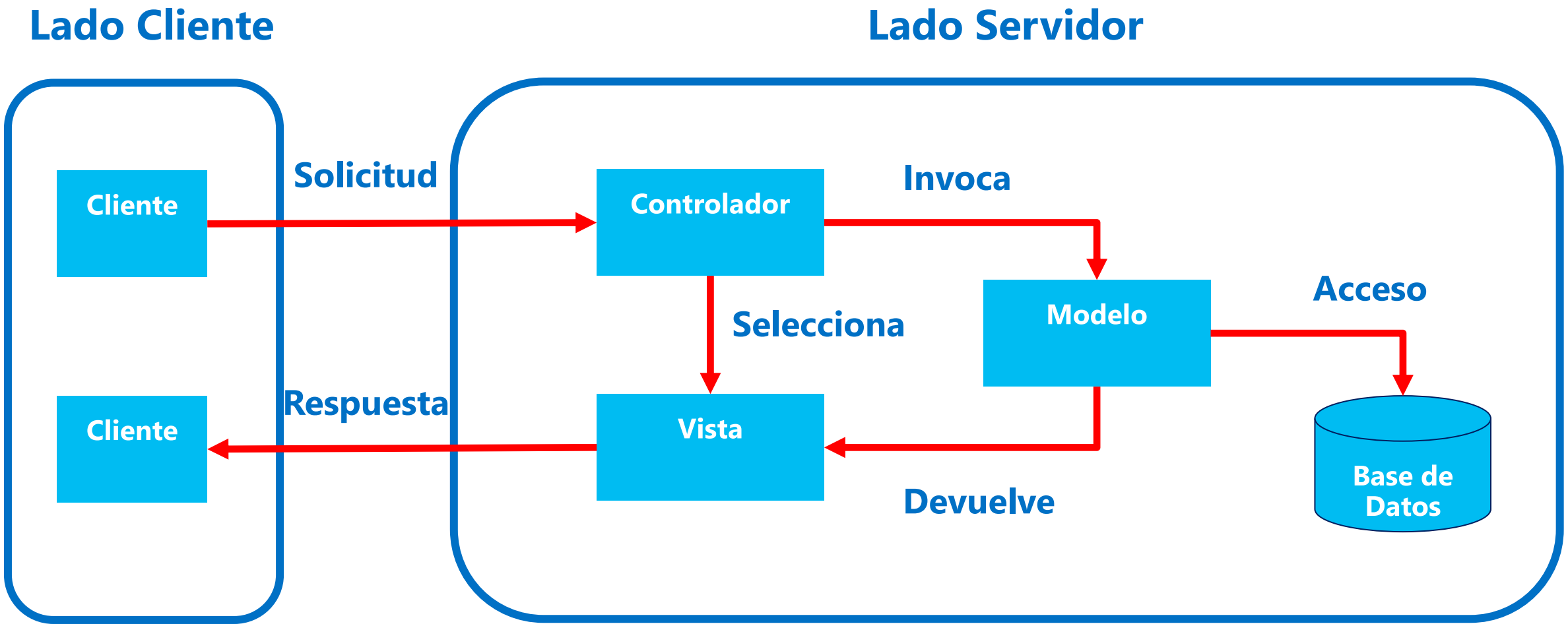
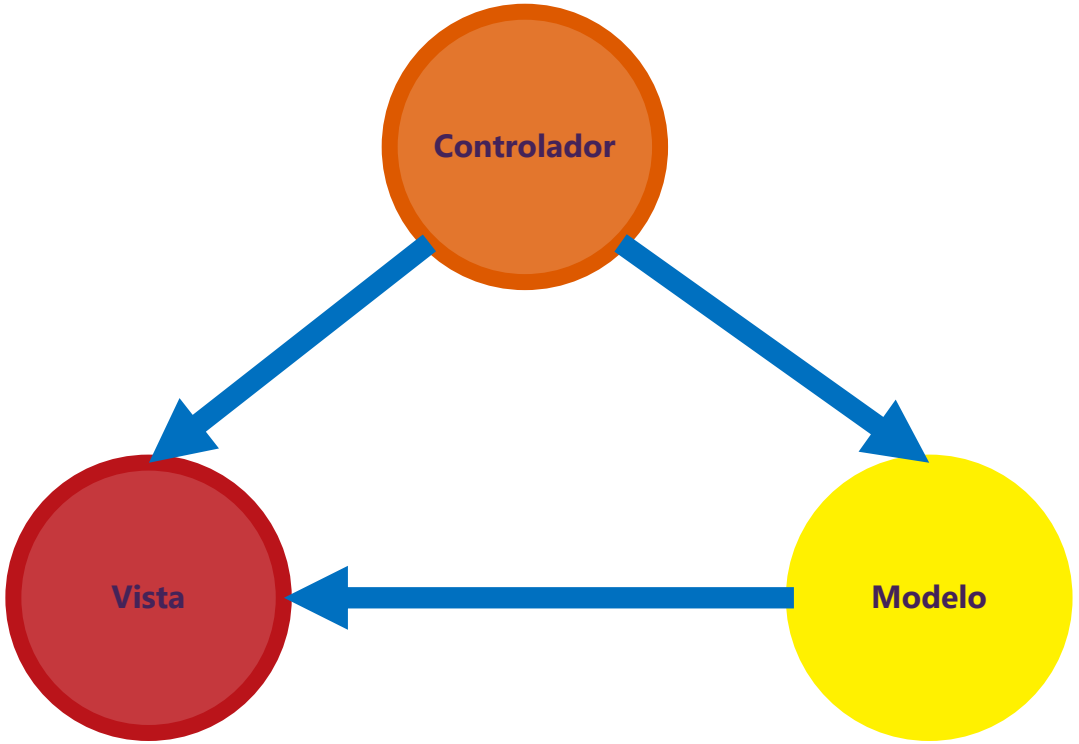
Sección 3. Arquitectura MVC

Patrón de Diseño MVC

Patrón MVC


Lado Cliente
Lado Servidor
Cliente
Cliente
Solicitud
Respuesta
Controlador
Vista
Selecciona
Invoca
Modelo
Devuelve
Acceso
Base de Datos

Modelo-Vista-Controlador (MVC)



Controlador

Interactúa con el Modelo y la Vista

Modelo

Provee datos y asocia la lógica a la vista

Vista

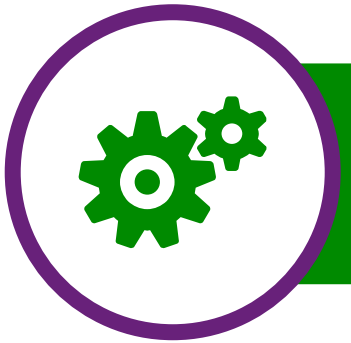
Renderiza el modelo a la vista

Responsabilidades de la Vista

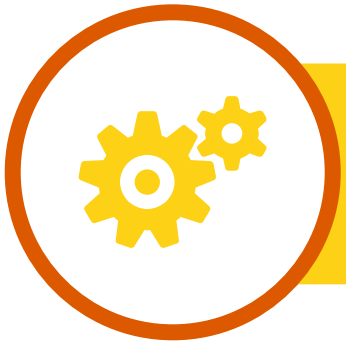
Responsable de interactuar con el Usuario

Renderiza el modelo al usuario

Recoge la interacción del usuario y la envía al controlador

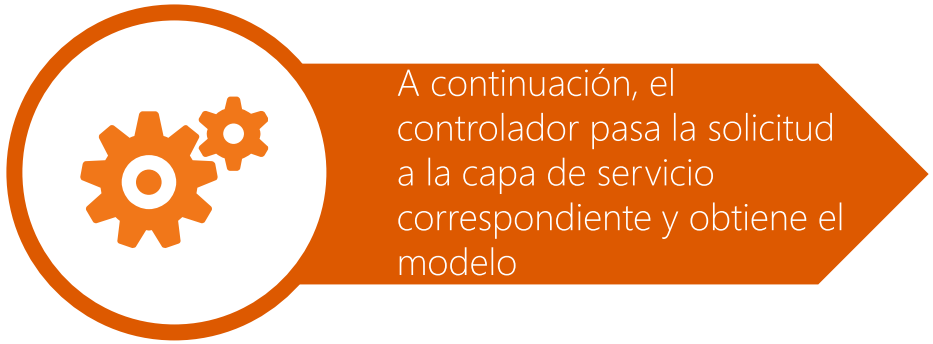


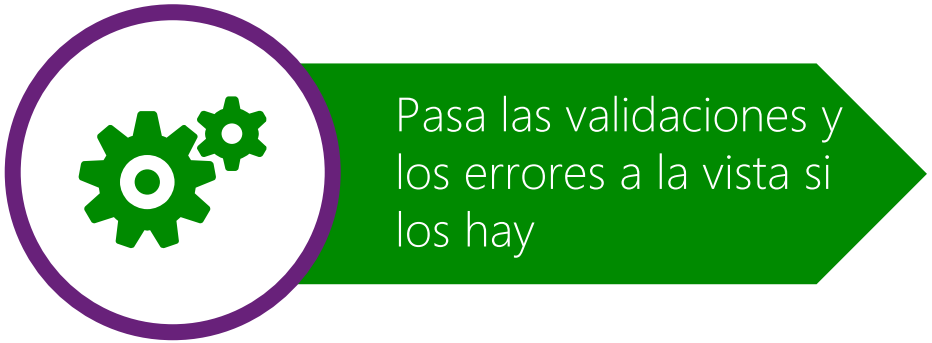
Consta de páginas HTML estándar, Javascript y CSS



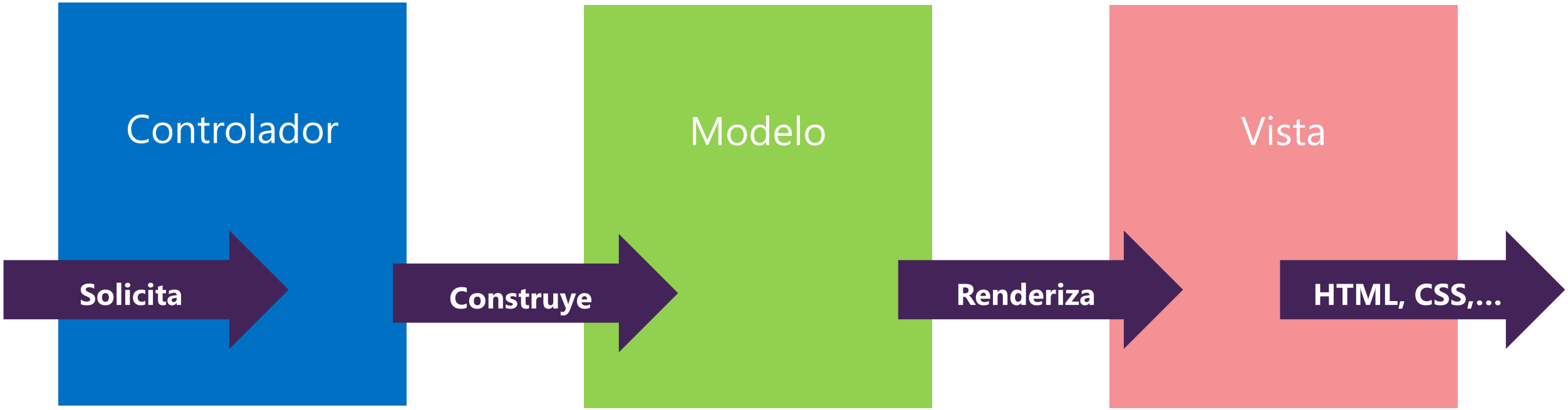
Puede renderizar Json, XML y tipos de retorno personalizados

Responsabilidades del Controlador





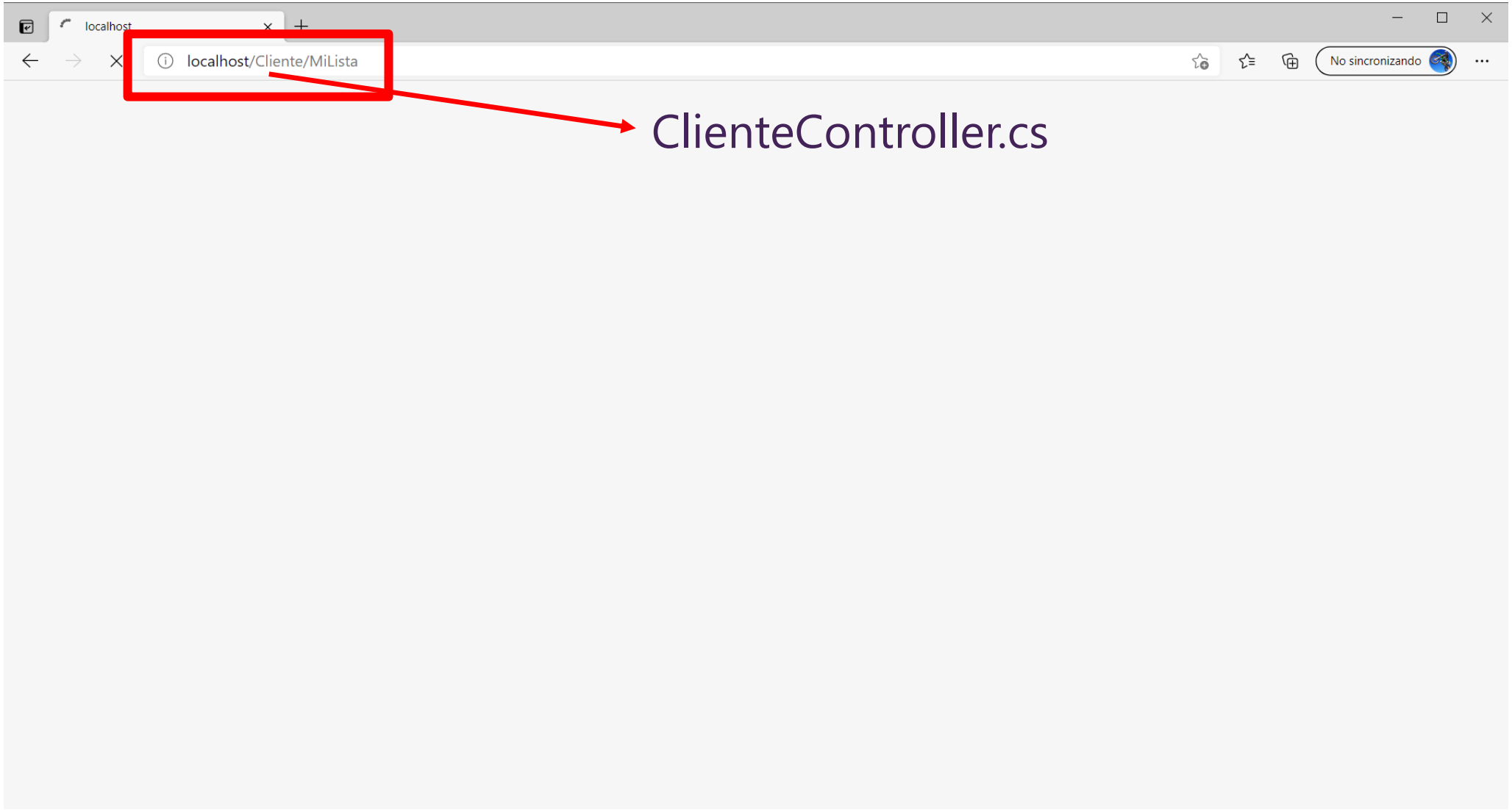
Componentes MVC



Sección 3. Arquitectura MVC

El Controlador

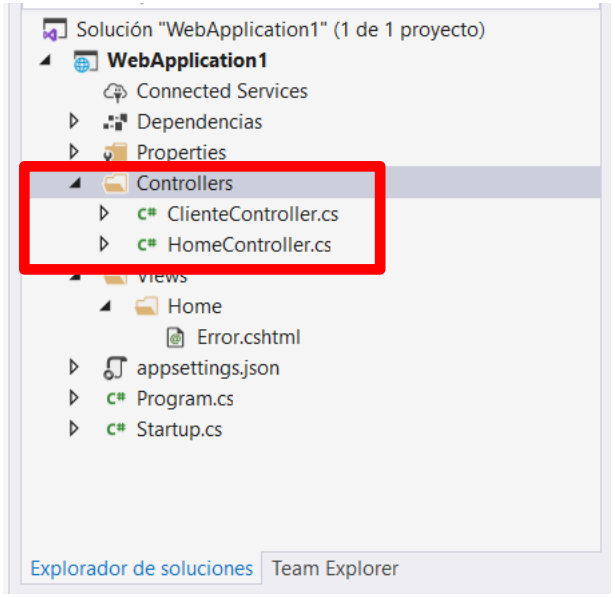
El Controlador

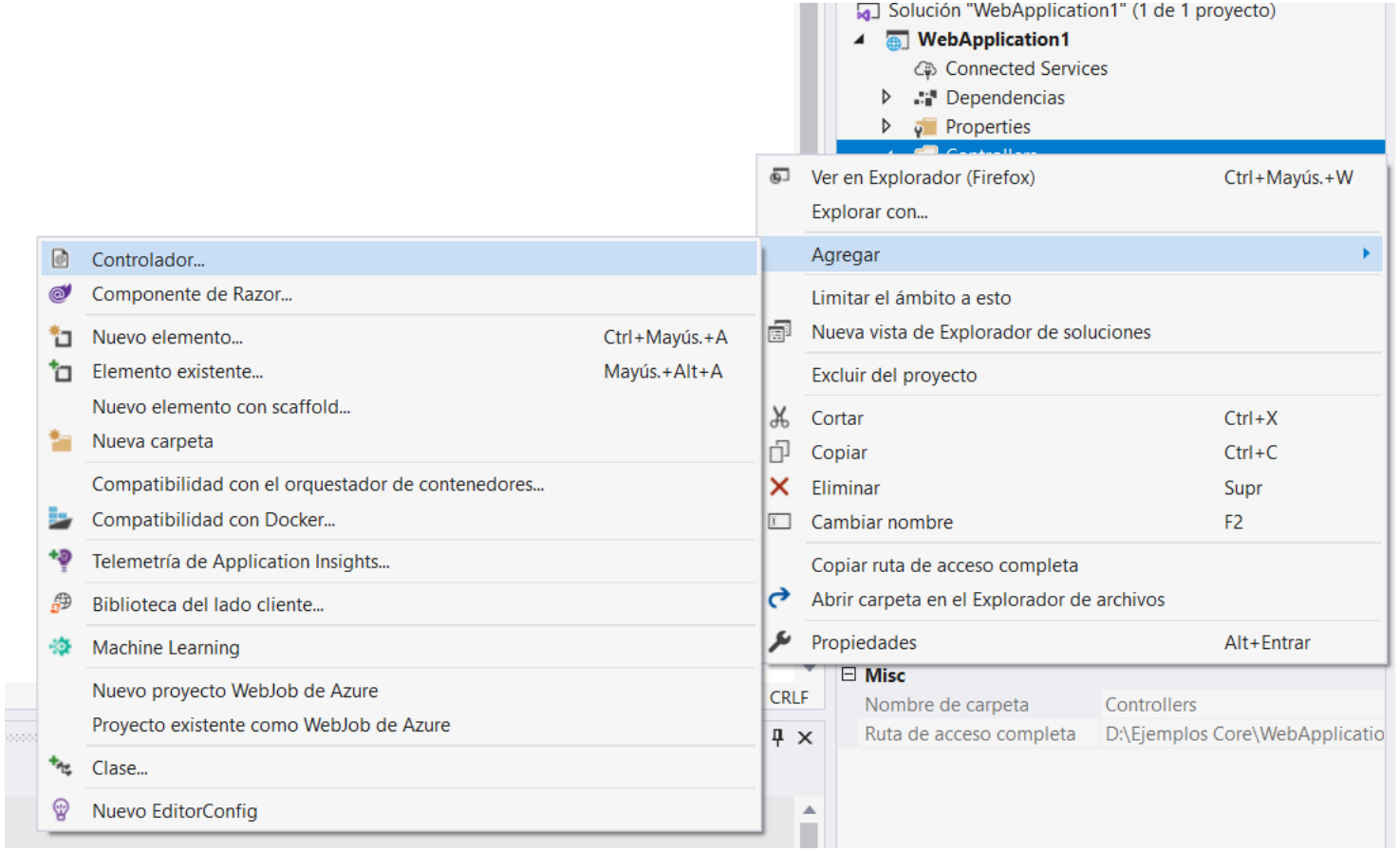


Responsabilidades

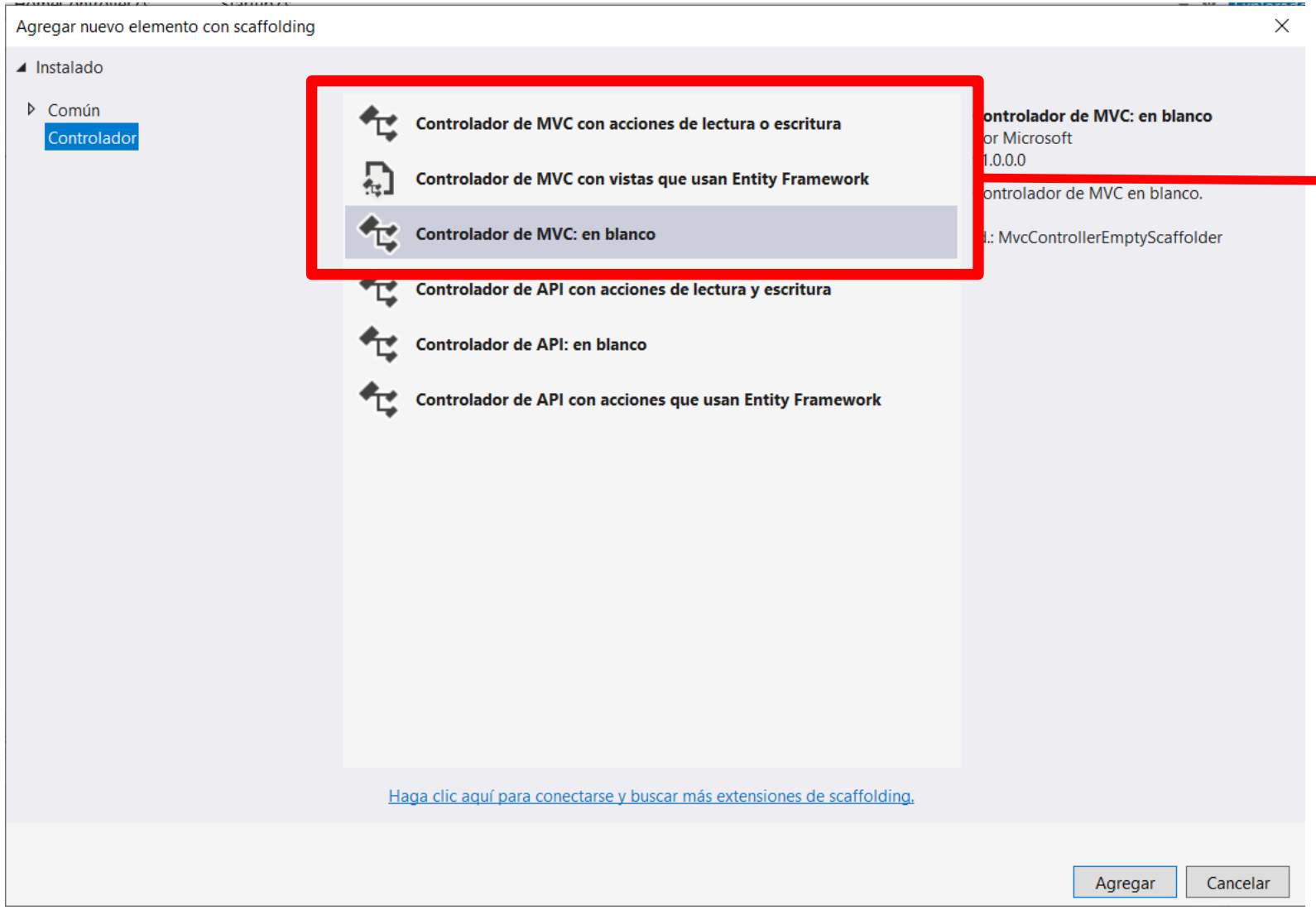
Manejar las solicitudes	El controlador es responsable de manejar las solicitudes del usuario
Construir un modelo	El método de acción Controller ejecuta la lógica de la aplicación y crea un modelo
Enviar las respuestas	Devuelve el Resultado en HTML//Json/XML/ Archivos o en cualquier formato solicitado por el usuario

Controllers y Agregar Controlador





Controllers y Agregar Controlador



Clase Controller

Clase ControllerBase

Condiciones

Deben de satisfacer al menos una

- El nombre de la clase tiene el sufijo "Controller"
- La clase hereda de una clase cuyo nombre tiene el sufijo "Controller"
- La clase tiene el atributo [Controller]

Métodos de Acción

- Métodos de acción Debe ser un método público
- El método de acción no puede ser un método estático ni un método de extensión.
- La clase hereda de una clase cuyo nombre tiene el sufijo "**Controller**"
- El Constructor, **getter** o **setter** no se pueden utilizar.
- Los métodos heredados no se pueden utilizar como método de acción.
- Métodos de acción No pueden contener parámetros **ref** ni **out**.
- Los métodos de acción no pueden contener el atributo **[NonAction].**
- Los métodos de acción no se pueden sobrecargar

Enrutamiento

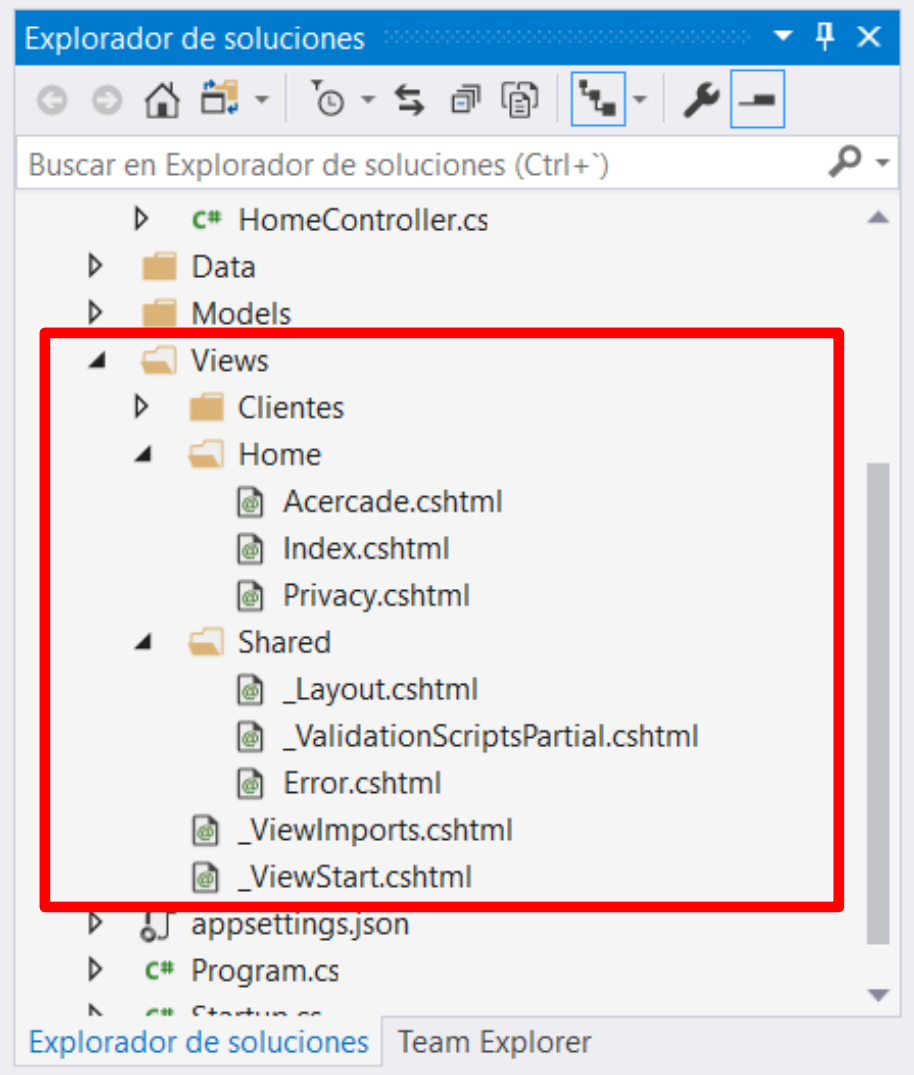
Action Result

ViewResult	Representa HTML y marcado.
EmptyResult	No representa ningún resultado.
RedirectResult	Representa una redirección a una nueva URL.
JsonResult	Representa un resultado de notación de objetos JavaScript que se puede utilizar en una aplicación AJAX.
JavaScriptResult	Representa un script de JavaScript.
ContentResult	Representa un resultado de texto.
FileContentResult	Representa un archivo descargable (con el contenido binario).
FilePathResult	Representa un archivo descargable (con una ruta).
FileStreamResult	Representa un archivo descargable (con una secuencia de archivos).

Sección 3. Arquitectura MVC

El Modelo

El Modelo



- **HTML, JSON, XML o cualquier otro**
- **formato que el usuario consuma**

- **Extensión .cshtml**

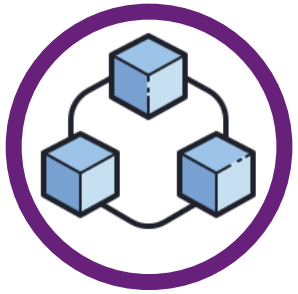
- **\Views\<NombreControlador>\<MetodoAccion>.cshtml**

- **\Views\Shared**

Sección 3. Arquitectura MVC

El Modelo

El Modelo

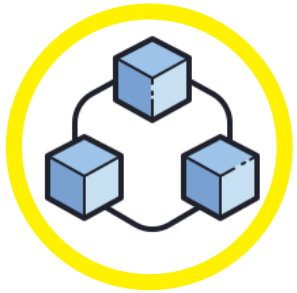


Modelo de Dominio (Model Domain)

Representa el objeto que representa los datos en la base de datos. El modelo de dominio generalmente tiene una relación uno a uno con las tablas de la base de datos. El modelo de dominio está relacionado con la capa de acceso a datos de nuestra aplicación.

ViewModel

Relacionado con la capa de presentación de nuestra aplicación. Se definen en función de cómo se presentan los datos al usuario en lugar de como se almacenan.



Modelo de Edición (Edit Model)

Representa los datos que deben presentarse al usuario para su modificación y/o inserción. Los requisitos de la interfaz del usuario del producto para editar pueden ser diferentes del modelo requerido para la visualización.

Modelo de Dominio (Model Domain)

```
public class Producto
{
    public int ProductoId { get; set; }
    public string Nombre { get; set; }

    public Decimal Precio { get; set; }
    public int Valoracion { get; set; }

    public Marca Marca { get; set; }
    public Proveedor Proveedor { get; set; }
}
```

ViewModel (Modelo de Vista)

```
public class ProductoViewModel
{
    public int ProductoId { get; set; }
    public string Nombre { get; set; }

    public Decimal Precio { get; set; }
    public int Valoracion { get; set; }

    public string NombreMarca { get; set; }
    public string NombreProveedor { get; set; }

    public string getValoracion()
    {
        if (Valoracion == 5)
        {
            return "*****";
        }
        else if (Valoracion >= 4)
        {
            return "****";
        }
      //…
    }
}
```

Modelo de Edición (Edit Model)

```
public class ProductoEditModel
{
    public int ProducotId { get; set; }

    [Required(ErrorMessage = "El nombre del producto es obligatorio")]
    [Display(Name = "Nombre Producto")]
    public string Nombre { get; set; }

    public Decimal Precio { get; set; }
    public int Valoracion { get; set; }

    public List<Marca> Marcas { get; set; }
    public List<Proveedor> Proveedores { get; set; }

    public int MarcaID { get; set; }
    public int ProveedorID { get; set; }
}
```

Modelo VS ViewModelo

Modelo

Representa un objeto del mundo real que está relacionado con el problema al que queremos dar solución

Estas clases, que son conocidas como modelos, tienen algunas propiedades y métodos

En ORM (Object Relational Mapper) como es Entity Framework, un modelo está estrechamente vinculado a una entidad

ViewModel

Una vista tiene la responsabilidad de representar los datos que normalmente provienen de un objeto

El ViewModel No es un modelo de dominio sino un modelo de vista porque, solamente lo va a usar una vista específica

No Representa a un objeto del mundo real y además, el ViewModel es inútil sin la Vista