**PortalEDU Seguimiento de implementación**

Crear nuevo proyecto Biblioteca Core

**- MOdels**

**- ACCeso a Datos**

**- Utilidades**

**INstalar Ngets**

**- MVC.RAzor.RuntimeCompilation (para guardar cambios y compilarlos en tiempo real)**

**- Microsoft.ASPnetcore.MVC para proyectos ADO y MOdels**

**- ASPnetcore.MVC.NewtonSoftjson para el proyecto principal**

**- Microsoft.Entityframeworkcore para ADO**

**- Microsoft.Extensions.Identity.Stores para ADO y models**

**- Microsoft.Aspnetcore.Identity.EntityframeworkCore para ADO**

**- Microsoft.EntityframworkCore.sqlserver para ADO**

Modificaion de clases (Esto es para que se guarden los cambios en vivo en el HTML “RuntimeCompilation”)

- En starup --> services.AddControllersWithViews().AddRazorRuntimeCompilation();

**Crear Areas para permisos**

* **Clic derecho en áreas y crearlas MVC**

**- Cambiar en el startup el endpoints**

**"{area=Anonimo}/{controller=Home}/{action=Index}/{id?}");**

**- Eliminar sus modelos y data porque usaremos los creados en biblioteca aspnetcore Models y ADO**

**- Mover "la clase" controllers global dentro de la carpeta controller de usuario anonimo y eliminamos la carpeta donde estaba**

**- Copiar ViewImport y viewstart dentro de los views de las areas. cambiar la ruta @{**

**Layout = "~/Views/Shared/\_Layout.cshtml";**

**}**

**- Setear home controller con [Area("Anonimo")]**

**- LLevarse clase errorviewmodels del proyewcto general al proyecto models creado con la biblioteca luego borrar models general**

**- Eliminamos las class1.cs creadas con las bibliotecas**

**- Arrastrar carpeta data a ADO luego elimarla**

**- Cambiar el namespace de applications dbcontext para usar el ADO**

**- Eliminar las migraciones existentes**

**- Mover carpeta Migrations a ADO**

**Crear carpetas dentro de DATA**

**- Inicalizador**

**- Repository**

**- Arrastrar carpeta home del view general para la carpeta view del usuario anonimo**

**Luego debemos de crar referencias en el pryecto general del proyecto ADO**

**Video 29**

**- dependencias del proyecto principasWEB agregarle Acceso a datos ADO**

**- Dendro de las dependencias acceso a datos de ADO agregar modelos y utilidades**

**OJO asegurarse que todos los paquetes estén actualizados en todos los proyectos**

Configurtacion del frontEnd

**Video 30**

**bajar css de** [**https://bootswatch.com/**](https://bootswatch.com/) **copiar el full no el mini y pegarlo en el Bootstrap original**

**Modificar navbar**

**Para que tome los cambios fijarse que se este importando el Bootstrap adecuado en el head**

Modificar la nav:

<a class="navbar-brand" **asp-area**="Anonimo" **asp-controller**="Home" **asp-action**="Index">Navbar</a>

Crear menú con estas características:

<a class="nav-link" **asp-area**="Admin" **asp-controller**="Configuracion" **asp-action**="Index">Configuracion</a>

* Cortar el login partial del origina y pegarlo en el div penultimo

video 31

Instalacion de plugin para frontend Instalar después de Bootstrap del head

<!--Css todos-->

<link rel="stylesheet" href="https://cdn.datatables.net/1.10.16/css/jquery.dataTables.min.css" />

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/jqueryui/1.12.1/jquery-ui.min.css" />

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/css/toastr.min.css" />

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/sweetalert/1.1.3/sweetalert.min.css" />

Instalar en la parte de plugins

<!--Js plugins-->

<script src="https://cdnjs.cloudflare.com/ajax/libs/jqueryui/1.12.1/jquery-ui.min.js"></script>

<script src="https://cdn.datatables.net/1.10.16/js/jquery.dataTables.min.js"></script>

<script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/jquery.serializeJSON/2.9.0/jquery.serializejson.min.js"></script>

<script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/js/toastr.min.js"></script>

<script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/sweetalert/1.1.3/sweetalert.min.js"></script>

<script src="https://kit.fontawesome.com/7588f654f8.js" crossorigin="anonymous"></script>

Inicia Repository

En el proyecto ADO carpeta data - repository agregar una interface (PUBLIC) IRepository donde irean los metodos globales

public interface IRepository<T> where T : class

Importar system.linq.expression

CLASE GLOBAL IREPOSITORY

{

T Get(int id);

IEnumerable<T> GetAll(

Expression<Func<T, bool>> Filter = null,

Func<IQueryable<T>, IOrderedQueryable<T>> orderBy = null,

string includeProperties = null

);

T GetFirstOrDefault(

Expression<Func<T, bool>> Filter = null,

string includeProperties = null

);

void Add(T entity);

void Remove(int id);

void Remove(T entity);

}

}

DENTRO de data crear una clase PUBLICA llamada Repository que debe de heredar de IRerpository

- IMplementar los metodos de la interfaz

- Agregar variable de solo lectura y modelar la Clase Repository

using Microsoft.EntityFrameworkCore;

using PortalEDU.ADO.Data.Repository;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Linq.Expressions;

using System.Text;

namespace PortalEDU.ADO.Data

{

public class Repository<T> : IRepository<T> where T : class

{

protected readonly DbContext Context;

internal DbSet<T> dbSet;

public Repository(DbContext context)

{

Context = context;

this.dbSet = context.Set<T>();

}

public void Add(T entity)

{

dbSet.Add(entity);

}

public T Get(int id)

{

return dbSet.Find(id);

}

public IEnumerable<T> GetAll(Expression<Func<T, bool>> filter = null, Func<IQueryable<T>, IOrderedQueryable<T>> orderBy = null, string includeProperties = null)

{

IQueryable<T> query = dbSet;

if (filter != null)

{

query = query.Where(filter);

}

// include property separadas por coma

if (includeProperties != null)

{

foreach (var includeProperty in includeProperties.Split

(new char[] { ',' }, StringSplitOptions.RemoveEmptyEntries))

{

query = query.Include(includeProperty);

}

}

if (orderBy != null)

{

return orderBy(query).ToList();

}

return query.ToList();

}

public T GetFirstOrDefault(Expression<Func<T, bool>> filter = null, string includeProperties = null)

{

IQueryable<T> query = dbSet;

if (filter != null)

{

query = query.Where(filter);

}

// include property separadas por coma

if (includeProperties != null)

{

foreach (var includeProperty in includeProperties.Split

(new char[] { ',' }, StringSplitOptions.RemoveEmptyEntries))

{

query = query.Include(includeProperty);

}

}

return query.FirstOrDefault();

}

public void Remove(int id)

{

T entityToRemove = dbSet.Find(id);

Remove(entityToRemove);

}

public void Remove(T entity)

{

dbSet.Remove(entity);

}

}

}

EMpezamos agregar las interfaces de las entidades (Tablas)

Crear el modelo Categoria en el proyecto Models con la entidades o atributos de la tabla

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Text;

namespace PortalEDU.Models

{

public class Categoria

{

[Key]

public int Id { get; set; }

[Required(ErrorMessage ="Ingrese un nombre para la categoria")]

[Display (Name ="Nombre Categoria")]

public string Nombre { get; set; }

[Required]

[Display(Name = "Orden de Vizualizacion")]

public string Orden { get; set; }

}

}

Creamos una interfaz dentro de repositry

ICategoriaRepository le damos atributo PUBLICA heredamos de IRepository y le pasamos la clase Categoria

Luego Listamos sus metodos

using Microsoft.AspNetCore.Mvc.Rendering;

using PortalEDU.Models;

using System;

using System.Collections.Generic;

using System.Text;

namespace PortalEDU.ADO.Data.Repository

{

public interface ICategoriaRepository : IRepository<Categoria>

{

IEnumerable<SelectListItem> GetListaCategoria();

void Update(Categoria categoria);

}

}

OJO: en la clase Application DBContext debemos de mapear todas las entidades, Ejemplo:

public DbSet<Categoria> Categoria { set; get; } // Asi todas las tablas de la base de datos.

Creamos una clase en carpeta Data de ADO con el nombre de CategoriaRepository para Instanciar la Interfaz

using Microsoft.AspNetCore.Mvc.Rendering;

using PortalEDU.ADO.Data.Repository;

using PortalEDU.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace PortalEDU.ADO.Data

{

public class CategoriaRepository : Repository<Categoria>, ICategoriaRepository

{

private readonly ApplicationDbContext \_db;

public CategoriaRepository(ApplicationDbContext db):base (db)

{

\_db = db;

}

public IEnumerable<SelectListItem> GetListaCategoria()

{

return \_db.Categoria.Select(i => new SelectListItem()

{

Text= i.Nombre,

Value= i.Id.ToString()

});

}

public void Update(Categoria categoria)

{

var ObjDesdeDb = \_db.Categoria.FirstOrDefault(s => s.Id == categoria.Id);

ObjDesdeDb.Nombre = categoria.Nombre;

ObjDesdeDb.Orden = categoria.Orden;

\_db.SaveChanges();

}

}

}

ES EL MOMENTO DE CREAR EL CONTENEDOR DE TRABAJO aqui se crearan todos los repositorios de trabajo EJEMPLO:

ICategoriaRepository Categoria { get; } // Y asi crear para todos los repositorios de las entidades creadas

Asi que esta ira creciendo a medida que crece nuestro proyecto

En IRepository Crear la interfaz IContenedorTrabajo

using System;

using System.Collections.Generic;

using System.Text;

namespace PortalEDU.ADO.Data.Repository

{

interface IContenedorTrabajo : IDisposable

{

ICategoriaRepository Categoria { get; }

void Save();

}

}

LUEGO EN DATA CREAMOS UNA CLASE DEL TIPO ContenedorTrabnajo

Aqui se van a crear el acceso a la interfaz y debe de ir creciendo junto a las nuevas entidades, aqui solo se creo categoria:

using PortalEDU.ADO.Data.Repository;

using System;

using System.Collections.Generic;

using System.Text;

namespace PortalEDU.ADO.Data

{

public class ContenedorTrabajo : IContenedorTrabajo

{

private readonly ApplicationDbContext \_db;

public ContenedorTrabajo(ApplicationDbContext db)

{

\_db = db;

Categoria = new CategoriaRepository(\_db);

}

public ICategoriaRepository Categoria { get; private set; }

public void Dispose()

{

\_db.Dispose();

}

public void Save()

{

\_db.SaveChanges();

}

}

}

----------AQUI TERMINA MODELADO DE LA ESTRUCTURA IREPOSITORYPATERN, SOLO SE DEBE DE IR MODIFICANDO A MEDIDA QUE

VAMOS CREANDO NUEVAS ENTIDADES--------

PERMANECEN

IRepository

Repository

Y en las unidades de trabajo IContenedorTrabajo y ContenedorTrabajo Añadir las nuevas entidades

Las nuevos elementos que se deben crear son:

Clase NuenoElemento --> Este va en el proyecto models y son los atributos de la BD

INuevoElementoRepository ---> Dentro de la carpeta repository

NuevoElementoRepository ---> Esta va dentro de Data

---------------------------------------------FINISH del Patron Repositorio-----------------------------------

Debemos de hacer la conexion con la BD

OJO deben de estar en el AplicationDbContext el DbSet de la nueva taba a crear

Desde la consola de paquetes: ---> verificar que esta en el proyecto Acceso a Datos ADO

add migration CreacionTablaInicialCategoria ---> No se pudo asi...

En mi caso no se pudo realizar la migracion segun el video

se realizo con Add-Migration ----> Revisar Link https://anexsoft.com/implementando-migraciones-con-entity-framework-y-asp-net-mvc

Setear Cadena de conexion

{

"ConnectionStrings": {

"DefaultConnection": "Server=(local);Database=PortalEDU\_3\_1;user id=sa; pwd=Ef1c13nc1@; Trusted\_Connection=false; MultipleActiveResultSets=true"

},

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft": "Warning",

"Microsoft.Hosting.Lifetime": "Information"

}

},

"AllowedHosts": "\*"

}

Luego enviamos la migracion a la BD con el comando update-database

------------------------------------------------TERMINA LA PRUEBA DE MIGRACION--------------------------------

INICIA CRUD CATEGORIAS

EN AREAS --> En el controllers de Admin crear un controlador CategoriasController

////Primer controlador

using Microsoft.AspNetCore.Mvc;

using PortalEDU.ADO.Data;

using PortalEDU.ADO.Data.Repository;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace PortalEdu3\_1.Areas.Admin.Controllers

{

[Area("Admin")]

public class CategoriasController : Controller

{

private readonly IContenedorTrabajo \_contenedorTrabajo;

public CategoriasController(IContenedorTrabajo contenedorTrabajo)

{

\_contenedorTrabajo = contenedorTrabajo;

}

public IActionResult Index()

{

return View();

}

// En esta region se manda a llamar los metodos de acceso a datos para traer la informacion que esta en nuestyro repositorio general

// llamado IRepository

#region LLAMADAS A LAS API

[HttpGet]

public IActionResult GetAll()

{

return Json(new {data = \_contenedorTrabajo.Categoria.GetAll()});

}

#endregion

}

}

-----------------------CREANDO LA VISTA---------------------------------------------------

- Clic derecho sobre el Index del controlador Agragr vista y elegir pagina de diseño \_Layout

-Luego de crar la vista asegurarse de instanciar la los contenedores de trabajo en el starup

Ejemplo:services.AddScoped<IContenedorTrabajo, ContenedorTrabajo>();

-Para la creacion de la tabla que tendra los datos se utilizo boopstrap y tambien javascrip se creo categoria.js

en la carpeta JS de wwwroot que tendra el script de la vista

---------------------------------------Script de la tabla---------------------------------------------------

var dataTable;

$(document).ready(function () {

cargarDatatable();

});

function cargarDatatable() {

dataTable = $("#tblCategorias").DataTable({

"ajax": {

"url": "/admin/categorias/GetAll",

"type": "GET",

"datatype": "json"

},

"columns": [

{ "data": "id", "width": "5%" },

{ "data": "nombre", "width": "50%" },

{ "data": "orden", "width": "20%" },

{

"data": "id",

"render": function (data) {

return `<div class="text-center">

<a href='/Admin/Categorias/Edit/${data}' class='btn btn-success text-white' style='cursor:pointer; width:100px;'>

<i class='fas fa-edit'></i> Editar

</a>

&nbsp;

<a onclick=Delete("/Admin/Categorias/Delete/${data}") class='btn btn-danger text-white' style='cursor:pointer; width:100px;'>

<i class='fas fa-trash-alt'></i> Borrar

</a>

`;

}, "width": "30%"

}

],

"language": {

"emptyTable": "No hay registros"

},

"width": "100%"

});

}

function Delete(url) {

swal({

title: "Esta seguro de borrar?",

text: "Este contenido no se puede recuperar!",

type: "warning",

showCancelButton: true,

confirmButtonColor: "#DD6B55",

confirmButtonText: "Si, borrar!",

closeOnconfirm: true

}, function () {

$.ajax({

type: 'DELETE',

url: url,

success: function (data) {

if (data.success) {

toastr.success(data.message);

dataTable.ajax.reload();

}

else {

toastr.error(data.message);

}

}

});

});

}

------------------------------------Aqui termina Script de la tabla--------------------------------------------

AHora se crea el formulario para crear categorias

- creamos el metodo get de create

[HttpGet]

public IActionResult Create() {

return View();

}

- Luegro agregamos la vista del metodo, click derecho sobre create y creamos la vista

- invocamos el modelo que vamos a utilizar @model PortalEDU.Models.Categoria

- Creamos el formulario con tagHelpers y se escribe el codigo para llamar una vista parcial para volver luego

se crea la vista parcial en Shared

ASI QUEDA CREATE:

@model PortalEDU.Models.Categoria

@{

ViewData["Title"] = "Create";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<h1>Create</h1>

<br />

<br />

<h3 class="text-center">Crear una nueva categoria </h3>

<form asp-action="Create" method="post">

<div class="row px-2 mx-2 border">

<div asp-validation-summary=" ModelOnly" class="text-danger"></div>

<div class="col-10 offset-1 mt-4">

<div class="form-group row">

<div class="col-4">

<label asp-for="Nombre"> </label>

</div>

<div class="col-8">

<input asp-for="Nombre" class="form-control" />

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

</div>

</div>

<div class="form-group row">

<div class="col-4">

<label asp-for="Orden"> </label>

</div>

<div class="col-8">

<input asp-for="Orden" class="form-control" />

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

</div>

</div>

<div class="form-group row">

<div class="col-8 offset-4">

<partial name="\_VistaCrearVolver"/>

</div>

</div>

</div>

</div>

</form>

Y la vista Parcial:

<div class="row">

<div class="col">

<button type="submit" class="btn btn-primary form-control"> Crear</button>

&nbsp;

</div>

<div class="col">

<a asp-action="Index" class="btn btn-success form-control">Volver a la vista</a>

</div>

</div>

Ahora a crear el metodo post crear de categoria para guardar cambios, queda de la siguiente manera:

[HttpPost]

[ValidateAntiForgeryToken]

public IActionResult Create(Categoria categoria)

{

if (ModelState.IsValid)

{

\_contenedorTrabajo.Categoria.Add(categoria);

\_contenedorTrabajo.Save();

return RedirectToAction(nameof(Index));

}

return View(categoria);

}

Continuadmos con el medoto Edit

[HttpGet]

public IActionResult Edit(int id)

{

Categoria categoria = new Categoria();

categoria = \_contenedorTrabajo.Categoria.Get(id);

if (categoria == null)

{

return NotFound();

}

return View(categoria);

}

EDIT CSHTML

@model PortalEDU.Models.Categoria

@{

ViewData["Title"] = "Edit";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<h1>Edit</h1>

<h1>Create</h1>

<br />

<br />

<h3 class="text-center">Editar categoria</h3>

<form asp-action="Create" method="post">

<div class="row px-2 mx-2 border">

<div asp-validation-summary=" ModelOnly" class="text-danger"></div>

<input type="hidden" asp-for="Id" />

<div class="col-10 offset-1 mt-4">

<div class="form-group row">

<div class="col-4">

<label asp-for="Nombre"> </label>

</div>

<div class="col-8">

<input asp-for="Nombre" class="form-control" placeholder="Categoria" />

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

</div>

</div>

<div class="form-group row">

<div class="col-4">

<label asp-for="Orden"> </label>

</div>

<div class="col-8">

<input asp-for="Orden" class="form-control" />

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

</div>

</div>

<div class="form-group row">

<div class="col-8 offset-4">

<div class="row">

<div class="col">

<button type="submit" class="btn btn-primary form-control" (////OJO AQUI PARA REGRESAR ID) asp-route-id="@Model.Id">Actualizar</button>

</div>

<div class="col">

<a asp-action="Index" class="btn btn-success form-control">Volver a la vista</a>

</div>

</div>

</div>

</div>

</div>

</div>

</form>

Y EN EL CONTROLER EDIT METODO POST -- Utilizamos el mismo de create y hacemos los cambios pertinentes

[HttpPost]

[ValidateAntiForgeryToken]

public IActionResult Edit(Categoria categoria)

{

if (ModelState.IsValid)

{

\_contenedorTrabajo.Categoria.Update(categoria);

\_contenedorTrabajo.Save();

return RedirectToAction(nameof(Index));

}

return View(categoria);

}

Continuamos con DELETE

Dentro de region: #region LLAMADAS A LAS API

public IActionResult Delete(int id)

{

var ObjFromDb = \_contenedorTrabajo.Categoria.Get(id);

if (ObjFromDb == null)

{

return Json(new { success = false, message = "Error al intentar borrar categoria" });

}

\_contenedorTrabajo.Categoria.Remove(ObjFromDb);

\_contenedorTrabajo.Save();

return Json(new { success = true, message = "Categoria borrada con exito" });

}

-----------------------------------------------------------------------------------------------------------------

Iniciamos CRUD de articulos

- Creamos el modelo

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using System.Text;

namespace PortalEDU.Models

{

public class Articulo

{

[Key]

public int Id { get; set; }

[Required(ErrorMessage ="El Nombre es requerido")]

[Display (Name ="Nombre del articulo")]

public string Nombre { get; set; }

[Display(Name = "Fecha de creacion")]

public string FechaCreacion { get; set; }

[Display(Name = "Imagen")]

[DataType(DataType.ImageUrl)]

public string UrlImagen { get; set; }

//////////////Ojo aqui hay una relacion

[Required]

public int CategoriaId { get; set; }

[ForeignKey("CategoriaId")]

public Categoria Categoria{ get; set; }

}

}

Luego agregamos la nueva entidad en el DbContext:

public DbSet<Articulo> Articulo { set; get; }

Realizar migracion y ya esta listo el nuevo modelo. para ADO

----------------------------------------------------------------------------------------------------------

Ahoro sigue agregar articulo a Repositorio y a la Unidad de trabajo

Dentro de data Repository duplicamos ICategioriaRepository y cambiamos el nombre a IArticuloRepository

Hacer lo mismo con CategiraRepositori que esta fuera de IRepository y cambiar nombre a ArticuloRepository

IArticuloRepository

using Microsoft.AspNetCore.Mvc.Rendering;

using PortalEDU.Models;

using System;

using System.Collections.Generic;

using System.Text;

namespace PortalEDU.ADO.Data.Repository

{

public interface IArticuloRepository : IRepository<Articulo>

{

void Update(Articulo articulo);

}

}

ArticuloRepository

using Microsoft.AspNetCore.Mvc.Rendering;

using PortalEDU.ADO.Data.Repository;

using PortalEDU.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace PortalEDU.ADO.Data

{

public class ArticuloRepository : Repository<Articulo>, IArticuloRepository

{

private readonly ApplicationDbContext \_db;

public ArticuloRepository(ApplicationDbContext db):base (db)

{

\_db = db;

}

public void Update(Articulo articulo)

{

var ObjDesdeDb = \_db.Articulo.FirstOrDefault(s => s.Id == articulo.Id);

ObjDesdeDb.Nombre = articulo.Nombre;

ObjDesdeDb.Descripcion = articulo.Descripcion;

ObjDesdeDb.FechaCreacion = articulo.FechaCreacion;

ObjDesdeDb.UrlImagen = articulo.UrlImagen;

ObjDesdeDb.CategoriaId = articulo.CategoriaId;

// \_db.SaveChanges();

}

}

}

///// En IcontenedorTrabajo, solamente agregar:IArticuloRepository Articulo { get; }

//// En contenedor de trabajo agregar las siguientes lineas:

Articulo = new ArticuloRepository(\_db);

public IArticuloRepository Articulo { get; private set; }

-----------------------------------------------------------------------------------------------------------------

Crear controlador Articulo, ViewModel, metodo y vista

//// En la region de este controlador se debe de llamar la categoria porque estaran relacionados:

#region LLAMADAS A LAS API

[HttpGet]

public IActionResult GetAll()

{

return Json(new { data = \_contenedorTrabajo.Articulo.GetAll(includeProperties: "Categoria") });

}

#endregion

MUCHO OJO CON ESTO: PARA PODER MOSTRAR LAS RELACIONES ENTRE LAS PROPIEDADES ES NECESARIO CREAR UNA NUEVA CLASE

QUE SE LLAMA VIEWMODEL (((ESTO ES CLAVE)))

using Microsoft.AspNetCore.Mvc.Rendering;

using System;

using System.Collections.Generic;

using System.Text;

namespace PortalEDU.Models.ViewModel

{

public class ArticuloVM

{

public Articulo Articulo { get; set; }

public IEnumerable<SelectListItem> ListaCategorias { get; set; }

}

}

En el controlador Instanciamos las propiedades del VM

public IActionResult Index()

{

ArticuloVM artivm = new ArticuloVM()

{

articulo = new PortalEDU.Models.Articulo(),

ListaCategorias = \_contenedorTrabajo.Categoria.GetListaCategoria()

// Articulo = new Models.

};

return View(artivm);

}

Luego en el Index cshtml y el javascript para llenado de la tabla solo hacemos los cambios pertienentes

-----------------------------------------------------------------------------------------------------------

Hora de crear el Create(Get) de Articulos ojo porque se debe de trabajar en ArticuloVM

Se copia el create de la vista anterior(Categoria) y se hacen los cambios pertinentes

Ojo con esto: dentro del form donde se defime el metodo POST o GET escribir esto: enctype="multipart/form-data"

Eso sirve para poder subir archivos

para mostrar la relacion de 1 a muchos se debe de hacer lo siguiente:

@\*// Aqui se despliega la lista de categoria, la relacion de 1 a muchos\*@

<div class="form-group row">

<div class="col-4">

<label asp-for="articulo.CategoriaId"> </label>

</div>

<div class="col-8">

@Html.DropDownListFor(m => m.articulo.CategoriaId, Model.ListaCategorias, "-Porfavor seleccione una categoria-", new

{ @class = "form-control" })

</div>

</div>

SE UTILIZO UN PLUGIN PARA PODER CREAR UN TEXTAREA DONDE SE PUEDA FORMATEAR EL TEXTO

PD:VER LAS PPROPIEDADES EN PLUGINS Y TOOLBARMAS ADECUADAS, AQUI VAN TODAS.

@section Scripts{

<script src="https://cdn.tiny.cloud/1/i2fj0cj70ajhbmkrqqx8su1fo2at2l2ssky2cwkigjkpr7ue/tinymce/5/tinymce.min.js"></script>

<script>

tinymce.init({

selector: 'textarea',

plugins: 'a11ychecker advcode casechange formatpainter linkchecker autolink lists checklist media mediaembed pageembed permanentpen powerpaste table advtable tinycomments tinymcespellchecker',

toolbar: 'a11ycheck addcomment showcomments casechange checklist code formatpainter pageembed permanentpen table',

toolbar\_mode: 'floating',

tinycomments\_mode: 'embedded',

tinycomments\_author: 'Portal Educativo',

});

</script>

}

-------------------------------------------------------------------------------------------------------------------------------

Inciia video 53 insertar articulos y subida de archivos

Ojo para el emtodo post create se debe de instaciar una propiedad que nos permita trabajar con subida de archvos, esta:

private readonly IWebHostEnvironment \_hostinEnvironment;

Luego se pasa como parmetro en el constructor de ArticulosControler Sin el guin bajo:

IMPORTANTE ANALIZAR ESTE METODO QUE RECIBE UNA IMAGEN, SIEMPRE EN EL METODO CREATE

[HttpPost]

[ValidateAntiForgeryToken]

public IActionResult Create(ArticuloVM artiVM)

{

if (ModelState.IsValid)

{

string rutaPrincipal = \_hostinEnvironment.WebRootPath;

var archivos = HttpContext.Request.Form.Files;

if (artiVM.articulo.Id == 0)

{

string nombreArchivo = Guid.NewGuid().ToString();

var subidas = Path.Combine(rutaPrincipal, @"imagenes\articulos");

var extension = Path.GetExtension(archivos[0].FileName);

using (var fileStreams = new FileStream(Path.Combine(subidas, nombreArchivo + extension), FileMode.Create))

{

archivos[0].CopyTo(fileStreams);

}

artiVM.articulo.UrlImagen = @"\imagenes\artiulos" + nombreArchivo + extension;

artiVM.articulo.FechaCreacion = DateTime.Now.ToString();

\_contenedorTrabajo.Articulo.Add(artiVM.articulo);

\_contenedorTrabajo.Save();

return RedirectToAction(nameof(Index));

}

}

artiVM.ListaCategorias = \_contenedorTrabajo.Categoria.GetListaCategoria();

return View(artiVM);

}

///Agegar estas lienas a metodo post para agregar una excepcion si no se coloca nada en los campos del articulo:

artiVM.ListaCategorias = \_contenedorTrabajo.Categoria.GetListaCategoria();

return View(artiVM);

}

// Creamos el EDIT de articulos

[HttpGet]

public IActionResult Edit(int? id)

{

ArticuloVM artivm = new ArticuloVM()

{

articulo = new PortalEDU.Models.Articulo(),

ListaCategorias = \_contenedorTrabajo.Categoria.GetListaCategoria()

// Articulo = new Models.

};

if (id != null)

{

artivm.articulo = \_contenedorTrabajo.Articulo.Get(id.GetValueOrDefault());

}

return View(artivm);

}

A CONTINUACION LA VISTA:

Se reutiliza todo del create, solo cambiar esta linea:

<button type="submit" class="btn btn-primary form-control" asp-route-id="@Model.articulo.Id}">Actualizar</button>

<input type="hidden" asp-for="articulo.Id" />

Y tambien poner la imagen anterior:

<div class="col-4 pt-3">

<img src="@Model.articulo.UrlImagen" width="100%" style="border-radius:5px; border:1px solid #bbb9b9;" />

</div>

-------------------------------------------------------------------------------------------------------------------

Ojo con el metodo EDIT POST, se encuentra la parte por si cambian imagen y una parte por si se conserva la imagen.

public IActionResult Edit(ArticuloVM artiVM)

{

if (ModelState.IsValid)

{

string rutaPrincipal = \_hostinEnvironment.WebRootPath;

var archivos = HttpContext.Request.Form.Files;

var articuloDesdeDb = \_contenedorTrabajo.Articulo.Get(artiVM.articulo.Id);

if (archivos.Count() > 0 )

{

// Editamos Imagenes

string nombreArchivo = Guid.NewGuid().ToString();

var subidas = Path.Combine(rutaPrincipal, @"imagenes\articulos");

var extension = Path.GetExtension(archivos[0].FileName);

var nuevaExtension = Path.GetExtension(archivos[0].FileName);

var rutaImagen = Path.Combine(rutaPrincipal, articuloDesdeDb.UrlImagen.TrimStart('\\'));

if (System.IO.File.Exists(rutaImagen))

{

System.IO.File.Delete(rutaImagen);

}

// Subimos nuevamente el archivo

using (var fileStreams = new FileStream(Path.Combine(subidas, nombreArchivo + nuevaExtension), FileMode.Create))

{

archivos[0].CopyTo(fileStreams);

}

artiVM.articulo.UrlImagen = @"\imagenes\articulos\" + nombreArchivo + nuevaExtension;

artiVM.articulo.FechaCreacion = DateTime.Now.ToString();

\_contenedorTrabajo.Articulo.Update(artiVM.articulo);

\_contenedorTrabajo.Save();

return RedirectToAction(nameof(Index));

}

else

{

// Aqui es cuando la imagen ya existe y no se reeemplaza

// debe conservar la que ya esta en la base de datos

artiVM.articulo.UrlImagen = articuloDesdeDb.UrlImagen;

}

\_contenedorTrabajo.Articulo.Update(artiVM.articulo);

\_contenedorTrabajo.Save();

return RedirectToAction(nameof(Index));

}

return View();

}

--------------------------------------------------------------------------------------------------------------------------

Ahora el metodo Borrar

public IActionResult Delete(int id)

{

var articuloDesdeDb = \_contenedorTrabajo.Articulo.Get(id);

string rutaDirectorioPrincipal = \_hostinEnvironment.WebRootPath;

var rutaImagen = Path.Combine(rutaDirectorioPrincipal, articuloDesdeDb.UrlImagen.TrimStart('\\'));

if (System.IO.File.Exists(rutaImagen))

{

System.IO.File.Delete(rutaImagen);

}

if (articuloDesdeDb == null)

{

return Json(new { success = false, message = "Error al intentar borrar articulo" });

}

\_contenedorTrabajo.Articulo.Remove(articuloDesdeDb);

\_contenedorTrabajo.Save();

return Json(new { success = true, message = "Articulo borrado con exito" });

}

-----------------------------------------------------------------------------------------------------------------------

Creacion de estructura para slider

- Crear la propiedad

- Agregar a Application DBCOntext

- Realizar migracion

- Crear el ISliderRepository

- Crear el SliderRepository

- Modifica IcontenedoTrabajo

- Modificar ContenedorTrabajo

- Crear El Controlador

- Crear la vista

- Crear el API JS

- De ahi en adelante se crea el CRUD

- En editar OJO siempre enviar el ID como campo oculto: <input type="hidden" asp-for="articulo.Id" />

-----------------------------------------------------------------------------------------------------------------------------

Crenado la vista para el slider de usuario anonimo

- Se va a acceder a los elementos de slider y articulos entonces se debe crear el VM respectivo:

namespace PortalEDU.Models.ViewModel

{

public class HomeVM

{

public IEnumerable<Slider> slider { get; set; }

public IEnumerable<Articulo> ListaArticulos { get; set; }

}

}

Luego se configura el controlador:

namespace PortalEdu3\_1.Controllers

{

[Area ("Anonimo")]

public class HomeController : Controller

{

private readonly IContenedorTrabajo \_contenedorTrabajo;

public HomeController(IContenedorTrabajo contenedorTrabajo)

{

\_contenedorTrabajo = contenedorTrabajo;

}

public IActionResult Index()

{

HomeVM homeVM = new HomeVM()

{

slider = \_contenedorTrabajo.Slider.GetAll(),

ListaArticulos = \_contenedorTrabajo.Articulo.GetAll()

};

return View(homeVM);

}

}

}

y Ojo con lo que se hace en la vista: ASI SE CREAN LAS TARJETAS:

Aqui inicia la creacion de los articulos\*@

@if (Model.ListaArticulos.Count() > 0)

{

<hr />

<div class="row">

@foreach (var articulo in Model.ListaArticulos.OrderBy(o => o.Id))

{

<div class="col-4">

<div class="card">

<img src="@articulo.UrlImagen" class="img-thumbnail" width="100%"/>

<div class="card-body">

<h5 class="text-center">@articulo.Nombre</h5>

<p>@articulo.FechaCreacion </p>

<a asp-action="Details" class="btn btn-success" style="border-radius:2px" asp-route-id="@articulo.Id">Más información</a>

</div>

</div>

</div>

}

</div>

}

else

{

<p> No exsiten articulos</p>

}

- cre creeara la vista details

Aqui se creo una card mas grande nada mas

-----------------------------------------------------------------------------------------------------------------------------------------------------

Inicia la configuracion de la identidad:

- primero debemos hacer un scaffold de la identidad, en el proyecto web se agrega iten con scaffold y se elige identidad

- Se crea una nueva clase en models para agregar mas propiedades a la tabla de usuarios que trae ASP por default, esa clase debe de eredar de IdentityUser:

using Microsoft.AspNetCore.Identity;

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Text;

namespace PortalEDU.Models

{

public class ApplicationUser: IdentityUser

{

[Required (ErrorMessage =("El Nombre es requerido"))]

public string Nombre { get; set; }

public string Direccion { get; set; }

}

}

- Luego como siempre, despues de crear una nueva propiedad, es necesario crear la entidad en el DBCOntext.

- Procedemos a ejecutar la migracion.

- Agregamos las nuevas propiedades a la clase register

- modelamos la vista con las nuevas propiedades

- para los roles creamos una nueva clase de constantes en el proyecto de utilideades, aqui dememos de crear propiedades constantes con los nombres de todos los roles que utilizaremos

- En la vista de registro importamos la clase utilidades parta poder hacer uso de las propiedades

- Se crea un nuevo div donde accedemos a los roles creados en constantes

- En el register debemos invocar el RoloManager juesto en el metodo registerModel: private readonly RoleManager<IdentityRole> \_roleManager;

- Se debe de instancia en el constructor tambien: RoleManager<IdentityRole> roleManager

- Luego lo asignamos en el mismo constructor= \_roleManager = roleManager;

- En el moetod OnPostAsync, realizamos los siguientes cambios:

var user = new ApplicationUser {

UserName = Input.Email,

Email = Input.Email,

Nombre = Input.Nombre,

Ciudad = Input.Ciudad,

Direccion = Input.Direccion,

Pais = Input.Pais,

PhoneNumber = Input.PhoneNumber,

EmailConfirmed = true};

- En el mismo metodo en el IF result.Succeded, escribimos las siguientes cambios, todo depende de cuantos roles necesitamos:

// Aqui validamos si los roles existen, sino se crean

if (!await \_roleManager.RoleExistsAsync(Constantes.Admin))

{

await \_roleManager.CreateAsync(new IdentityRole(Constantes.Admin));

await \_roleManager.CreateAsync(new IdentityRole(Constantes.Alumno));

await \_roleManager.CreateAsync(new IdentityRole(Constantes.Anonimo));

await \_roleManager.CreateAsync(new IdentityRole(Constantes.Docente));

await \_roleManager.CreateAsync(new IdentityRole(Constantes.Padre));

}

//Obterne el rol del usuario

string rol = Request.Form["radUsuarioRole"].ToString();

//Validamos el rol seleccionado

if (rol ==Constantes.Admin)

{

await \_userManager.AddToRoleAsync(user, Constantes.Admin);

}

else if (rol == Constantes.Alumno)

{

await \_userManager.AddToRoleAsync(user, Constantes.Alumno);

}

else if (rol == Constantes.Anonimo)

{

await \_userManager.AddToRoleAsync(user, Constantes.Anonimo);

}

else if (rol == Constantes.Docente)

{

await \_userManager.AddToRoleAsync(user, Constantes.Docente);

}

else

{

if(rol == Constantes.Padre)

{

await \_userManager.AddToRoleAsync(user, Constantes.Padre);

}

}

- Pasra est ejercicio comentamos el envio y confirmacion de email

- Ahora en el archivo starup modificamos lo siguiente:

services.AddIdentity<IdentityUser, IdentityRole>(options => options.SignIn.RequireConfirmedAccount = true)

.AddEntityFrameworkStores<ApplicationDbContext>().AddDefaultTokenProviders();

------------------------------------------------------------------------------------------------------------------------

- Corregir el error del emailSender:

- En utilidades creamos una nueva clase EmailSender y heredamos de IEmailSender y implementamos esa interfaz

- En consigureservice agregamos un nuevo servicio: services.AddSingleton<IEmailSender, EmailSender>();

------------------------------------------------------------------------------------------------------------------

Crear los usuarios:

- Agregarlo a los repositorios

En el Irepository unicamente esto:

public interface IUsuarioRepository : IRepository<ApplicationUser>

{

void BloquearUsuario(string IdUsuario);

void DesbloquearUsuario(string IdUsuario);

}

}

-En el UsuarioRepository:

Implemetamos interfaz y accedemos a los metodos:

public class UsuarioRepository : Repository<ApplicationUser>, IUsuarioRepository

{

private readonly ApplicationDbContext \_db;

public UsuarioRepository(ApplicationDbContext db) : base(db)

{

\_db = db;

}

public void BloquearUsuario(string IdUsuario)

{

var usuarioDesdeDB = \_db.ApplicationUser.FirstOrDefault(u => u.Id == IdUsuario);

usuarioDesdeDB.LockoutEnd = DateTime.Now.AddYears(100);

\_db.SaveChanges();

}

public void DesbloquearUsuario(string IdUsuario)

{

var usuarioDesdeDB = \_db.ApplicationUser.FirstOrDefault(u => u.Id == IdUsuario);

usuarioDesdeDB.LockoutEnd = DateTime.Now;

\_db.SaveChanges();

}

}

}

- Lo agregamos en el IContenedorTrabajo: IUsuarioRepository Usuario { get; }

- Lo agregamos en el ContenedorTrabajo

---------------------------------------------------------------------------------------------------------------

Creamos el controlador:

namespace PortalEdu3\_1.Areas.Admin.Controllers

{

[Area("Admin")]

public class UsuariosController : Controller

{

private readonly IContenedorTrabajo \_contenedorTrabajo;

public UsuariosController(IContenedorTrabajo contenedorTrabajo)

{

\_contenedorTrabajo = contenedorTrabajo;

}

public IActionResult Index()

{

var claimsIdentity = (ClaimsIdentity)this.User.Identity;

var usuarioActual = claimsIdentity.FindFirst(ClaimTypes.NameIdentifier);

return View(\_contenedorTrabajo.Usuario.GetAll(u => u.Id !=usuarioActual.Value));

}

}

}

Seguimos con la vista:

- Ver vista de usuarios para mayor referencia.

Metodos para bloquear y desbloquear usuario: esto invoca lo qcreado en el contenerdor, cuando se bloque escribe en lockoutend 100 años

public IActionResult Bloquear(string id)

{

if (id == null)

{

return NotFound();

}

\_contenedorTrabajo.Usuario.BloquearUsuario(id);

return RedirectToAction(nameof(Index));

}

- Cambiamos algunas cosas de la vista login

- IMPORTANTE cambiar en Manage Login todas las partes donde aparece IdentityUser por ApplicationUser (Yuca)

Luego en el starup service, agregamos esta linea: services.AddIdentity<ApplicationUser, IdentityRole>(options => options.SignIn.RequireConfirmedAccount = true)

.AddRoleManager<RoleManager<IdentityRole>>()

- Ojo en ManageNav permanece IdentityUser

- Ojo en la vista LoginPartial se debe de cambiar a ApplicationUser

- En el manage, index.cs debemos de colocar las propiedades que se mostraran para el cambio de perfil

- En la misma clase index del manage en el metodo LoadAsync agregamos las propiedades que se deben mantaner para que el usuario pueda ver los datos actuales

- Ojo ver el modelado de la clase Index.CS

- Ojo para que se guarden los campos es necesario colocar el siguiente codigo en el metodo OnPostAsync: user.Nombre = Input.Nombre;

await \_userManager.UpdateAsync(user); <------ Aqui se deben de colocar todos los campos que queramos que se puedan cambiar del perfil

- Buscar Status Massge esta bonito.

---------------------------------------------------------------------------------------------------------------------------------------------------

AHORA VANOS A PROTEGER LOS ACCESOS

- Primero protegemos el menu del Layout. Importamos el Using de utilidades para poder acceder a los roles(Constantes)

- Luego colocalos la siguiente linea donde queramos proteger, dependiendo el usuario:

@if (User.IsInRole(Constantes.Admin))

{

--------> Aqui encerramos lo que queremos proteger.

}

- Ojo, aqui todavia se puede ingresar por el navegador a los controladores, por lo que al inicio de la clase debemos de colocar [Autorize] EASYYYY...

- Con el mismo if protegemos el html de cualquier vista

- "Pequeño tip, para ocultar la columnas de jquery ponemos lo siguiente: "visible": false, "

----------------------------------------------------------------------------------------------------

Iniciamos con siembra de datos, esto sirve para tener unos parametros iniciales cuando se hace un deploy, por lo menos tener el usuario admin

- Utilizamos la carpeta inicializadora

- Creamos una clase "puplica" normar InicializadorDB

- Creamos Una interfaz publica IinicializadorDB

- Dentro de la interfaz creamos :

void Inicializar();

- Revisar la clase e interfaz

- Para que se inicie con el programa debemos de modificar el startUP agregando un nuevo servicio:services.AddScoped<IInicializadorDB, InicializadorDB>();

- En el metodo COnfigure le debemos pasar un nuevo argumento de la interfaz IInicializadorDB y luego llamarla con la variable creada en el argumento, queda asi:

public void Configure(IApplicationBuilder app, IWebHostEnvironment env, IInicializadorDB dbInicial)

dbInicial.Inicializar();