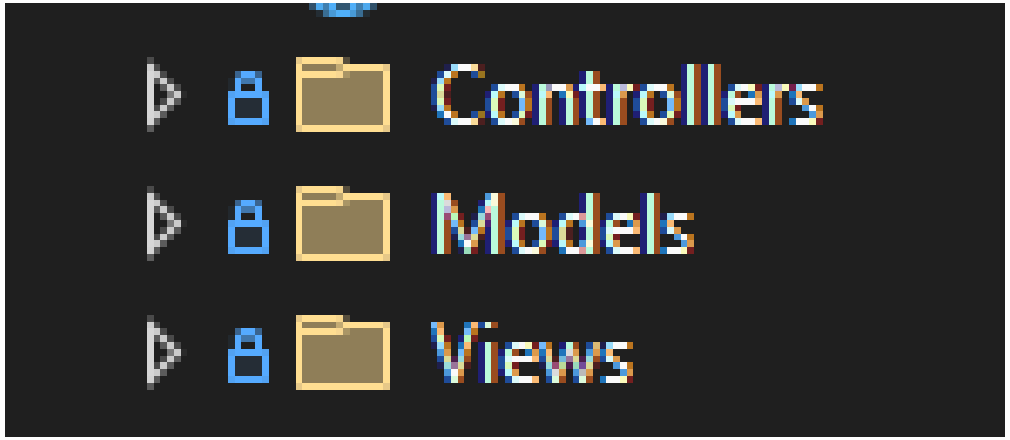
Explicación del Patrón MVC en ASP.NET Core

Entendiendo MVC en ASP.NET Core

"Modelo - Vista - Controlador"



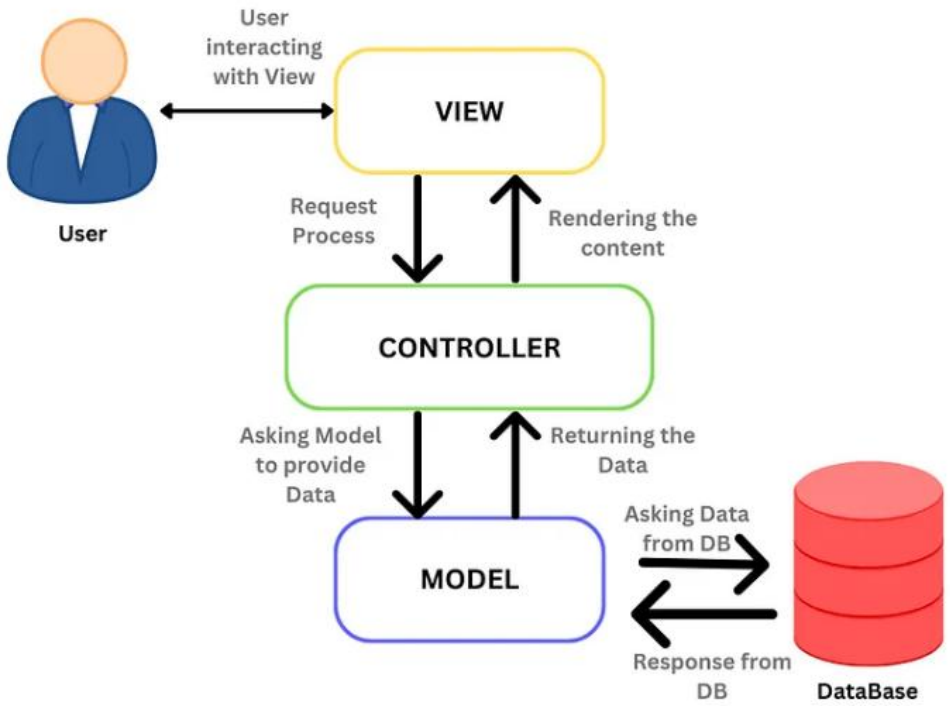
Repaso a MVC

Definición breve del patrón MVC

*Dia*grama MVC

Diagrama visual del flujo MVC

Usuario ➞ Controlador ➞ Modelo ➞ Vista



El Modelo

1. Clase C#, representa los datos
2. Incluye validaciones con Data Annotations

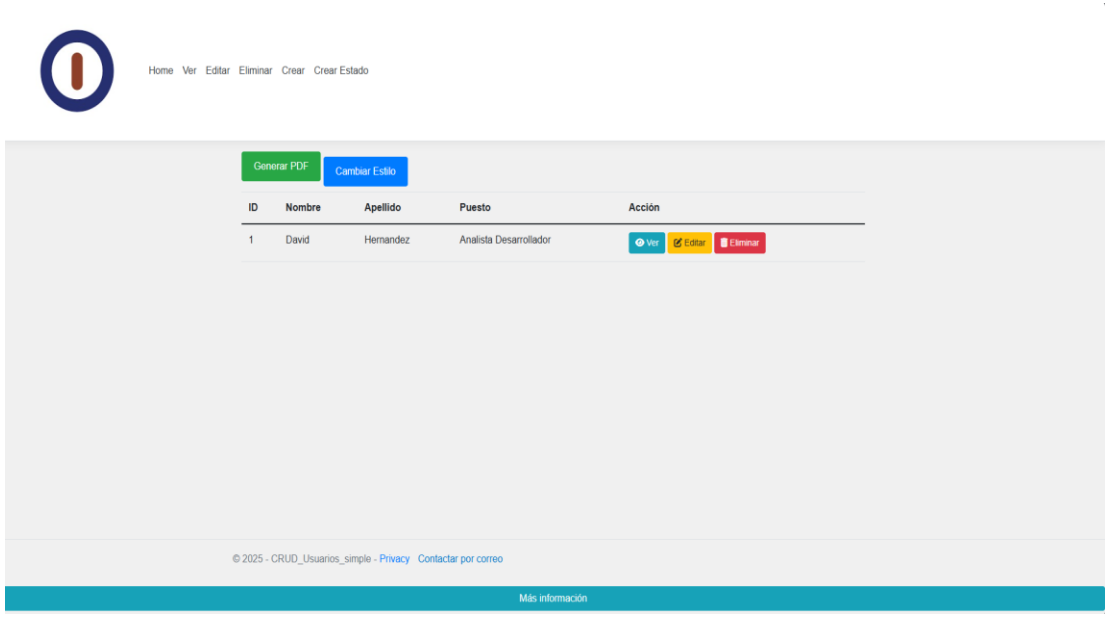
Validación

```
public class UsuarioViewModel
    {
        [Required(ErrorMessage = "El campo {0} es obligatorio.")]
        2 referencias
        public int Id { get; set; }
```

Propiedad: representa un espacio de la base de datos

La Vista

- Archivos .cshtml
- Contienen HTML + Razor
- Se vinculan al modelo con @model



¿Como se relaciona la vista y el modelo?

Concepto General:

En el patrón **MVC**, la **Vista** (View) es la encargada de mostrar la información al usuario, mientras que el **Modelo** (Model) contiene los datos y la lógica de negocio.

La Vista **no debería contener lógica compleja**, solo la presentación de datos. Por eso, la relación entre ambos es **a través de datos que el controlador pasa a la vista**, casi siempre en forma de un objeto del modelo.

El Controlador

- Clase que recibe las peticiones del usuario
- Conecta modelo y vista

Modelo de datos USUARIO

```
16 referencias
public class UsuarioViewModel
    {
        [Required(ErrorMessage = "El campo {0} es obligatorio.")]
        2 referencias
        public int Id { get; set; }
        6 referencias
        public string Nombre { get; set; }
        6 referencias
        public string Apellido { get; set; }
}
```

Tu controlador recibe ese modelo y lo pasa a la vista:

```
0 referencias
public IActionResult Crear()
{
    return View(new UsuarioViewModel());
}

[HttpPost]
[ValidateAntiForgeryToken]
0 referencias
public IActionResult CrearUsuario(UsuarioViewModel usuario)
{
    if (ModelState.IsValid)
    {
        usuario.Id = _usuarios.Count + 1;
        _usuarios.Add(usuario);

        return RedirectToAction(nameof(Index));
    }
    return View("Crear", usuario);
}
```

Conexión entre componentes

Ejemplo de código: controlador que recibe modelo y retorna vista

La vista usa @model para indicar qué tipo de datos está recibiendo.

Luego accede a las propiedades del modelo con @Model.Propiedad.

```
@model UsuarioViewModel;

@{
    ViewData["Title"] = "Crear Usuario";
}

<div class="container">
<h1>Crear Usuario</h1>
    <form id="crearUsuarioForm" asp-action="CrearUsuario" asp-controller="Crear" method="post" class="row g-3">
    <div class="col-md-6">
        <label asp-for="Nombre" class="form-label"></label>
            <input id="Nombre" asp-for="Nombre" type="text" class="form-control" placeholder="Nombre del usuario" required>
        <!--span style="color:red;" asp-validation-for="Nombre"-->
            <!--/--span-->
            <span id="errorNombre" style="color:red; font-size: 0.875em; display:none;"></span>
    </div>
```

¿Qué está pasando?

1. El modelo Usuario es creado y llenado en el controlador.

2. El controlador pasa ese modelo a la vista.

3. La vista usa @model para indicar qué tipo de datos está recibiendo.

4. Luego accede a las propiedades del modelo con @Model.Propiedad.

Cuando el usuario envía datos

Si tienes un formulario, por ejemplo para crear un nuevo usuario:

```
<h1>Crear Usuario</h1>
    <form id="crearUsuarioForm" asp-action="CrearUsuario" asp-controller="Crear" method="post" class="row g-3">
    <div class="col-md-6">
        <label asp-for="Nombre" class="form-label"></label>
            <input id="Nombre" asp-for="Nombre" type="text" class="form-control" placeholder="Nombre del usuario" required>
            <span id="errorNombre" style="color: red; font-size: 0.875em; display:none;"></span>
    </div>
    <div class="col-6">
        <button type="submit" class="btn btn-success">Crear Usuario</button>
    </div>
    <div class="col-6">
        <button class="btn btn-primary">Regresar</button>

    </div>
</form>
```

Cuando el usuario envía datos

Si tienes un formulario, por ejemplo para crear un nuevo usuario:

```
<form asp-action="Crear" method="post">
    <input asp-for="Nombre" />
    <input asp-for="Edad" type="number" />
    <button type="submit">Guardar</button>
</form>
```

Y el controlador recibe los datos así:

```
[HttpPost]
public IActionResult Crear(Usuario usuario)
{
    // Aquí el modelo se llena automáticamente con los datos del formulario
    if (ModelState.IsValid)
    {
        // Guardar, procesar, etc.
    }

    return View(usuario);
}
```

Validaciones desde el modelo

Con Data Annotations puedes validar desde el modelo:

```
public class Usuario
{
    [Required(ErrorMessage = "El nombre es obligatorio.")]
    public string Nombre { get; set; }

    [Range(18, 99, ErrorMessage = "La edad debe estar entre 18 y 99.")]
    public int Edad { get; set; }
}
```

Y en la vista puedes mostrar los mensajes de error con:

```
<span asp-validation-for="Nombre" class="text-danger"></span>
```

Nota importante

1. El modelo se usa tanto para mostrar como para recibir datos.
2. La vista no debe tener lógica, solo mostrar información y recoger datos del usuario.

Conclusión:

1. La vista y el modelo se relacionan porque:
2. La vista muestra los datos del modelo.
3. El modelo recibe los datos del usuario a través de formularios.
4. Todo esto es gestionado por el controlador, que actúa como intermediario.

Validaciones en el Modelo

- Uso de [Required], [Range], etc.
- Mostrar errores en la vista con asp-validation-for

Conexión con Base de Datos

- Contexto DbContext
- appsettings.json
- Entity Framework

Preguntas Frecuentes

1.¿Cómo se conectan Vista, Modelo y Controlador?

2.¿Dónde van las validaciones?

3.¿Cómo se conecta la BD?

4.¿Diferencia entre controladores, vistas, archivos Razor?

Conclusión

- MVC mejora la estructura del código
- Cada componente tiene su rol
- Es clave para proyectos escalables