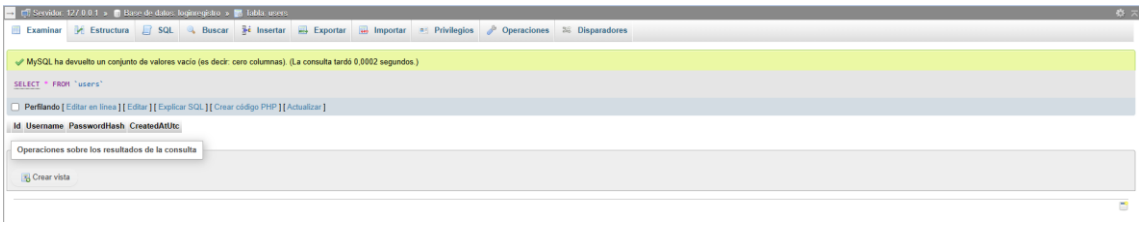
Una vez que hemos generado el código mencionado anteriormente vamos a ir a la carpeta Data y creaermos el archivo AppDbContext.cs; esta clase lo que hace es conectar el proyecto con la base de datos y le dice a Entity Framework que tablas existen y cómo deben ser, y escribiremos el siguiente código:

```
LoginRegistro > Data > AppDbContext.cs
2  using LoginRegistro.Models;
3  //Importamos Entity Framework Core para así poder usar la conexión con la base de datos, los set y el modelo constructor.
4  using Microsoft.EntityFrameworkCore;
5
6  //Esta linea sirve para poder organizar el código dentro del poryecto LoginRegistro; evitando conflictos y poder mantener el orden del proyecto
7  namespace LoginRegistro.Data;
8
9  //Clase Principal de Entity Framework en la cual se encarga de conectar la app con la base de datos
10 public class AppDbContext : DbContext
11 {
12     //Creamos el constructor de la base de datos en el cual recibe la configuración necesaria para la conexión y se le pasa al constructor padre
13     public AppDbContext(DbContextOptions<AppDbContext> options) : base(options) { }
14
15     //Podemos acceder a la tabla lllamada Users con datos de tipo AppUser
16     public DbSet<AppUser> Users => Set<AppUser>();
17
18     //Con este metodo vamos a ajustar las configuraciónes extras de la base de datos.
19     protected override void OnModelCreating(ModelBuilder modelBuilder)
20     {
21         //Dejamos la configuración que viene por defecto de Entity Framework
22         base.OnModelCreating(modelBuilder);
23
24         //En esta línea indicamos que el campo Username debe ser único haciendo que no se puedan registrar dos usuarios con el mismo nombre
25         modelBuilder.Entity<AppUser>()
26             .HasIndex(u => u.Username)
27             .IsUnique();
28     }
29 }
```

Tras finalizar el archivo mencionado anteriormente vamos a proceder a configurar la cadena de conexión que se encuentra en appsettings.json y añadiremos la configuración para la base de datos tal y como podemos visualizar a continuación

```
PS C:\Users\Antonio\Desktop\Sistema Login Registro .NET\LoginRegistro> dotnet tool run dotnet-ef migrations add InitialCreate
Build started...
Build succeeded.
Done. To undo this action, use 'ef migrations remove'
PS C:\Users\Antonio\Desktop\Sistema_Login_Registro_.NET\LoginRegistro> dotnet tool run dotnet-ef database update
Build started...
Build succeeded.
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (3ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      SELECT 1 FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_SCHEMA='loginregistro' AND TABLE_NAME='__EFMigrationsHistory';
```

Una vez que se han ejecutado los comandos que se pueden visualizar en la imagen si nos vamos al panel de administración de la base de datos podemos comprobar que se ha creado las tablas correctamente.



3. Crear Registro (Formulario + Validación + Almacenar en MySQL)

Lo primero que vamos a realizar es crear el modelo del formulario para ello dentro de la carpeta Models; vamos a crear el archivo RegisterVm.cs; este será un modelo del formulario en el cual no guarda datos en la base de datos y sirve para recoger los datos del usuario cuando se rellena el formulario.

```cs
//Importamos DataAnnotations; esto sirve para validar los campos que indiquemos
using System.ComponentModel.DataAnnotations;

//Esta linea sirve para poder organizar el código dentro del poryecto LoginRegistro; evitando conflictos y poder mantener el orden del proyecto
namespace LoginRegistro.Models;

//Clase en la cual representa los datos del formulario de registro; esto solo sirve para el formulario
public class RegisterVm
{
    //La variable Username es obligatoria y tiene un mínimo 3 caracteres y un máximo de 50 caracteres.
    [Required, StringLength(50, MinimumLength = 3)]
    public string Username { get; set; } = string.Empty;

    //La variable Password es obligatoria y tiene un mínimo 8 caracteres y un máximo de 100 caracteres; para así poner contraseñas más seguras
    [Required, StringLength(100, MinimumLength = 8)]
    public string Password { get; set; } = string.Empty;

    //La variable ConfirmPassword es obligatoria y tiene un mínimo 8 caracteres y un máximo de 100 caracteres; para así poner contraseñas más seguras
    //Esta variable se pone para validar la contraseña si la contraseña de arriba con esta es igual son validas
    [Required, Compare(nameof(Password))]
    public string ConfirmPassword { get; set; } = string.Empty;
}
```

Lo segundo que vamos a realizar es crear el controlador AccountController; este controlador es el que se encarga de gestionar las cuentas de usuario en la web para ello se mostrará el formulario de registro, mostrar el login, etc

```cs
LoginRegistro > Controllers > AccountController.cs
1   //Importamos el DbContext que es la conexión a la base de datos.
2   using LoginRegistro.Data;
3   //Importamos los modelos del proyecto que son AppUser y RegisterVm
4   using LoginRegistro.Models;
5   //Importamos lo que es necesario para usar Controllers en ASP.NET MVC
6   using Microsoft.AspNetCore.Mvc;
7   //Importamos EntityFrameworkCore en el cual lo usaremos para los métodos AnyAsync o SaveChangesAsync
8   using Microsoft.EntityFrameworkCore;
9   //Importamos la librerías de criptografía para realizar el hash de contraseñas
10  using System.Security.Cryptography;
11
```

```cs
//Esta linea sirve para poder organizar el código dentro del poryecto LoginRegistro; evitando conflictos y poder mantener el orden del proyecto
namespace LoginRegistro.Controllers;

//Clase Controller que sera la encargada de las acciones que se realicen con las cuentas
public class AccountController : Controller
{
    //Variable privada en la cual almacena el acceso a la base de datos.
    private readonly AppDbContext _db;
    //Constructor que se ejecuta al crear el controlador; este recibe el AppDbContext para poder usar la base de datos
    public AccountController(AppDbContext db)
    {
        _db = db;
    }

    //Este método se muestra cuando indicamos /Account/Register y con el metodo Get lo que estamos haciendo es mostrar el formulario del registro
    [HttpGet]
    public IActionResult Register()
    {
        //Devuelve la vista Register.cshtml
        return View();
    }
```

```
//Este método se ejecuta cuando se pulsa el botón del formulario y con el metodo Post; lo que realizamos es procesar el formulario de registro
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Register(RegisterVm vm)
{
    //Primero validamos de manera automatica el formulario; se comprueba si el modelo cumple con los requisitos que hemos indicado.
    if (!ModelState.IsValid)
        return View(vm);

    //Segundo se comprueba si ya existe el usuario en la base de datos
    var exists = await _db.Users.AnyAsync(u => u.Username == vm.Username);
    //Si existe muestra un mensaje indicando "Ese nombre de usuario ya existe"
    if (exists)
    {
        ModelState.AddModelError(nameof(vm.Username), "Ese nombre de usuario ya existe.");
        return View(vm);
    }

    //Tercero creamos el usuario nuevo con los datos que se nos ha proporcionado en el formulario; en la cual hasheamos la contraseña en la
    //base de datos, etc
    var user = new AppUser
    {
        Username = vm.Username,
        PasswordHash = HashPassword(vm.Password),
        CreatedAtUtc = DateTime.UtcNow
    };

    //Guardamos el usuario en la base de datos y guardamos los cambios
    _db.Users.Add(user);
    await _db.SaveChangesAsync();

    //Mostramos un mensaje indicando que el registro se ha completado y ya se puede iniciar sesión
    TempData["Msg"] = "Registro completado. Ya puedes iniciar sesión.";
    //Una vez que el usuario ya se encuentra creado redirigimos al usuario a la página de Login
    return RedirectToAction("Login");
}
```

```
    //Método privado el cual se usa para crear un hash seguro de la contraseña, haciendo que no se almacene en texto plano.
    private static string HashPassword(string password)
    {
        //Generamos una cadena aleatoria que se añade a la contraseña; haciendo que si dos usuarios tienen la misma contraseña, el hash sea diferente
        byte[] salt = RandomNumberGenerator.GetBytes(16);

        //Generamos el hash usando PBKDF2, este es un método seguro y recomendado
        var pbkdf2 = new Rfc2898DeriveBytes(password, salt, 100000, HashAlgorithmName.SHA256);
        byte[] hash = pbkdf2.GetBytes(32);

        //Guardamos la contraseña hasheada en formato salt.hash
        return $"{Convert.ToBase64String(salt)}.{Convert.ToBase64String(hash)}";
    }
}
```

Lo siguiente que vamos a realizar es crear la vista del formulario de registro para ello vamos a ir a la carpeta Views y dentro de ella accederemos a Auth en el cual crearemos el archivo Register.cshtml.

```
@model LoginRegistro.Models.RegisterVm
//Indicamos que esta vista va a usar el modelo RegisterVm; el formulario trabaja con el nombre de usuario, la contraseña y la confirmación de la contra

@{
    ViewData["Title"] = "Registro";
    //Este título se usa normalmente para mostrar el nombre de la página
}

<h2>Registro</h2>
<!-- Titulo visible que se va a ver en la página -->

<form asp-action="Register" method="post">
    <!-- Formulario que enviará los datos al método Register. -->
    @Html.AntiForgeryToken()
    <!-- Se genera un token de seguridad para evitar ataques CSRF -->
    <div asp-validation-summary="All" style="color:red;"></div>
    <!-- Mostramos los errores generales de validacion en caso de la existencia de alguno -->
```

```
    <div>
        <label asp-for="Username"></label>
        <!-- Se genera una etiqueta label que se encuentra vinculada al campo Username del modelo -->
        <input asp-for="Username" />
        <!-- Se crea el input del formulario que se encuentra vinculado a Username -->
        <span asp-validation-for="Username" style="color:red;"></span>
        <!-- Muestra el error específico del campo Username en caso de la existencia de algún problema -->
    </div>

    <div>
        <label asp-for="Password"></label>
        <!-- Se genera una etiqueta label que se encuentra vinculada al campo Password del modelo -->
        <input asp-for="Password" type="password" />
        <!-- Se crea el input del formulario que se encuentra vinculado a Password -->
        <span asp-validation-for="Password" style="color:red;"></span>
        <!-- Muestra el error específico del campo Password en caso de la existencia de algún problema -->
    </div>
```

```
    <div>
        <label asp-for="ConfirmPassword"></label>
        <!-- Se genera una etiqueta label que se encuentra vinculada al campo ConfirmPassword del modelo -->
        <input asp-for="ConfirmPassword" type="password" />
        <!-- Se crea el input del formulario que se encuentra vinculado a ConfirmPassword -->
        <span asp-validation-for="ConfirmPassword" style="color:red;"></span>
        <!-- Muestra el error específico del campo ConfirmPassword en caso de la existencia de algún problema -->
    </div>

    <button type="submit">Registrar</button>
    <!-- Boton que envia el formulario -->
</form>

@section Scripts{
    <partial name="_ValidationScriptsPartial" />
    <!-- Carga los scripts que son necesarios para que las validaciones funcionen en el navegador -->
}
```

4. Login Completo

Lo primero que vamos a realizar en este apartado es crear el modelo del formulario de inicio de sesión. Para ello nos tenemos que ir a la carpeta Models y crearemos el archivo LoginVm.cs.

```
1   //Importamos DataAnnotations para poder usar validaciones como pueede ser dato obligatorio
2   using System.ComponentModel.DataAnnotations;
3   //Esta linea sirve para poder organizar el código dentro del poryecto LoginRegistro; evitando conflictos y poder mantener el orden del proyecto
4   namespace LoginRegistro.Models;
5   //Clase que va a recoger los datos del formulario de login; estos no se almacenan en la base de datos y solo se usa para verlo en el login.
6   public class LoginVm
7   {
8       //Variable Username obligatoria.
9       [Required]
10      public string Username { get; set; } = string.Empty;
11      //Variable Username obligatoria.
12      [Required]
13      public string Password { get; set; } = string.Empty;
14      //Añadimos este campo el cual sirve para que el usuario pueda marcar si el inicio de sesión quiere que se recuerde
15      public bool RememberMe { get; set; }
16  }
```

Ahora toca configurar la autenticación para que el sistema login funcione correctamente para ello dentro del archivo AccountController.cs añadiremos las siguientes líneas:

```
//El metodo Get lo que estamos haciendo es mostrar la vista del login
[HttpGet]
public IActionResult Login()
{
    return View();
}
```

```
//El metodo POST procesa el formulario cuando el usuario procede a entrar con su cuenta
[HttpPost]
//Se protege el formulario contra ataques CSRF
[ValidateAntiForgeryToken]
public async Task<IActionResult> Login(LoginVm vm)
{
    //Primero comprobamos si el modelo del formulario es decir se han introducido todos los campos que son required
    if (!ModelState.IsValid)
        return View(vm);
    //Segundo se busca en la base de datos el usuario que hemos introducido para más adelante comprobar si existe
    var user = await _db.Users.FirstOrDefaultAsync(u => u.Username == vm.Username);
```

```
//Tercero; tal y como se ha comentado anteriormente se comprueba si el usuario existe y si la contraseña es correcta; en caso de
//que no suceda muestra un mensaje de error diciendo que las credenciales son inválidas
if (user == null || !VerifyPassword(vm.Password, user.PasswordHash))
{
    ModelState.AddModelError(string.Empty, "Credenciales inválidas.");
    return View(vm);
}

//Cuarto si el login es correcto, se crean los claims que son datos que identifican al usuario dentro de la sesión
var claims = new List<Claim>
{
    //Se guarda el Id del usuario como identificador principal
    new Claim(ClaimTypes.NameIdentifier, user.Id.ToString()),
    //Se guarda el nombre de usuario
    new Claim(ClaimTypes.Name, user.Username)
};
//Quinto, se crea la identidad del usuario con el esquema de cookies
var identity = new ClaimsIdentity(claims, CookieAuthenticationDefaults.AuthenticationScheme);
//Sexto se crea la variable principal, que representa al usuario que se encuentra logueado en el sistema.
var principal = new ClaimsPrincipal(identity);
```

```
    //Septimo se configura como se va a guardar la cookie de login
    var authProps = new AuthenticationProperties
    {
        IsPersistent = vm.RememberMe,
        ExpiresUtc = vm.RememberMe
            ? DateTimeOffset.UtcNow.AddDays(7)
            : DateTimeOffset.UtcNow.AddMinutes(30)
    };
    //Octavo se inicia la sesión y se crea la cookie de autenticación necesaria
    await HttpContext.SignInAsync(CookieAuthenticationDefaults.AuthenticationScheme, principal, authProps);
    //Si el login es correcto redirigimos al Home/Index
    return RedirectToAction("Index", "Home");
}
```

```
//Metodo privado en la cual se comprueba si la contraseña escrita en el login es la misma que hay en la base de datos
private static bool VerifyPassword(string password, string storedHash)
{
    //Lo primero que realizamos es separar el texto guardado usando el punto "."
    var parts = storedHash.Split('.');
    //En caso de que no tenga las dos partes, esto significa que se encuentra mál guardado
    if (parts.Length != 2) return false;
    //Segundo se convierte la salt de Base64 a bytes
    var salt = Convert.FromBase64String(parts[0]);
    //Tercero se convierte el hash guardadode Base64 a bytes
    var hashStored = Convert.FromBase64String(parts[1]);
    //Cuarto calculamos el hash de la contraseñaque el usuario ha escrito; a parte usamos el mismo salt y el mismo método PBKDF2
    var pbkdf2 = new Rfc2898DeriveBytes(password, salt, 100000, HashAlgorithmName.SHA256);
    var hashComputed = pbkdf2.GetBytes(32);
    //Por ultimo se compara el hash guardado con el hash que se ha calculado
    return CryptographicOperations.FixedTimeEquals(hashStored, hashComputed);
}
```

Para finalizar vamos a crear la vista login que será la página donde el usuario puede iniciar sesión para ello vamos a crear el archivo Login.cshtml dentro de la carpeta Views/Account.

```
@model LoginRegistro.Models.LoginVm

@{
    ViewData["Title"] = "Login";
    //Se define el nombre del titulo de la página
}

<h2>Login</h2>
<!-- Título que será visible para el usuario -->
<form asp-action="Login" method="post">
    <!-- Formulario que se va a enviar para cuando se haga el login; usamos el método POST debido a que se van a enviar los datos del formulario -->
    @Html.AntiForgeryToken()
    <!-- Token de seguridad el cual se va a evitar ataques CSRF; este sirve para comprobar que el formulario se ha enviado desde la web -->

    <div asp-validation-summary="All" class="text-danger"></div>
    <!-- En caso de que se produzca algún error aquí estamos indicando donde se debe de mostrar -->

    <div class="mb-2">
        <!-- Se crea el campo Username -->
        <label asp-for="Username"></label>
        <!-- Se vincula el Input al campo Username del modelo Login -->
        <input asp-for="Username" class="form-control" />
        <!-- Se muestra si hay un error específico en el campo Username -->
        <span asp-validation-for="Username" class="text-danger"></span>
    </div>
```

```
    <div class="mb-2">
        <!-- Se crea el campo Password -->
        <label asp-for="Password"></label>
        <!-- Se vincula el Input al campo Password del modelo Login -->
        <input asp-for="Password" type="password" class="form-control" />
        <!-- Se muestra si hay un error específico en el campo Password -->
        <span asp-validation-for="Password" class="text-danger"></span>
    </div>

    <div class="form-check mb-3">
        <!-- Se vincula el Input al campo RememberMe -->
        <input asp-for="RememberMe" class="form-check-input" />
        <!-- Se muestra el texto Recordarme el cual se indica al usuario para que sirve -->
        <label asp-for="RememberMe" class="form-check-label">Recordarme</label>
    </div>

    <button type="submit" class="btn btn-primary">Entrar</button>
    <!-- Botón que envia el formulario -->
</form>
```

```
@section Scripts{
    <partial name="_ValidationScriptsPartial" />
    <!-- Se cargan todos los scripts para validar los datos de los clientes -->
}
```

5. Sesión Activa y Zona Privada Protegida.

En este apartado vamos a crear un controlador privado; esto significa que cuando el usuario se ha logueado le redireccionara a una página web que solamente tendrán acceso aquellas personas que hagan el login correctamente para ello dentro de la carpeta Controllers vamos a crear el archivo PrivateController.cs.

```
//Importamos Authorization que sirve para proteger las páginas y que solo entren usuarios que han sido logueados
using Microsoft.AspNetCore.Authorization;
//Importamos lo necesario para usar Controllers y devolver la vista
using Microsoft.AspNetCore.Mvc;

//Esta linea sirve para poder organizar el código dentro del poryecto LoginRegistro; evitando conflictos y poder mantener el orden del proyecto
namespace LoginRegistro.Controllers;

//Authorize significa que este controlador se encuentra protegido y solo podrán acceder los usuarios que hayan iniciado sesión
[Authorize]
public class PrivateController : Controller
{
    //Se devuelve la vista del Index.cshtml que se encuentra dentro de Views/Private
    public IActionResult Index()
    {
        return View();
    }
}
```

Una vez que nos hemos creado el archivo y lo hemos completado vamos a crear la vista privada para ello dentro de la carpeta Views vamos a crear la carpeta Private y dentro de ella el Index.cshtml

```
@{
    //Se define el título de la página
    ViewData["Title"] = "Zona Privada";
}

<h2>Zona Privada</h2>
<!-- Titulo visible que se va a ver en la página -->

<p>Bienvenido, <strong>@User.Identity?.Name</strong></p>
<!-- Damos la Bienvenido al usuario que se encuentra logueado; y mostramos el nombre de usuario gracias a las cookies -->

<p>Si puedes ver esta página, significa que estás logueado correctamente.</p>
<!-- Se muestra un texto en el cual se indica que si se puede ver esa página ha iniciado sesión correctamente-->
```

6. Logout (Cierre de la Sesión)

Una vez que ya tenemos configurado el registro, el login y la página personal vamos a añadir una opción la cual el usuario pueda cerrar sesión cuando se termine de usar la aplicación para ello tenemos que ir a la carpeta de Controllers y en el archivo AccountController.cs introducimos el siguiente código:

```
//Método el cual se ejecuta cuando el usuario cierra sesión
[HttpPost]
//Token de seguridad para evitar ataques CSRF
[ValidateAntiForgeryToken]
public async Task<IActionResult> Logout()
{
    //Se cierra la sesión del usuario eliminado. SignOutAsync borra la cookie en la cual se creó cuando se hizo el login
    await HttpContext.SignOutAsync(CookieAuthenticationDefaults.AuthenticationScheme);

    //Cuando se cierra la sesión redirigimos al usuario al inicio de la web
    return RedirectToAction("Index", "Home");
}
```

A continuación, se va a añadir en la página privada un botón para cerrar sesión:

```
<form asp-controller="Account" asp-action="Logout" method="post">
<!-- Formulario que se llama al controlador Account y a la acción Logout -->
    @Html.AntiForgeryToken()
    <!-- Token de Seguridad para evitar ataques CSRF -->
    <button type="submit" class="btn btn-danger">Logout</button>
    <!-- Botón que envía el formulario -->
</form>
```