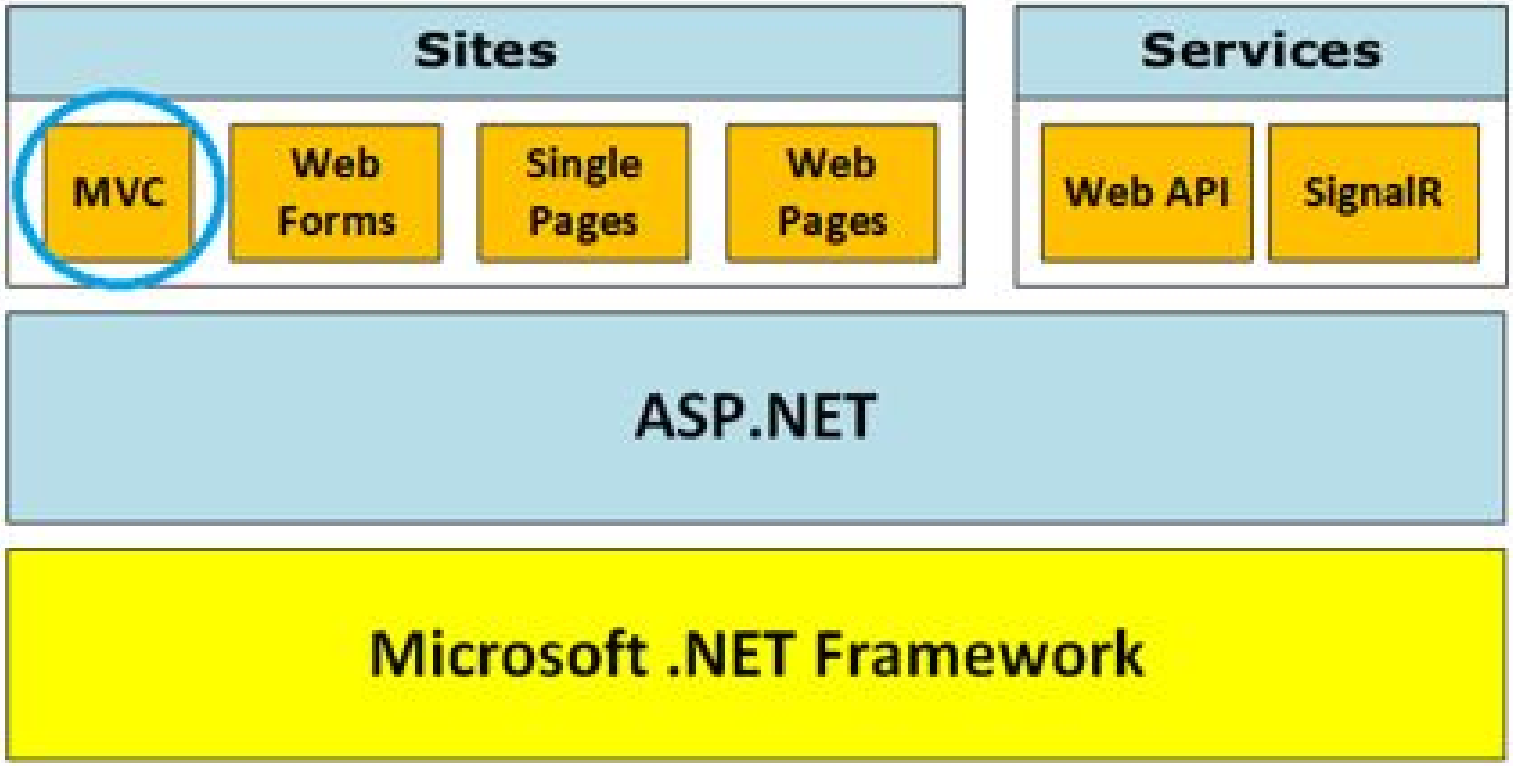
Desarrollador de Aplicaciones Web
Programación Web III

Departamento de Ingeniería e Investigaciones Tecnológicas

Introducción a ASP.NET MVC

Ing. Mariano Juiz
Ing. Matias Paz Wasiuchnik
Ing. Pablo Nicolás Sanchez

ASP.NET: Alcance - Visión global



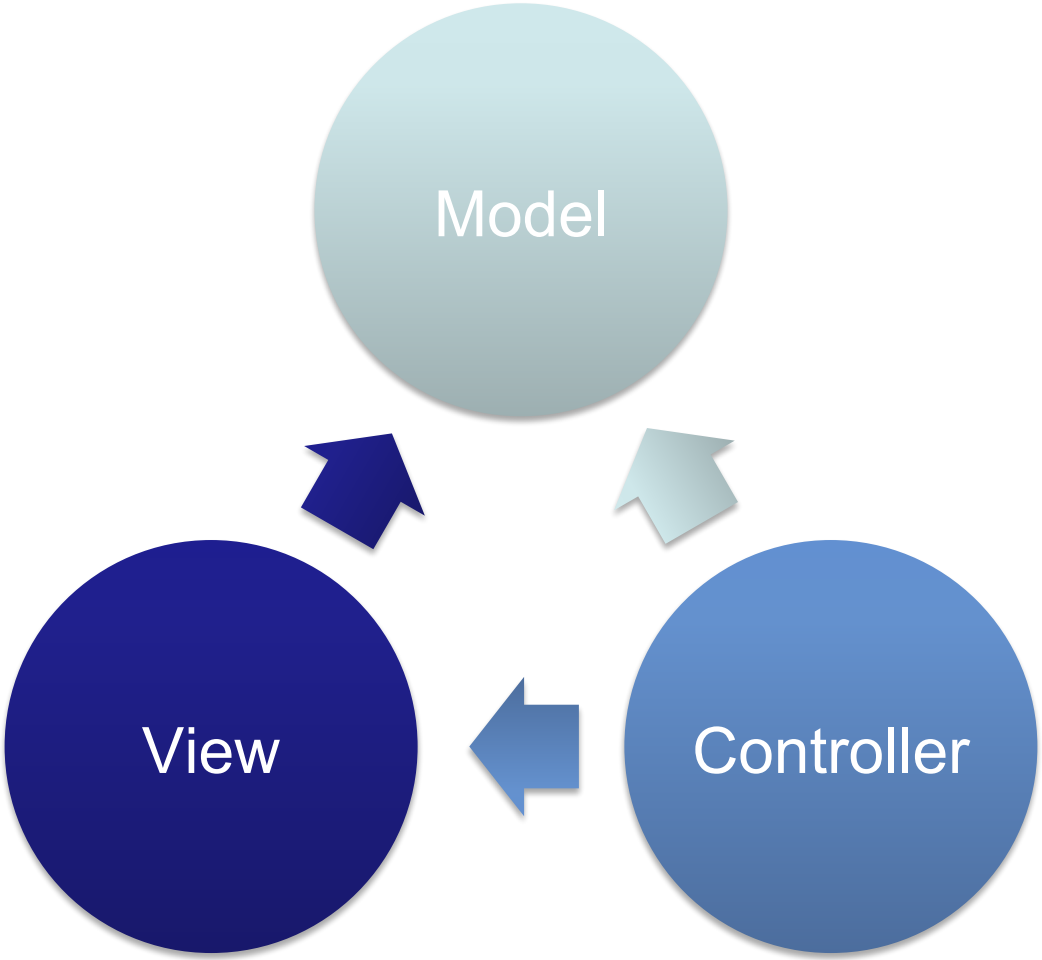
Patrón MVC

El patrón de diseño MVC (Modelo-Vista-Controlador) ha existido desde hace algunas décadas, y se ha usado en muchas tecnologías diferentes. Desde Smalltalk, C ++, Java, .NET sea ha utilizado este patrón de diseño para construir una interfaz de usuario.

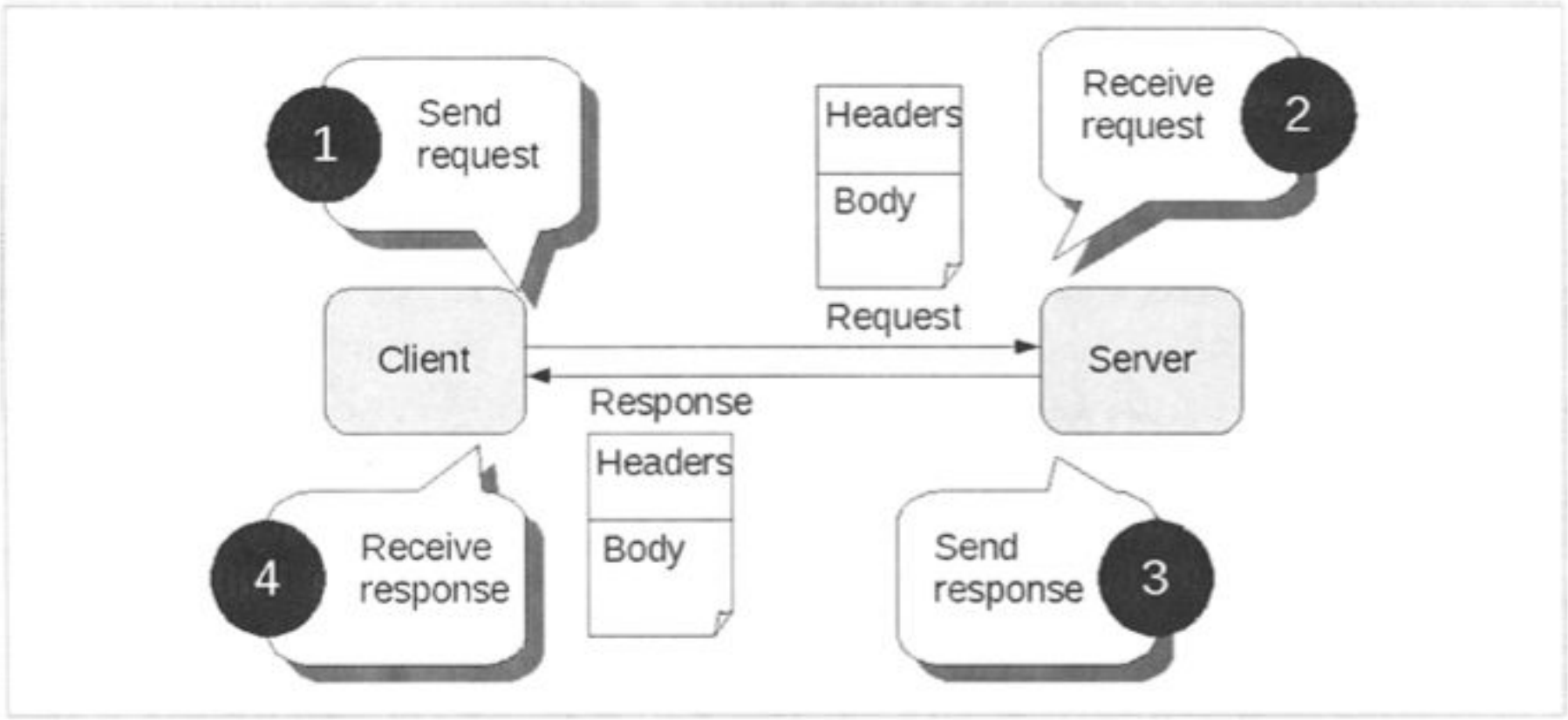
Es un medio poderoso y elegante de la separación de las responsabilidades dentro de una aplicación (por ejemplo, la separación de la lógica de acceso a datos de la lógica de visualización) y se aplica en sí muy bien a las aplicaciones web.

El patrón de arquitectura MVC separa la interfaz de usuario (UI) de una aplicación en tres partes principales: Modelo - Vista – Controlador.

Modelo Vista Controlador



Repasando una vez mas http:



Ciclo de Vida: MVC

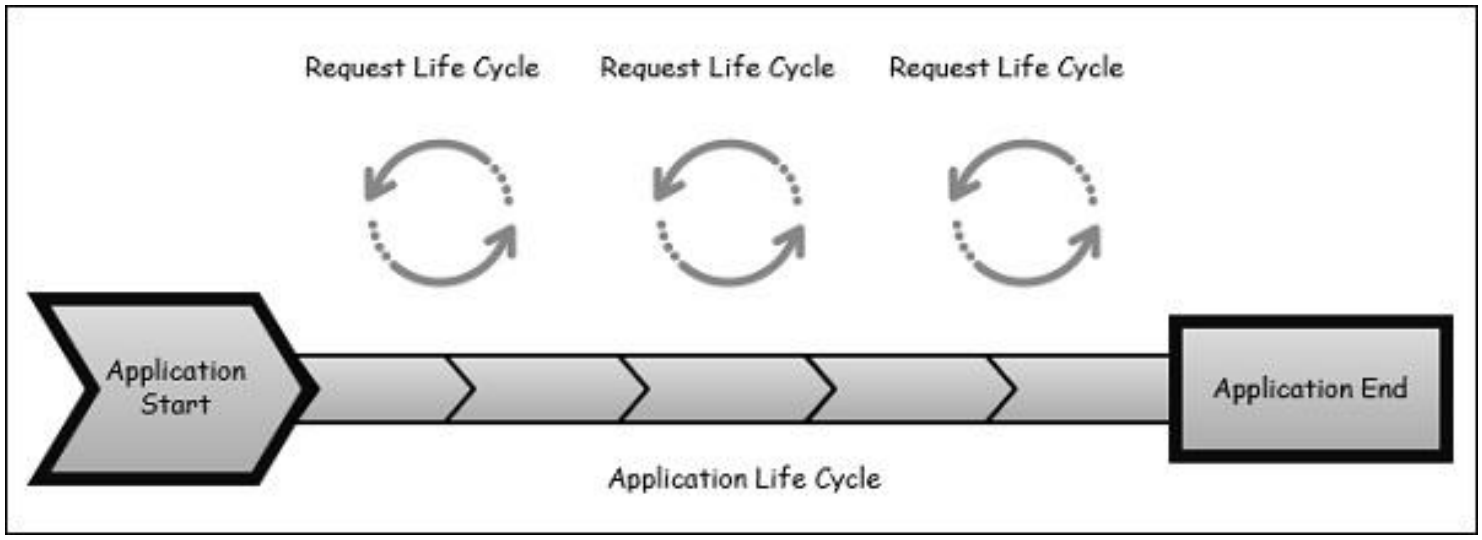
En un nivel alto, un ciclo de vida es simplemente una serie o secuencia de pasos o eventos que se utilizan para manejar algún tipo de solicitud o para cambiar un estado de aplicación.

MVC tiene dos ciclos de vida:

- El ciclo de vida de las aplicaciones
- El ciclo de vida de la solicitud

Ciclo de vida de las aplicaciones:

El ciclo de vida de la aplicación se refiere al tiempo en el que el proceso de aplicación comienza realmente, es decir desde que IIS empieza a funcionar sobre la aplicación hasta el momento en que se detiene. Esto está marcado por los eventos de inicio y fin de aplicación en el archivo de inicio de la aplicación.

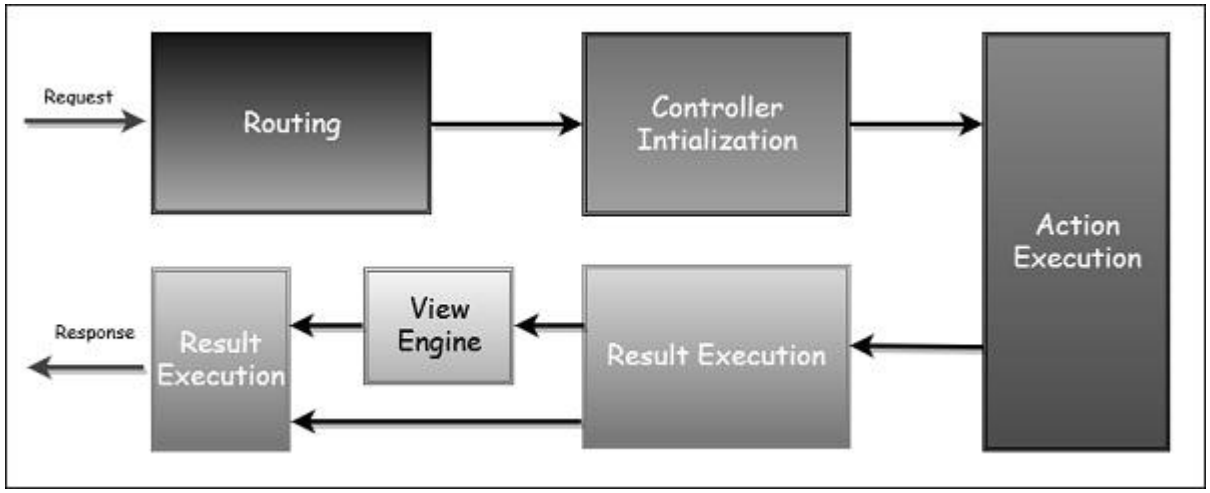


Ciclo de vida de la solicitud:

Es la secuencia de eventos que suceden cada vez que una petición HTTP es manejado por nuestra aplicación.
El punto de entrada para cada aplicación MVC comienza con el enrutamiento. Después que ASP.NET ha recibido una solicitud, gestiona la url mediante el modulo de enrutamiento.

Los módulos son componentes .NET que se pueden enganchar en el ciclo de vida de las aplicaciones y añadir funcionalidad. El módulo de enrutamiento es responsable de resolver la coincidencia entre la URL de entrada y las rutas validadas definidas en nuestra aplicación.

Todas las rutas tienen un controlador asociado con ellas y este es el punto de entrada al marco MVC.



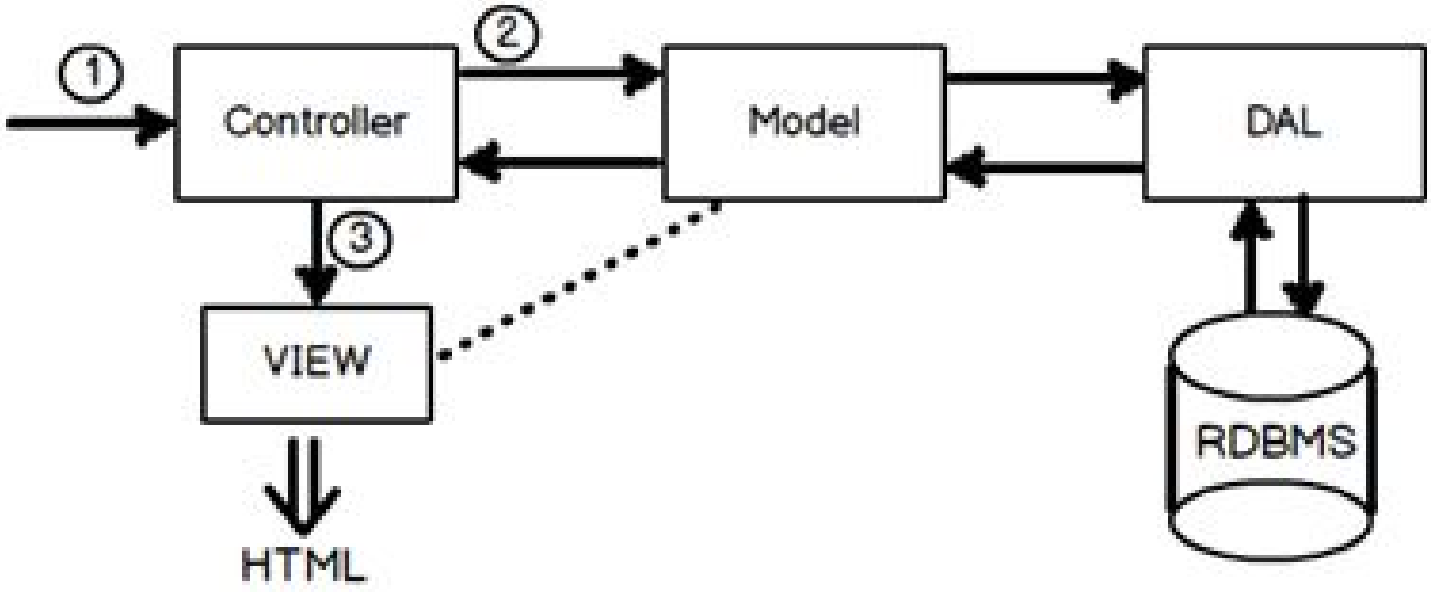
Ciclo de vida de la solicitud:

El marco MVC se encarga de resolver la ruta escogiendo un controlador concreto que puede manejar las peticiones. Una vez elegido el controlador, el siguiente paso importante es **la acción de ejecución.** Un componente llamado **invocador de acción encuentra y selecciona un método de acción apropiado sobre el controlador**

Después que la acción es invocada, la siguiente etapa activa es **el resultado de ejecución.** Si el resultado es una vista, la vista del motor es invocada y este se encarga de procesarla.

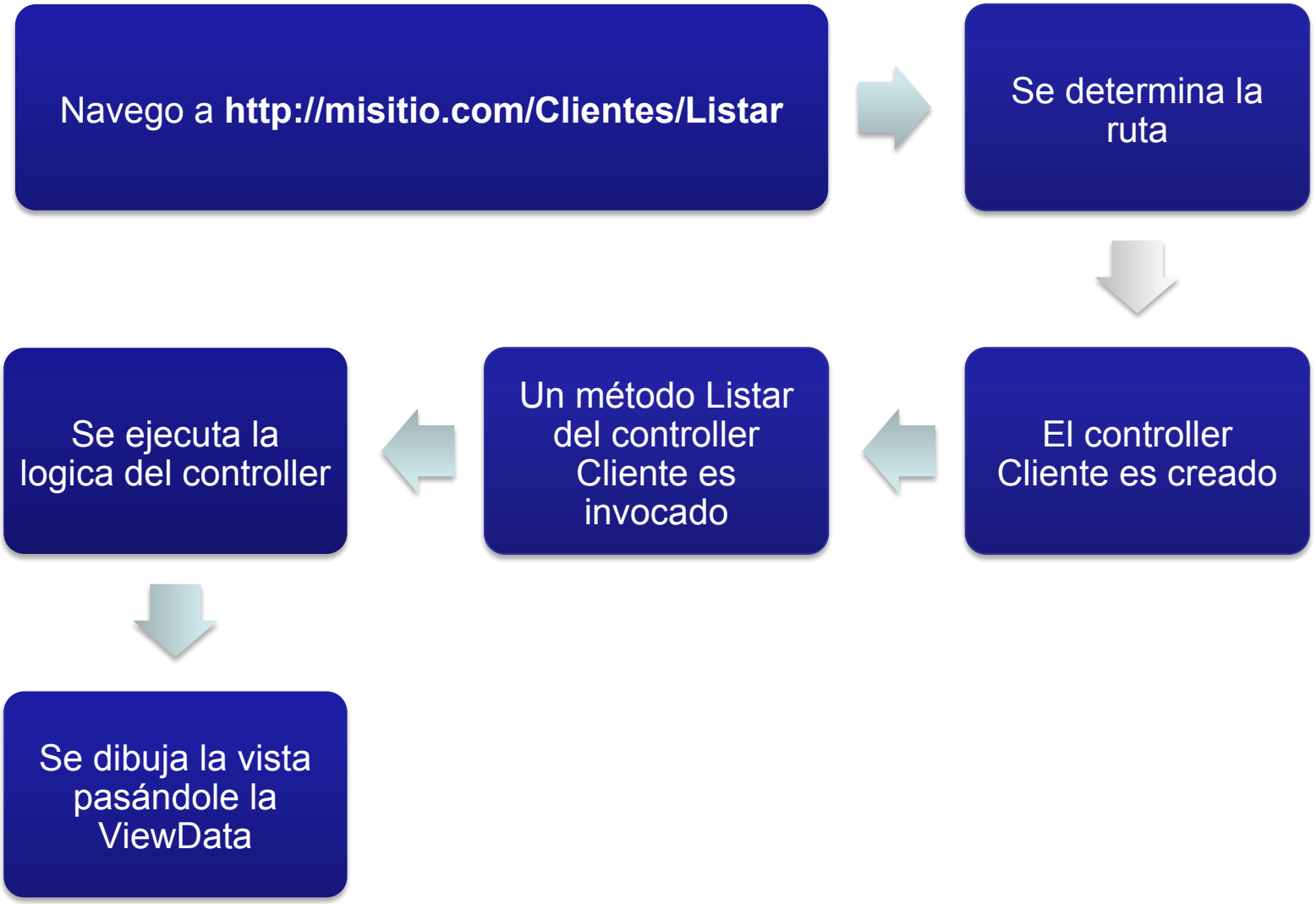
Si el resultado no es una vista, el resultado de la acción se ejecutará por su cuenta (ejemplo un string). Este resultado de ejecución es lo que genera una respuesta real a la solicitud HTTP inicial.

Funcionamiento MVC integrado a la arquitectura



El controller atiende todas las solicitudes (Request), resuelve lógica a través del modelo y devuelve como respuesta una vista html u otros formatos de salida. El modelo puede interactuar con una capa de menor jerarquía, por ejemplo con una capa de servicios o una capa de acceso a datos DAL, como se muestra en la figura.

Como Funciona ??



Introducción ASP.NET MVC

- Un framework para Web Development
- Más control sobre el HTML
  - Más Web-Frendly
- Más testeable
- **Pensado para no ocultar la naturaleza de la arquitectura Web**
- **No es una nueva versión de ASP.NET Web Forms**
- Esta construido sobre en ASP.NET

Ventajas:

- SoC (Separation of Concerns)
  - TDD por default
  - Mantenibilidad
- Url y HTML mas limpio
  - SEO y REST friendly
    - /Alumno/BuscarMateria/pw3
  - CSS Friendly
    - <html> <label>
- Modelo de programación mas performante
  - No hay ViewState
  - No hay modelo de eventos

Rutas

www.misitio.com/products/reports/1/06/2008

```
using System;

public class ProductsController : Controller
{
    public ActionResult Reports(int id, int month, int year)
    {
        //...
        View();
    }
}
```

URLs Amigables

Legibles:

www.sitio.com/products.aspx?module=reports&productId=1&month=6year=2008 => ☹

www.sitio.com/products/report/1/6/2008 => ☺

Predecibles

http://es.**wikipedia**.org/**wiki**/MVC

Enrutamiento

Enrutamiento es el proceso de dirigir una solicitud HTTP hacia un controlador.

Este proceso es llevado a cabo por System.Web.Routing, que no es parte de ASP.NET MVC exclusivamente, sino que es parte de la rutina de Ejecución de ASP.NET que fue lanzada con .Net SP1 3.5

El marco MVC aprovecha la encaminamiento para dirigir una petición a un controlador.

En el archivo Global.asax es donde se definirá la ruta para su aplicación:

```
using System.Web;
using System.Web.Mvc;
using System.Web.Routing;

namespace APPMVC {
   public class MvcApplication : System.Web.HttpApplication {
      protected void Application_Start(){
    RouteConfig.RegisterRoutes(RouteTable.Routes);
      }
   }
}
```

Enrutamiento: RouteConfig.cs

La siguiente es la implementación de la clase RouteConfig, que contiene el método RegisterRoutes en su versión por defecto.

```
namespace MVCFirstApp {
  public class RouteConfig {

    public static void RegisterRoutes(RouteCollection routes){

        routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

        routes.MapRoute(
            name: "Default",
            url: "{controller}/{action}/{id}",
            defaults: new{ controller = "Home", action = "Index", id =
UrlParameter.Optional});
      }
   }
}
```

Enrutamiento: RouteConfig.cs

routes.IgnoreRoute: Evita o ignora rutas especificas. En este caso se evitan rutas que invoquen archivos de recursos web de asp.net.

Si quisiéramos ignorar o evitar solicitudes a un cierto controlador llamado Clientes, deberíamos agregar la siguiente línea en el Metodo RegisterRoutes:

```
routes.IgnoreRoute("Clientes/");
```

routes.MapRoute():

```
routes.MapRoute(
            name: "Default",
            url: "{controller}/{action}/{id}",
            defaults: new{ controller = "Home", action = "Index", id =
UrlParameter.Optional});
```

Define rutas validas por medio del patrón definido en el parámetro **url.** Esas rutas validas existen en una tabla de enrutamiento o routing. Luego esas rutas se mapean a las URLs solicitadas a una acción de un controlador específico. Una acción es sólo un método en el controlador. También puede recoger parámetros de la URL y pasarlos como parámetros en el método.

El código anterior corresponde al patrón de urls validas por defecto. De esta forma cuando llegue una solicitud tipo: **(algo) / (algo) / (algo)**, MVC interpretara el primer "algo" como el nombre del controlador, el 2do "algo" como el nombre de la acción y el 3er algo como el parámetro.

Enrutamiento: RouteConfig.cs

La llamada default anterior, permitirá al motor de enrutamiento aceptar URLs de la forma:
http://www.mitisio.com/Home/Inicio
http://www.mitisio.com/Home
http://www.mitisio.com/Clientes/Todos
http://www.mitisio.com/Clientes/Buscar/1
www.mitisio.com

En cambio la siguiente url NO seria aceptada:
www.misitio.com/products/reports/1/06/2008

Ya que el patrón de url aceptado es /controlador/acción/id (con valores por defecto), y el motor de enrutamiento no encuentra en su tabla una url con ese patrón o características.

Para hacer valida la solicitud anterior se debería agregar explícitamente una nueva entrada en la tabla de enrutamiento para el patrón:

```
routes.MapRoute(
   "Fecha",
   "{controller}/{action}/{dia}/{mes}/{anio}");
```

Controlador Uso Básico

El controller por medio de la acción, no necesariamiente tiene que devolver una vista html.

En el siguiente caso se esta devolviendo un string como respuesta a la solicitud (request): **/TestNoSoloHTML/SimpleString**

```
public class TestNoSoloHTMLController : Controller
{
    public string SimpleString()
    {
        return "Hola Clase, esto no es html";
    }
}
```

Controlador Uso Básico

- Escenarios, Objetivos y Diseño:
  - Las URLs indican "acciones" del Controlador, **NO páginas**
  - Las acciones se declaran en el Controlador.
  - El controlador ejecuta lógica y elige la vista.

```
public ActionResult MostrarCliente(int id)
{
     Cliente c = clienteRepository.GetClienteById(id);
      if (c != null)
      {
         return View(c);
      } else
      {
         return View("noencontrado", id);
      }
}
```

Vistas

- Generan HTML u otro tipo de contenido.
  - Helpers pre-definidos.
- Pueden ser .ASPX, .ASCX, .MASTER, etc.
- Pueden reemplazarse con otras tecnologías:
  - Template engines (NVelocity, Spark, …).
  - Formatos de salida (images, RSS, JSON, …).
  - Pueden definirse **vistas Mock** para testing.
- El controlador ofrece datos a la Vista
  - Datos Loosely typed o Strongly Typed ☺.
- Puedes ser tipadas o No tipadas.

Vistas

Pueden ser Tipadas:

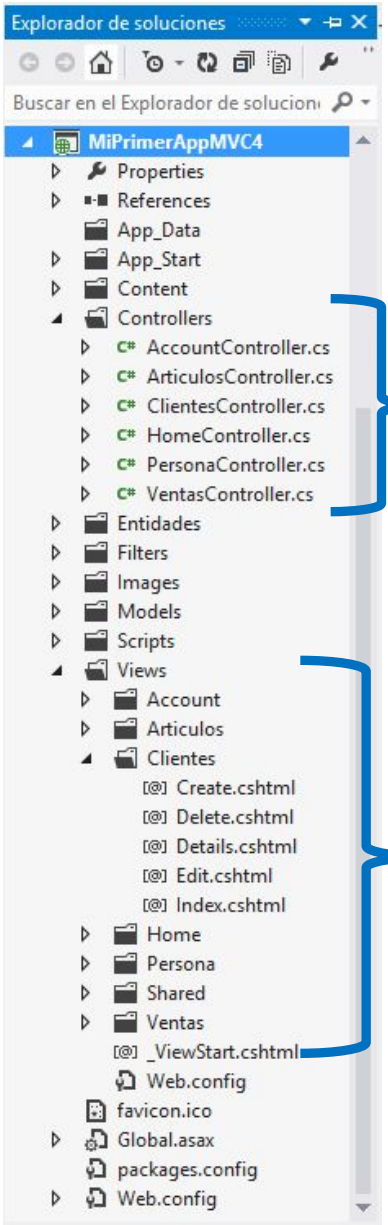
Controller Empleados, en respuesta al request: **/Empleados/Mostrar/10**

```
public ActionResult Mostrar (int id)
{
    Empleado e = empleadoRepository.GetEmpleadoById(id);
    return View(e);
}
```

Vista Mostrar:

```
@model MVCFirstApp.Models.Empleado

@{
    Layout = null;
}

<h2>Empleado: @Model.Nombre @Model.Apellido</h2>
```

Estructura en Visual studio



Controladores

Vistas

Model Binding

Enlace de elementos HTML a propiedades de un objeto (incluyendo su creación)

Model Binding

Por defecto las acciones en el Controlador responden al verbo **GET**, para capturar el verbo **POST**, se debe anteponer en la acción el atributo [**HttpPost**]

```
[HttpPost]
public ActionResult CrearNuevo(Empleado em)
{
    if (ModelState.IsValid)
    {
        empleadoRepository.Grabar(em);
                return RedirectToAction("Lista");
        }
        else
                 return View(em);
 }
```

Binding Manual (1)

En este caso, el objeto del dominio se crea y se carga manualmente a partir del formulario (**FormCollection**) recibido por parámetro o por el objeto **Request**.

```
[HttpPost]
public ActionResult Create(FormCollection form)
{
    Cliente cliente = new Cliente();
    if (ModelState.IsValid)
    {

        cliente.Nombre = form["Nombre"];
        cliente.Direccion = form["Direccion"];

        SaveCliente(cliente);
        return RedirectToAction("Index");
    }

    return View(cliente);
}
```

```
[HttpPost]
public ActionResult Create()
{
    Cliente cliente = new Cliente();
    if (ModelState.IsValid)
    {

        cliente.Nombre =  Request["Nombre"];
        cliente.Direccion = Request["Direccion"];

        SaveCliente(cliente);
        return RedirectToAction("Index");
    }

    return View(cliente);
}
```

Binding Manual (2)

Para ello se deben respetar los valores del atributo **name** de cada input del formulario.

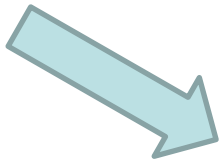
```
@{ Layout = null; }
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width" />
<title>Crear Cliente</title>
</head>
<body>
<div>
<form action="/Clientes/Create" method="post">
    Nombre: <input type="text" id="txtNombre" name="Nombre" value="" /><br />
    Dirección: <input type="text" id="txtDireccion" name="Direccion" value="" /><br />

    <input type="submit" name="BtnSave" value="Grabar Cliente" />
</form>
</div>
</body>
</html>
```

Binding Automático (1)

A diferencia del manual, el objeto del dominio ya viene cargado automáticamente.

```
[HttpPost]
public ActionResult CrearNuevo(Empleado em)
{
    if (ModelState.IsValid)
    {
        empleadoRepository.Grabar(em);
                return RedirectToAction("Lista");
        }
        else
                 return View(em);
 }
```

Binding Automático (2)



El objeto de la acción que atiende o responde la solicitud del submit de la vista, es **mapeado automáticamente** con los controles a través del atributo **name** de los mismos. De esta forma, para que el mapeo sea correcto, cada nombre de la propiedad de la clase debe coincidir con el nombre del atributo o propiedad **name** de cada control.

```
namespace MVCFirstApp.Models
{
    public class Empleado
    {
        public string Nombre { get; set; }
        public string Apellido { get; set; }
        public int Edad { get; set; }
        public double Sueldo { get; set; }
    }
}
```

```
@{ Layout = null; }
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width" />
<title>Crear Empleado</title>
</head>
<body>
<div>
<form action="/Empleados/CrearNuevo" method="post">
       Nombre: <input type="text" id="txtNombre" name="Nombre" value="" /><br />
       Apellido: <input type="text" id="txtApellido" name="Apellido" value="" /><br />
       Sueldo: <input type="text" id="txtSueldo" name="Sueldo" value="" /><br />
       Edad: <input type="text" id="txtEdad" name="Edad" value="" /><br />

       <input type="submit" name="BtnSave" value="Grabar Empleado" />
</form>
</div>
</body>
</html>
```

Y donde esta la M en ASP.NET MVC ?

ASP.NET MVC no provee una infraestructura en particular obligatoria para el modelo pero existen una gran cantidad de opciones.

- Business Layer / Repositories
- Entity Framework (Business - DAL)
- Nhibernate (Business - DAL)
- Subsonic
- L2SQL

y muchas otras……

Desarrollador de Aplicaciones Web
Programación Web III

Departamento de Ingeniería e Investigaciones Tecnológicas

Muchas gracias

Ing. Mariano Juiz
Ing. Matias Paz Wasiuchnik
Ing. Pablo Nicolás Sanchez