Notas de .NET 6

Crear la BD con EF

1. Crear la entidad
2. Crear el DbContext

    public class DataContext : DbContext

    {

        public DataContext(DbContextOptions<DataContext> options) : base(options)

        {

        }

        public DbSet<Country> Countries { get; set; }

        protected override void OnModelCreating(ModelBuilder modelBuilder)

        {

            base.OnModelCreating(modelBuilder);

            modelBuilder.Entity<Country>().HasIndex(c => c.Name).IsUnique();

        }

    }

1. Configurar el string de conexión:

  "ConnectionStrings": {

    "DefaultConnection": "Server=(localdb)\\MSSQLLocalDB;Database=Shopping;Trusted\_Connection=True;MultipleActiveResultSets=true"

  }

1. Agregar los paquetes:

Microsoft.EntityFrameworkCore.SqlServer

Microsoft.EntityFrameworkCore.Tools

1. Configurar la inyección del data context:

builder.Services.AddDbContext<DataContext>(o =>

{

    o.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection"));

});

1. Correr los comandos:

add-migration InitialDb

update-database

1. Crear el controlador y adicionar algunos registros.

Ejemplo del DataTable

@model IEnumerable<Shooping.Data.Entities.Country>

@{

    ViewData["Title"] = "Index";

}

<link rel="stylesheet" href="https://cdn.datatables.net/1.10.19/css/jquery.dataTables.min.css" />

<p>

    <a asp-action="Create" class="btn btn-outline-primary">Crear Nuevo</a>

</p>

<div class="row">

    <div class="col-md-12">

        <div class="panel panel-default">

            <div class="panel-heading">

                <h3 class="panel-title">Países</h3>

            </div>

            <div class="panel-body">

                <table class="table table-hover table-responsive table-striped" id="MyTable">

                    <thead>

                        <tr>

                            <th>

                                @Html.DisplayNameFor(model => model.Name)

                            </th>

                            <th>

                                @Html.DisplayNameFor(model => model.StatesNumber)

                            </th>

                            <th></th>

                        </tr>

                    </thead>

                    <tbody>

                        @foreach (var item in Model)

                        {

                            <tr>

                                <td>

                                    @Html.DisplayFor(modelItem => item.Name)

                                </td>

                                <td>

                                    @Html.DisplayFor(modelItem => item.StatesNumber)

                                </td>

                                <td>

                                    <a asp-action="Edit" asp-route-id="@item.Id" class="btn btn-outline-warning">Editar</a>

                                    <a asp-action="Details" asp-route-id="@item.Id" class="btn btn-outline-info">Detalles</a>

                                    <a asp-action="Delete" asp-route-id="@item.Id" class="btn btn-outline-danger">Borrar</a>

                                </td>

                            </tr>

                        }

                    </tbody>

                </table>

            </div>

        </div>

    </div>

</div>

@section Scripts {

    @{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

    <script src="//cdn.datatables.net/1.10.19/js/jquery.dataTables.min.js"></script>

    <script type="text/javascript">

        $(document).ready(function () {

            $('#MyTable').DataTable({

                "language": {

                    "url": "//cdn.datatables.net/plug-ins/9dcbecd42ad/i18n/Spanish.json"

                },

                "aLengthMenu": [

                    [25, 50, 100, 200, -1],

                    [25, 50, 100, 200, "Todos"]

                ]

            });

        });

    </script>

}

Validación de duplicidad de índice

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create(Country country)

{

    if (ModelState.IsValid)

    {

        \_context.Add(country);

        try

        {

            await \_context.SaveChangesAsync();

            return RedirectToAction(nameof(Index));

        }

        catch (DbUpdateException dbUpdateException)

        {

            if (dbUpdateException.InnerException.Message.Contains("duplicate"))

            {

                ModelState.AddModelError(string.Empty, "Ya existe un país con el mismo nombre.");

            }

            else

            {

                ModelState.AddModelError(string.Empty, dbUpdateException.InnerException.Message);

            }

        }

        catch (Exception exception)

        {

            ModelState.AddModelError(string.Empty, exception.Message);

        }

    }

    return View(country);

}

Cambios en caliente

1. Agregar el paquete: **Microsoft.AspNetCore.Mvc.Razor.RuntimeCompilation**
2. Agregar esta línea en el **Program**: **builder.Services.AddRazorPages().AddRazorRuntimeCompilation();**

Relación uno a muchos e índice compuesto

* Clase **Country**:

using System.ComponentModel.DataAnnotations;

namespace Shooping.Data.Entities

{

    public class Country

    {

        public int Id { get; set; }

        [Display(Name = "País")]

        [MaxLength(50, ErrorMessage = "El campo {0} debe tener máximo {1} caractéres.")]

        [Required(ErrorMessage = "El campo {0} es obligatorio.")]

        public string Name { get; set; }

        public ICollection<State> States { get; set; }

        [Display(Name = "Estados / Departamentos")]

        public int StatesNumber =>  States == null ? 0: States.Count;

    }

}

* Clase **State**:

using System.ComponentModel.DataAnnotations;

namespace Shooping.Data.Entities

{

    public class State

    {

        public int Id { get; set; }

        [Display(Name = "Departamento/Estado")]

        [MaxLength(50, ErrorMessage = "El campo {0} debe tener máximo {1} caractéres.")]

        [Required(ErrorMessage = "El campo {0} es obligatorio.")]

        public string Name { get; set; }

        public Country Country { get; set; }

        public ICollection<City> Cities { get; set; }

        [Display(Name = "Ciudades")]

        public int CitiesNumber => Cities == null ? 0 : Cities.Count;

    }

}

* Clase **City**:

using System.ComponentModel.DataAnnotations;

namespace Shooping.Data.Entities

{

    public class City

    {

        public int Id { get; set; }

        [Display(Name = "Ciudad")]

        [MaxLength(50, ErrorMessage = "El campo {0} debe tener máximo {1} caractéres.")]

        [Required(ErrorMessage = "El campo {0} es obligatorio.")]

        public string Name { get; set; }

        public State State { get; set; }

    }

}

* Modificación al **DataContext**:

public DbSet<Category> Categories { get; set; }

public DbSet<City> Cities { get; set; }

public DbSet<Country> Countries { get; set; }

public DbSet<State> States { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

    base.OnModelCreating(modelBuilder);

    modelBuilder.Entity<Category>().HasIndex(c => c.Name).IsUnique();

    modelBuilder.Entity<City>().HasIndex("Name", "StateId").IsUnique();

    modelBuilder.Entity<Country>().HasIndex(c => c.Name).IsUnique();

    modelBuilder.Entity<State>().HasIndex("Name", "CountryId").IsUnique();

}

Configuración del alimentador de la BD

1. Agregamos la clase **SeedDb** dentro de la carpeta **Data**:

using Shooping.Data.Entities;

namespace Shooping.Data

{

    public class SeedDb

    {

        private readonly DataContext \_context;

        public SeedDb(DataContext context)

        {

            \_context = context;

        }

        public async Task SeedAsync()

        {

            await \_context.Database.EnsureCreatedAsync();

            await CheckCountriesAsync();

            await CheckCategoriesAsync();

        }

        private async Task CheckCategoriesAsync()

        {

            if (!\_context.Categories.Any())

            {

                \_context.Categories.Add(new Category { Name = "Tecnología" });

                \_context.Categories.Add(new Category { Name = "Ropa" });

                \_context.Categories.Add(new Category { Name = "Gamer" });

                \_context.Categories.Add(new Category { Name = "Belleza" });

                \_context.Categories.Add(new Category { Name = "Nutrición" });

            }

            await \_context.SaveChangesAsync();

        }

        private async Task CheckCountriesAsync()

        {

            if (!\_context.Countries.Any())

            {

                \_context.Countries.Add(new Country

                {

                    Name = "Colombia",

                    States = new List<State>()

                    {

                        new State()

                        {

                            Name = "Antioquia",

                            Cities = new List<City>() {

                                new City() { Name = "Medellín" },

                                new City() { Name = "Itagüí" },

                                new City() { Name = "Envigado" },

                                new City() { Name = "Bello" },

                                new City() { Name = "Rionegro" },

                            }

                        },

                        new State()

                        {

                            Name = "Bogotá",

                            Cities = new List<City>() {

                                new City() { Name = "Usaquen" },

                                new City() { Name = "Champinero" },

                                new City() { Name = "Santa fe" },

                                new City() { Name = "Useme" },

                                new City() { Name = "Bosa" },

                            }

                        },

                    }

                    });

                \_context.Countries.Add(new Country

                {

                    Name = "Estados Unidos",

                    States = new List<State>()

                    {

                        new State()

                        {

                            Name = "Florida",

                            Cities = new List<City>() {

                                new City() { Name = "Orlando" },

                                new City() { Name = "Miami" },

                                new City() { Name = "Tampa" },

                                new City() { Name = "Fort Lauderdale" },

                                new City() { Name = "Key West" },

                            }

                        },

                        new State()

                        {

                            Name = "Texas",

                            Cities = new List<City>() {

                                new City() { Name = "Houston" },

                                new City() { Name = "San Antonio" },

                                new City() { Name = "Dallas" },

                                new City() { Name = "Austin" },

                                new City() { Name = "El Paso" },

                            }

                        },

                    }

                });

            }

            await \_context.SaveChangesAsync();

        }

    }

}

1. Modificamos el **Program**:

builder.Services.AddTransient<SeedDb>();

WebApplication? app = builder.Build();

SeedData(app);

void SeedData(WebApplication app)

{

    IServiceScopeFactory? scopedFactory = app.Services.GetService<IServiceScopeFactory>();

    using (IServiceScope? scope = scopedFactory.CreateScope())

    {

        SeedDb? service = scope.ServiceProvider.GetService<SeedDb>();

        service.SeedAsync().Wait();

    }

}

1. Modificamos el **Index** de **Countries** para que muestre los estados.

Combos Helper

1. Creamos la interfaz:

using Microsoft.AspNetCore.Mvc.Rendering;

namespace Shooping.Helpers

{

    public interface ICombosHelper

    {

        IEnumerable<SelectListItem> GetComboCategories();

    }

}

1. Creamos la implementation:

using Microsoft.AspNetCore.Mvc.Rendering;

using Shooping.Data;

namespace Shooping.Helpers

{

    public class CombosHelper : ICombosHelper

    {

        private readonly DataContext \_context;

        public CombosHelper(DataContext context)

        {

            \_context = context;

        }

        public IEnumerable<SelectListItem> GetComboCategories()

        {

            List<SelectListItem> list = \_context.Categories.Select(x => new SelectListItem

            {

                Text = x.Name,

                Value = $"{x.Id}"

            })

                .OrderBy(x => x.Text)

                .ToList();

            list.Insert(0, new SelectListItem

            {

                Text = "[Seleccione una categoría...]",

                Value = "0"

            });

            return list;

        }

    }

}

1. Configuramos la inyección:

builder.Services.AddScoped<ICombosHelper, CombosHelper>();

Blob Helper

1. Creamos el blob en azure y agregamos valores al **appsettings**:

"Blob": {

  "ConnectionString": "DefaultEndpointsProtocol=https;AccountName=shoppingprep;AccountKey=9azHu2kSy5Lq199tvX9fOsdtacLhucwHYAt+xj+qKXIvzHNzfdV5e4IrJzRcnymnh2CTv8Xtl7w+VBc1PW72ng==;EndpointSuffix=core.windows.net"

}

1. Creamos la interfaz:

namespace Shooping.Helpers

{

    public interface IBlobHelper

    {

        Task<Guid> UploadBlobAsync(IFormFile file, string containerName);

        Task<Guid> UploadBlobAsync(byte[] file, string containerName);

        Task<Guid> UploadBlobAsync(string image, string containerName);

        Task DeleteBlobAsync(Guid id, string containerName);

    }

}

1. Creamos la implementation:

using Microsoft.WindowsAzure.Storage;

using Microsoft.WindowsAzure.Storage.Blob;

namespace Shooping.Helpers

{

    public class BlobHelper : IBlobHelper

    {

        private readonly CloudBlobClient \_blobClient;

        public BlobHelper(IConfiguration configuration)

        {

            string keys = configuration["Blob:ConnectionString"];

            CloudStorageAccount storageAccount = CloudStorageAccount.Parse(keys);

            \_blobClient = storageAccount.CreateCloudBlobClient();

        }

        public async Task<Guid> UploadBlobAsync(byte[] file, string containerName)

        {

            MemoryStream stream = new MemoryStream(file);

            Guid name = Guid.NewGuid();

            CloudBlobContainer container = \_blobClient.GetContainerReference(containerName);

            CloudBlockBlob blockBlob = container.GetBlockBlobReference($"{name}");

            await blockBlob.UploadFromStreamAsync(stream);

            return name;

        }

        public async Task<Guid> UploadBlobAsync(IFormFile file, string containerName)

        {

            Stream stream = file.OpenReadStream();

            Guid name = Guid.NewGuid();

            CloudBlobContainer container = \_blobClient.GetContainerReference(containerName);

            CloudBlockBlob blockBlob = container.GetBlockBlobReference($"{name}");

            await blockBlob.UploadFromStreamAsync(stream);

            return name;

        }

        public async Task<Guid> UploadBlobAsync(string image, string containerName)

        {

            Stream stream = File.OpenRead(image);

            Guid name = Guid.NewGuid();

            CloudBlobContainer container = \_blobClient.GetContainerReference(containerName);

            CloudBlockBlob blockBlob = container.GetBlockBlobReference($"{name}");

            await blockBlob.UploadFromStreamAsync(stream);

            return name;

        }

        public async Task DeleteBlobAsync(Guid id, string containerName)

        {

            CloudBlobContainer container = \_blobClient.GetContainerReference(containerName);

            CloudBlockBlob blockBlob = container.GetBlockBlobReference($"{id}");

            await blockBlob.DeleteAsync();

        }

    }

}

1. Configuramos la inyección:

builder.Services.AddScoped<IBlobHelper, BlobHelper>();

Adición de entidades de usuarios

1. Como vamos a tener dos tipos de usuarios; administradores y usuarios. Vamos a crear una enumeración para diferenciarlos. Creamos la carpeta **Enums** en el proyecto **Common** y dentro de esta carpeta la enumeración **UserType**:

public enum UserType

{

    Admin,

    User

}

1. En el proyecto **Web** en la carpeta **Data**, crear la carpeta **Entities** y dentro de esta, crear la entidad **User**:

public class User : IdentityUser

{

    [Display(Name = "Documento")]

    [MaxLength(20, ErrorMessage = "El campo {0} debe tener máximo {1} caractéres.")]

    [Required(ErrorMessage = "El campo {0} es obligatorio.")]

    public string Document { get; set; }

    [Display(Name = "Nombres")]

    [MaxLength(50, ErrorMessage = "El campo {0} debe tener máximo {1} caractéres.")]

    [Required(ErrorMessage = "El campo {0} es obligatorio.")]

    public string FirstName { get; set; }

    [Display(Name = "Apellidos")]

    [MaxLength(50, ErrorMessage = "El campo {0} debe tener máximo {1} caractéres.")]

    [Required(ErrorMessage = "El campo {0} es obligatorio.")]

    public string LastName { get; set; }

    [Display(Name = "Dirección")]

    [MaxLength(200, ErrorMessage = "El campo {0} debe tener máximo {1} caractéres.")]

    [Required(ErrorMessage = "El campo {0} es obligatorio.")]

    public string Address { get; set; }

    [Display(Name = "Foto")]

    public Guid ImageId { get; set; }

    //TODO: Pending to put the correct paths

    [Display(Name = "Foto")]

    public string ImageFullPath => ImageId == Guid.Empty

        ? $"https://localhost:7057/images/noimage.png"

        : $"https://shoppingprep.blob.core.windows.net/users/{ImageId}";

    [Display(Name = "Tipo de usuario")]

    public UserType UserType { get; set; }

    [Display(Name = "Ciudad")]

    public City City { get; set; }

    [Display(Name = "Usuario")]

    public string FullName => $"{FirstName} {LastName}";

    [Display(Name = "Usuario")]

    public string FullNameWithDocument => $"{FirstName} {LastName} - {Document}";

}

1. Modificar el **DataContext**:

public class DataContext : IdentityDbContext<User>

1. Crear la interfaz **IUserHelper**:

public interface IUserHelper

{

    Task<User> GetUserAsync(string email);

    Task<IdentityResult> AddUserAsync(User user, string password);

    Task CheckRoleAsync(string roleName);

    Task AddUserToRoleAsync(User user, string roleName);

    Task<bool> IsUserInRoleAsync(User user, string roleName);

}

1. Creamos la implementación de la interfaz **UserHelper**:

public class UserHelper : IUserHelper

{

    private readonly DataContext \_context;

    private readonly UserManager<User> \_userManager;

    private readonly RoleManager<IdentityRole> \_roleManager;

    public UserHelper(DataContext context, UserManager<User> userManager, RoleManager<IdentityRole> roleManager)

    {

        \_context = context;

        \_userManager = userManager;

        \_roleManager = roleManager;

    }

    public async Task<IdentityResult> AddUserAsync(User user, string password)

    {

        return await \_userManager.CreateAsync(user, password);

    }

    public async Task AddUserToRoleAsync(User user, string roleName)

    {

        await \_userManager.AddToRoleAsync(user, roleName);

    }

    public async Task CheckRoleAsync(string roleName)

    {

        bool roleExists = await \_roleManager.RoleExistsAsync(roleName);

        if (!roleExists)

        {

            await \_roleManager.CreateAsync(new IdentityRole

            {

                Name = roleName

            });

        }

    }

    public async Task<User> GetUserAsync(string email)

    {

        return await \_context.Users

            .Include(u => u.City)

            .FirstOrDefaultAsync(u => u.Email == email);

    }

    public async Task<bool> IsUserInRoleAsync(User user, string roleName)

    {

        return await \_userManager.IsInRoleAsync(user, roleName);

    }

}

1. Modificamos el **Program**:

builder.Services.AddDbContext<DataContext>(o =>

{

    o.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection"));

});

builder.Services.AddIdentity<User, IdentityRole>(cfg =>

{

    cfg.User.RequireUniqueEmail = true;

    cfg.Password.RequireDigit = false;

    cfg.Password.RequiredUniqueChars = 0;

    cfg.Password.RequireLowercase = false;

    cfg.Password.RequireNonAlphanumeric = false;

    cfg.Password.RequireUppercase = false;

}).AddEntityFrameworkStores<DataContext>();

builder.Services.AddTransient<SeedDb>();

builder.Services.AddScoped<ICombosHelper, CombosHelper>();

builder.Services.AddScoped<IBlobHelper, BlobHelper>();

builder.Services.AddScoped<IUserHelper, UserHelper>();

builder.Services.AddRazorPages().AddRazorRuntimeCompilation();

WebApplication? app = builder.Build();

SeedData(app);

void SeedData(WebApplication app)

{

    IServiceScopeFactory? scopedFactory = app.Services.GetService<IServiceScopeFactory>();

    using (IServiceScope? scope = scopedFactory.CreateScope())

    {

        SeedDb? service = scope.ServiceProvider.GetService<SeedDb>();

        service.SeedAsync().Wait();

    }

}

if (!app.Environment.IsDevelopment())

{

    app.UseExceptionHandler("/Home/Error");

    app.UseHsts();

}

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseRouting();

app.UseAuthentication();

app.UseAuthorization();

1. Modificamos el **SeedDb**:

public async Task SeedAsync()

{

    await \_context.Database.EnsureCreatedAsync();

    await CheckCountriesAsync();

    await CheckCategoriesAsync();

    await CheckProductsAsync();

    await CheckRolesAsync();

    await CheckUserAsync("1010", "Juan", "Zuluaga", "zulu@yopmail.com", "322 311 4620", "Calle Luna Calle Sol", UserType.Admin);

}

private async Task CheckProductsAsync()

{

    if (!\_context.Products.Any())

    {

        \_context.Products.Add(new Product

        {

            Description = "AirPods",

            Name = "AirPods",

            Price = 1300000M,

            Stock = 12F,

            ProductCategories = new List<ProductCategory>()

            {

                new ProductCategory { Category = await \_context.Categories.FirstOrDefaultAsync(c => c.Name == "Tecnología") }

            }

        });

        \_context.Products.Add(new Product

        {

            Description = "iPad",

            Name = "iPad",

            Price = 2500000M,

            Stock = 6F,

            ProductCategories = new List<ProductCategory>()

            {

                new ProductCategory { Category = await \_context.Categories.FirstOrDefaultAsync(c => c.Name == "Tecnología") }

            }

        });

        \_context.Products.Add(new Product

        {

            Description = "Mascarilla para la Cara",

            Name = "Mascarilla para la Cara",

            Price = 8000M,

            Stock = 200F,

            ProductCategories = new List<ProductCategory>()

            {

                new ProductCategory { Category = await \_context.Categories.FirstOrDefaultAsync(c => c.Name == "Belleza") }

            }

        });

        \_context.Products.Add(new Product

        {

            Description = "Camisa a Cuadros",

            Name = "Camisa a Cuadros",

            Price = 52000M,

            Stock = 24F,

            ProductCategories = new List<ProductCategory>()

            {

                new ProductCategory { Category = await \_context.Categories.FirstOrDefaultAsync(c => c.Name == "Ropa") }

            }

        });

        \_context.Products.Add(new Product

        {

            Description = "iPhone 13",

            Name = "iPhone 13",

            Price = 5200000M,

            Stock = 8F,

            ProductCategories = new List<ProductCategory>()

            {

                new ProductCategory { Category = await \_context.Categories.FirstOrDefaultAsync(c => c.Name == "Tecnología") }

            }

        });

    }

    await \_context.SaveChangesAsync();

}

private async Task<User> CheckUserAsync(

    string document,

    string firstName,

    string lastName,

    string email,

    string phone,

    string address,

    UserType userType)

{

    User user = await \_userHelper.GetUserAsync(email);

    if (user == null)

    {

        user = new User

        {

            FirstName = firstName,

            LastName = lastName,

            Email = email,

            UserName = email,

            PhoneNumber = phone,

            Address = address,

            Document = document,

            City = \_context.Cities.FirstOrDefault(),

            UserType = userType,

        };

        await \_userHelper.AddUserAsync(user, "123456");

        await \_userHelper.AddUserToRoleAsync(user, userType.ToString());

    }

    return user;

}

private async Task CheckRolesAsync()

{

    await \_userHelper.CheckRoleAsync(UserType.Admin.ToString());

    await \_userHelper.CheckRoleAsync(UserType.User.ToString());

}

1. Corremos los siguientes comandos:

PM> drop-database

PM> add-migration Users

PM> update-database

Fin