MVC

# Es un Patrón de diseño

# En nuestro modelo MVC no hace falta crear constructores

# Modelos :

Representan los datos de la aplicación , las clases de modelo usan lógica de validación para aplicar las reglas de negocio para esos datos. Los objetos de modelo recuperan y almacenan el estado del modelo en una base de datos. En nuestros proyectos los modelos recuperan los datos de sus sistemas de una base de datos , estos se proporcionaran a las vistas o los actualizaran. Los datos actualizados se escribirán nuevamente en la base de datos.

# Vistas:

Las vistas son los componentes que muestran la interfaz de usuario (UI) de la aplicación. Por lo gral esta interfaz de usuario muestra los datos del modelo.

# Controladores:

Son clases que :

* Controlan las solicitudes del explorador (Request - Response)
* Recuperan datos del modelo (Por medio de un contexto de datos por ejemplo)
* Llaman a las plantillas de vista que devuelven una respuesta (Método View)

# Guid : Global Uquique Identifier

Tiene la posibilidad de generar un identificador único (en el mundo) y que no se repita, en este proyecto se suele usar Int como id por que es mas fácil de identificar : ?

# Id : identifier

El id es un identificador único e irrepetible , nosotros por ejemplo los usamos para definir por ejemplo personas con un identificador que no se repita.

Nosotros vamos a usar el identificador como entero por una cuestión de simpleza y performance. Ya que es el tipo de dato mas rápido.

# List<T valor> : Valor genérico de lista ?

Es una colección lista que principalmente vamos a usar, hay mas como Icolection y demás..

# {get; set;}

Es un tipo de propiedad de acceso publico que nos brinca C#

CUALQUIER propiedad que NO tenga un SET Entity framework la va a IGNORAR

# “Entidad de fuerza”:

# Es la entidad que tiene mas peso en el sistema con respecto a otra entidad con la que se relaciona, se le dice así por que tiene mas peso

Data Notations : se aplican a la propiedad que se encuentra debajo

algunas tienen implicancia en el cliente y otras en el cliente y el servidor.

# Required : Significa que el campo no puede ser null (validación del lado del cliente y del servidor)… IMPORTANTE el required hace eliminacion en cascada al momento de borrar una persona, ya que borra los datos que pertenecen a esa persona

# Propiedades navegacionales :

Son propiedades que me van a permitir acceder a los datos de otras entidades que están relacionadas, no importa si es 1 a 1 o N a M . No es obligatoria pero como ayuda.

Posible pregunta de final :  
CICLO DE VIDA DE UNA APLICACIÓN WEB MVC

* 1. HTTP REQUEST (se dispara una solicitud web)
* 2. Hay un proceso de ruteo
* 3. Luego pasa al MVC HANDLER (Manejador MVC)
* 3.1 Creacion de un Controlador para responder las peticiones que se hacen
* 3.2 Autenticacion y Autorizacion (Identity buscarlo despues)
* 3.3 Model binding (Buscar mas adelante, pero habla de que hay un entendimiento de lo que me manda el usuario , y de lo que estoy en condiciones de recibir)
* 3.4 Invocación de Action methods (que forman parte del controlador, por ejemplo devolver una vista
* 3.5 Dar un resultado de la ejecución
* 4. HTTP RESPONSEGráfico

  Descripción generada automáticamente con confianza baja

Action methods

Son los Controllers de MVC

# HomeController : Controller

Home controller hereda de controller por ende tiene los mismos atributos y métodos mas sus propias especializaciones.

# IActionResult(Interfaz) :

Por ejemplo uno es Index y retorna una vista , justamente la vista Index , por que hace referencia a la vista de Index

Return View() : no retorna una vista, ejecuta el método View() y devuelve un ViewResult que viene de Controller.ViewInterfaz de usuario gráfica, Texto, Aplicación, Sitio web

Descripción generada automáticamentedemostración de envio de parámetros al método de la vista desde un método iactionResult del controlador

# Tipo :ViewResult View(string viewName) : nombre de la vista

View( string viewName , object modelo) : le paso el nombre de la vista y un objeto

# Views:

El return del método anterior ViewResult View() , llamado en los controller bajo la firma de view() hace referencia a la carpeta Views donde ahí se encuentran efectivamente las vistas a retornar

 (como muestra la imagen de arriba , se declara de esta manera el parámetro mandado)

# Entity Framewor , ofrece el Engine Razor

Este nos facilita implementar código de CSharp en las Vistas , por eso tienen una extension CSHTML para incluir el código del mismo en un Lenguaje de Maquetacion Web.

@{

Bloque de Código CSharp

}

@LlamadaVariableCSharp(Ya declarado en un bloque de código anterior)

ViewBag y ViewData

ViewBag : TBC

# ViewData: TBC

MVC.TagHelpers

Si empiezan con el prefijo ASP son tag helpers que nos van a ayudar a adecuar, y manipular los elementos HTML

Para que sirven? Sirven para simplificar y clasificar de forma simple las redirecciones?

Asp-controller = “home” : es un tag helper para redireccionar a otras vistas

Asp-Action = “Privacy” : Te redirige a un subdirectorio de “home”(?

[HttpGet] : Se puede usar para enviar o recibir datos no hay problema , el problema es que se envían por query String esto significa que es como si fuera un String de búsqueda que aparece al lado de la Url , por que NO usarlo para búsquedas, por que expone datos sensibles.

[HttpPost] : Es para enviar datos

Tag Helpers

ASP . Net Framework nos facilita la vida con la utilización de tag helpers , hay diversos tag helpers específicos para cada tipo de Tag estos nos facilitan la lectura gracias a que mejoran la semántica, además hacen referencia directa a los elementos del modelo y soportan los tipos de dato de cada atributo del modelo.

Asp-for : especifica para que propiedad va a ser usado y además sirve para especificar el tipo de dato del atributo del modelo.

Html tag helpers , son mas poderosos que los tag helpers de ASP. Net.

Entity Framework

# Clases Entity Framework:

 : La clase mas usada de entity Framework. Es de la cual nosotros heredamos el contexto de base de datos y el que usamos para gestionar las bases de datos , ya que nuestro contexto hereda/implementa este mismo

 : Esto es una colección de personas, similar a una List<Persona>. Es un set de base de datos.

# Constructor :



El constructor de nuestro contexto de base de datos recibe unas opciones de contexto de base de datos , pero para poder pasar estas opciones de configuración al DbContext se necesita referenciar a la clase padre de esta manera :



Inyección De Dependencias

Texto

Descripción generada automáticamente

Cada vez que se instancie un objeto persona (Create) voy a traerme de mi contexto de base de datos en el estado en el que se encuentre y lo guardo en una variable para poder usarlo posteriormente, esto me permite hacer modificaciones al modelo de base de datos en un solo lugar. Esto aplica tambien a todo CRUD.

. Net Framework Scaffolding

Tipos de Scaffolding :

Code first : lo que nosotros hacemos , primero el código y nos genera la base de datos y modelos

Database First : me baso en una base de datos ya existente y se generan modelos en base a las tablas de la base.

Model First : Actualmente en desuso.