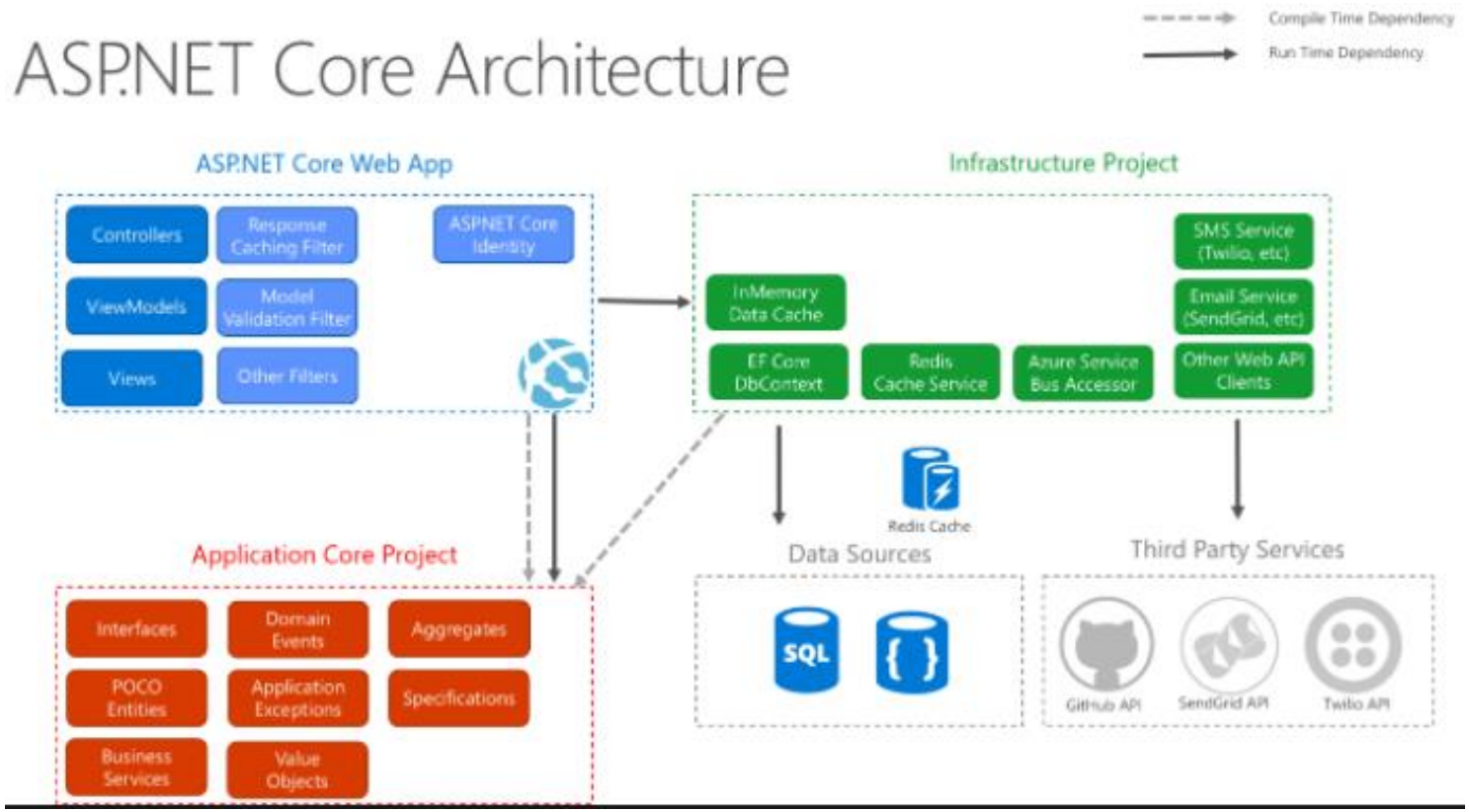
MVC

Modelo – Vista - Controlador

MVC en el Framework

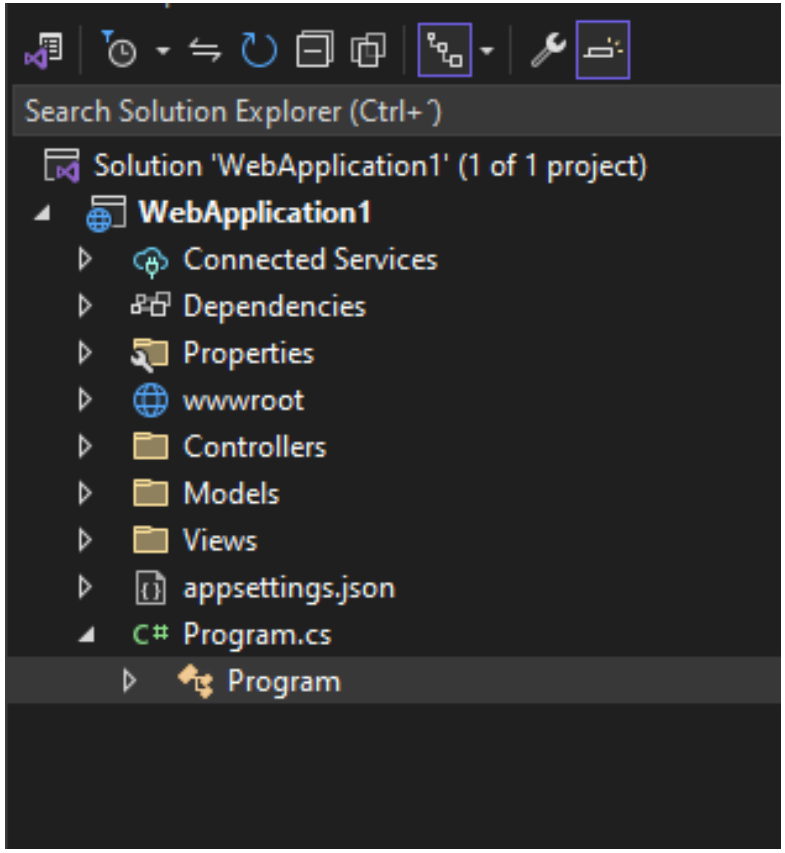


[+ info]

PATRÓN MVC

+ info

Estructura app MVC



Archivo program.cs

```
{
    0 references
    public static void Main(string[] args)
    {
        var builder = WebApplication.CreateBuilder(args);

        // Add services to the container.
        builder.Services.AddControllersWithViews();

        var app = builder.Build();

        // Configure the HTTP request pipeline.
        if (!app.Environment.IsDevelopment())
        {
            app.UseExceptionHandler("/Home/Error");
            // The default HSTS value is 30 days. You may want to change this for production scenarios
            app.UseHsts();
        }

        app.UseHttpsRedirection();
        app.UseStaticFiles();

        app.UseRouting();

        app.UseAuthorization();

        app.MapControllerRoute(
            name: "default",
            pattern: "{controller=Home}/{action=Index}/{id?}");

        app.Run();
    }
}
```

MVC - Características

- El modelo encapsula objetos y datos.
- La Vista genera la interfaz de usuario.
- Los controladores interactúan con las acciones de los usuarios.
- El código está en archivos .cshtml y .cs.
- Permite un control preciso del HTML y de las URLs.

Modelo

- Puede ser un objeto, un string, una colección o un array.
- Esto no implica que sea dependiente del sistema de almacenamiento.
- Definir las reglas de negocio (la funcionalidad del sistema).
  Un ejemplo de regla puede ser: "Si el artículo llegó al stock mínimo, incluirlo en una lista de artículos a comprar.
- Habitualmente incluye:
  - Entidades de negocio
  - Clases de lógica empresarial, que implementan los procesos y reglas de negocio
- Mecanismos de acceso a datos a través de:
  - ADO.NET,
  - Un ORM que nos aísle de la persistencia, por ejemplo Entity framework

Controlador

Es el elemento encargado de recibir y procesar la peticiones Http. Su responsabilidad es retornar una respuesta Response Http.

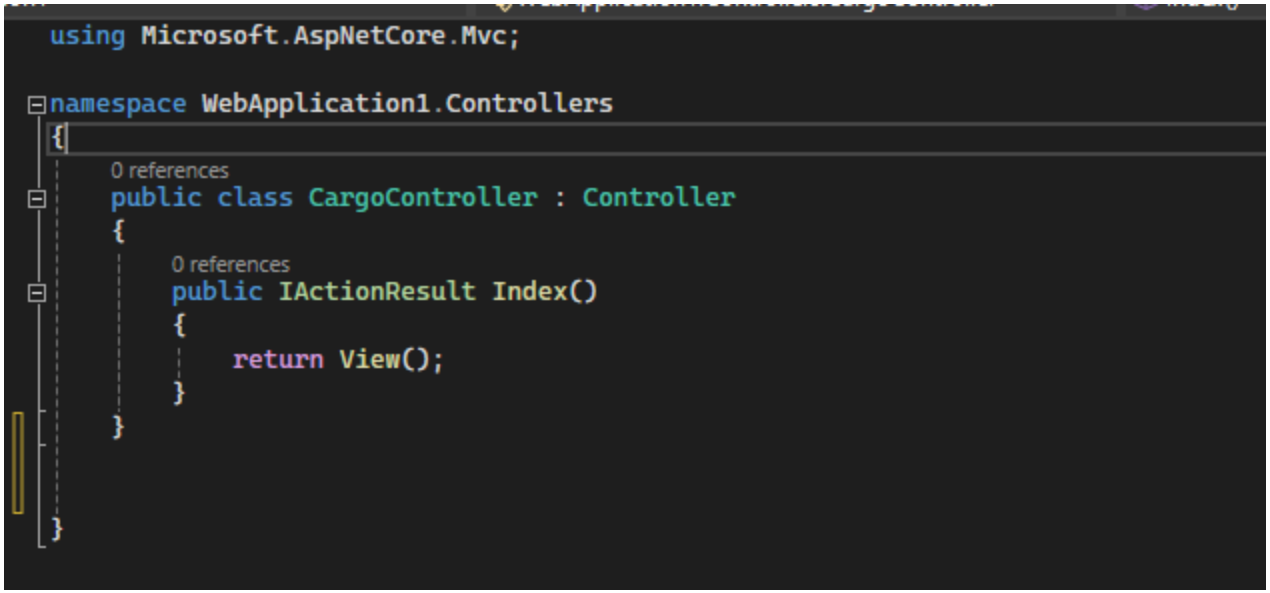
- Recibe la petición Http.
- Procesa los datos, y crea u obtiene el modelo.
- Retorna una respuesta Http.

+ info

¿Cómo funciona?

- Un usuario hace una solicitud al sitio, y en lugar de servir un archivo (o procesar el código de un archivo) en MVC se invoca un método de una clase.
- Esa clase es un controlador. Es quien “hace cosas”, por ejemplo autenticar, retornar una lista de los clientes, armar un gráfico, etc.
- Lo que retorna es el “modelo”, que es simplemente una clase. No se conecta directo a la BD, sino que retorna el resultado al que debe acceder el usuario.
- El controlador incluye la lógica para “buscar” y “retornar” los datos.

Código básico de un controlador



```
using Microsoft.AspNetCore.Mvc;

namespace WebApplication1.Controllers
{
    0 references
    public class CargoController : Controller
    {
        0 references
        public IActionResult Index()
        {
            return View();
        }
    }

}
```

Pasaje de información entre vista y el controlador

ViewBag es utilizado, para pasar datos desde el controller a la vista, su duración es solo durante el request actual, y una vez terminado este, el dato es borrado, se utiliza para pasar cualquier tipo de datos, incluso objetos, se crea en forma dinámica.

Controller:
ViewBag.nombreCompleto= “Pedro Pérez”

Vista:
Mi nombre es @ViewBag.nombreCompleto

Pasaje de información entre vista y el controlador

ViewData es utilizado para pasar datos desde el controller a la vista, su duración es solo durante el actual request, una vez que este termina, este dato se habrá borrado, de ahi que su uso es solo temporal, comúnmente es usado para mandar mensajes de error a la vista. Otro dato importante en cuanto su uso, es que se lo puede utilizar como si fuera un arreglo, aquí un ejemplo:

Controller:

```
ViewData["nombre"] = "Pedro";
ViewData["apellido"] = "Pérez";
```

Vista:

```
Mi nombre es @ViewData["nombre"] @ViewData["apellido"]
```

Pasaje de información entre vista y el controlador

Model es otro mecanismo de enviar y recibir información, pero a través de los modelos del dominio. Aquí un ejemplo:

Controller:
Para recibir el modelo public ActionResult Alta(Cargo cargo)
El objeto es enviado desde la petición http

Para enviarlo a la vista
return View(cargo)

Vista:
@model cargo

Posibles retornos

Retornar una vista:

Se envía el objeto cargo a través de un ViewBag.Cargo

```
0 references
public ActionResult Index()
{
    ViewBag.Cargo = unS.ObtenerCargo(5);
    return View();
}
```

Se envía un objeto cargo a través del Model, el mismo es enviado en el parámetro View()

```
0 references
public ActionResult Index()
{
    Cargo unC = unS.ObtenerCargo(5);
    return View(unC);
}
```

Posibles retornos

También se puede redirigir a otra acción:

```
0 references
public ActionResult AltaMensual(int id, Mensual cargo)
{
    unS.AgregarCargoMensual(cargo);
    return RedirectToAction("IndexJson", new { id = id});
}

0 references
public IActionResult BUscar()
{
    return Redirect("https://www.gogole.com");
}
```

Model Binding

Nos permite asignar los datos de una petición http a un modelo de nuestro dominio. En enlace y conversión de tipos de los valores lo hace por nombre.

Por ejemplo, los datos de ruta pueden proporcionar valores a través de la query string o los campos de formulario pueden proporcionar valores para las propiedades del modelo. + info

Filtros

Se pueden definir filtros para las acciones de los controladores y sirve para que dicho método sea ejecutado si corresponde con el verbo del Http, ejemplo:

[HttpGet] valor por defecto
[HttpPost]

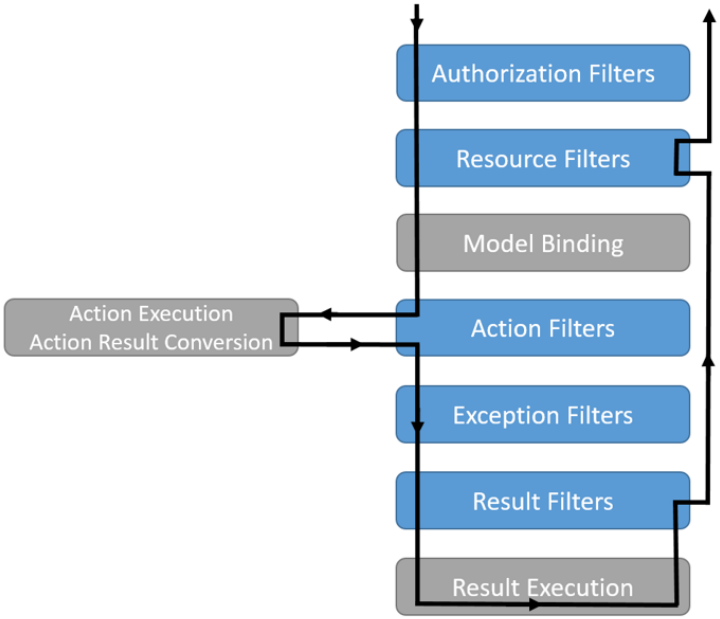
```
Sistema unS = new Sistema();

[HttpGet]
0 references
public ActionResult AltaMensual()
{
    return View();
}

[HttpPost]
0 references
public ActionResult AltaMensual(int id, Mensual cargo)
{
    if (unS.AgregarCargoMensual(cargo))
    {
        return RedirectToAction("Ver", new { id = id });
    }
    else
    {
        return View(cargo);
    }
}
```

Filtros

Se pueden definir filtros personalizados para centralizar los acciones de los controladores. Por ejemplo el filtro de autorizacion mediante la interfaz IAuthorizationFilter




```
namespace AppWeb.Filtros
{
    1 reference
    public class Admin : Attribute, IAuthorizationFilter
    {
        0 references
        public void OnAuthorization(AuthorizationFilterContext context)
        {
            string rol = context.HttpContext.Session.GetString("rol");
            if (string.IsNullOrEmpty(rol))
            {
                context.Result = new RedirectResult("/usuario/index");
                return;
            }
            if (rol != "admin")
            {
                context.Result = new RedirectResult("/cargo/index");
                return;
            }
        }
    }
}
```

Vista

- El contenido que va a ser devuelto como consecuencia de la petición HTTP. La vista recibe el modelo para poder realizar la presentación de los datos.
- Es una representación de los datos.
- Toma lo que contiene el modelo y lo convierte en HTML.
- Es un sistema de plantillas.
- No tiene lógica, sólo HTML.

+ info

Vistas

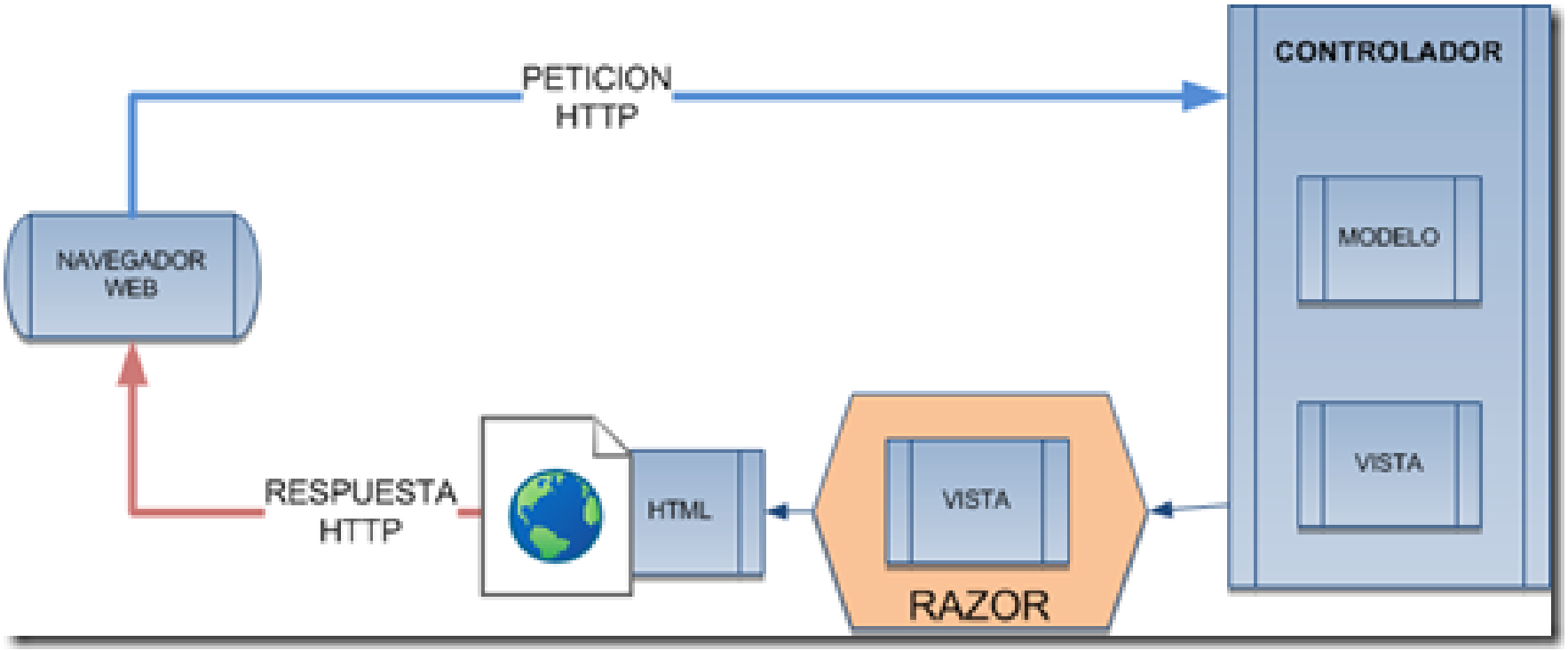
Se crea una vista Razor View - Empty

```
@*
    For more information on enabling MVC for empty projects, visit https://go.microsoft.com/fwlink/?LinkID=397860
*@
@{
}

<h1>Alumno</h1>


@if (ViewBag.Edad > 18)
{
    <p>@ViewBag.Nombre es mayor de edad.</p>
}
else
{
    <p>@ViewBag.Nombre tiene @ViewBag.edad años</p>
}
```

MOTOR DE VISTAS RAZOR



Routing

- El enrutamiento es responsable de hacer coincidir las solicitudes HTTP entrantes y de enviarlas a los puntos de conexión ejecutables de la aplicación.
- Son atendidos por las acciones del controlador.
- Se definen en la aplicación y se configuran al iniciarla.
- El proceso de búsqueda de coincidencias de puntos de conexión puede extraer valores de la dirección URL de la solicitud y proporcionarlos para el procesamiento de la solicitud. Si no encuentra una concidencia da un error 404.

+ info

Routing

- Se debe agregar en la aplicación useRouting y UseEndPoint en el archivo Startup.cs
- La petición HTTP es recibida, para detectar cuál es el controlador y qué método invocar , se accede a un “mapa de enrutamiento”.

URL Limpias

¿Qué son las URL amigables y para qué sirve?
Las URLs amigables están especialmente diseñadas para satisfacer las necesidades del público objetivo y buscadores web. Sus elementos, organizados de una forma lógica, tienen un impacto positivo en la visibilidad y el posicionamiento del sitio web

URL sin sistema de enrutamiento:
http://ventasOnline.com/productos.aspx?id=12

URL con sistema de enrutamiento:
http://ventasOnline.com/producto/ver/12

Formato de una ruta

- Incluyen marcadores de posición (habitualmente {controller} y {action})
  Formato: {controller}/{action}/{id}

- Una dirección URL que incluye la ruta de acceso /Producto está asignada a un controlador denominado ProductoController. El valor del parámetro action es el nombre del método de acción que se invoca.

- Una URL que incluya /Producto/Ver invocará el método Ver de la clase ProductoController

Mapeo de rutas

- Las rutas por defecto están incluidas en el archivo StartUp.cs.
- Se puede editar para modificarlas.

```
endpoints.MapControllerRoute(
        name: "default",
        pattern: "{controller=Home}/{action=Index}/{id?}");
```