Consumo de archivos con API Rest en DOTNET

GitHub del repositorio: https://github.com/AndresAngarita10/ProyUpload

Contents

Desarrollo de la base de datos:

Entidades

Creación de la entidad FileUpload. Esta se encarga de almacenar la información de cada archivo, como el nombre, extensión, tamaño, ruta y tipo de archivo, ya que los archivos se guardaran en una ruta especifica y es a través de los registros en la base de datos que accederemos a ellos.

```
namespace Domain.Entities;

public class FileUpload : BaseEntity
{
    public string Name { get; set; }
    public string Extension { get; set; }
    public double Size { get; set; }
    public string Route { get; set; }
    public int TypeFileFk { get; set; }
    public TypeFile TypeFile { get; set; }
}
```

Creación de la entidad TypeFiles. Esta entidad se encarga de contener la información acerca de la clasificación del archivo, ejempl: imágenes, documentos, videos, etc.

```
namespace Domain.Entities;

public class TypeFile : BaseEntity
{
    public string Description { get; set; }
    public ICollection<FileUpload> FileUploads { get; set; }
}
```

Configuración de la entidad

Configuración de la entidad FileUpload para la creación de la DB, es importante destinar una longitud adecuada para los campos de nombre y ruta, ya que nombres o rutas demasiado largas generaran conflictos.

```
using Microsoft.EntityFrameworkCore;
using Domain.Entities;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
namespace Persistence.Data.Configuration;

public class FileUploadConfiguration : IEntityTypeConfiguration<FileUpload>
{
    public void Configure(EntityTypeBuilder<FileUpload> builder)
    {
        builder.ToTable("FileUpload");
        builder.HasKey(f => f.Id);


        builder.Property(f => f.Name)
        .HasColumnName("name")
        .HasColumnType("varchar")
        .IsRequired()
        .HasMaxLength(250);

        builder.Property(f => f.Extension)
        .HasColumnName("extension")
        .HasColumnType("varchar")
        .IsRequired()
        .HasMaxLength(10);

        builder.Property(f => f.Size)
        .HasColumnName("size")
        .HasColumnType("double")
        .IsRequired();

        builder.Property(f => f.Route)
        .HasColumnName("route")
```

```
        .HasColumnType("varchar")
        .IsRequired()
        .HasMaxLength(250);
    }
}
```

Configuración de la entidad TypeFiles para la creación de la DB

```
using Domain.Entities;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace Persistence.Data.Configuration;

public class TypeFilesConfiguration: IEntityTypeConfiguration<TypeFile>
{
    public void Configure(EntityTypeBuilder<TypeFile> builder)
    {
        builder.ToTable("typeFile");

        builder.HasKey(d => d.Id);

        builder.Property(d => d.Description)
        .HasColumnName("description")
        .HasColumnType("varchar")
        .IsRequired()
        .HasMaxLength(250);
    }
}
```

Conexión a la DB

Nombre de la base de datos filemanagement

Adición de la conexión en asppsettings.json

```
"ConnectionStrings": {
    "ConexSqlServer":"Data Source=localhost\\sqlexpress;Initial
catalog=db;Integrate Security=true",
    "ConexMySql":"server=localhost;user=root;password=123456;database=filema
nagementdb"
  }
```

Realizar migración

Se realiza la migración y la creación de la base de datos, posteriormente se verifica la adecuada creación de la DB

Crear migracion:

dotnet ef migrations add InitialCreateMig --project .\Persistence\ --startup-project .\API\ --output-dir .\Data\Migrations\

Crear Base de datos:

dotnet ef database update --project .\Persistence\ --startup-project .\API\

Creación de los endpoints

Creación de los métodos en la interfaz de la entidad

Metodo POST

Realizamos en la interfaz de FileUpload el método para el post el cual recibe un archivo desde la clase IFormFile, la cual es optima para el consumo de archivos

```
using Domain.Entities;
using Dominio.Interfaces;
using Microsoft.AspNetCore.Http;

namespace Domain.Interfaces;

public interface IFileUpload : IGenericRepo<FileUpload>
{
     public Task<FileUpload> FileUploadAsync(IFormFile file)
}
```

Creación de los métodos en el Repositorio de la entidad

Metodo POST

En este método post tenemos algunas validaciones, una de estas es la validación de la longitud del archivo o tamaño, también se realiza una división de la ubicación de almacenamiento de los archivos en función del tipo de archivo, en la sección de imágenes se incluye para el guardado una compresión para la optimización de recursos.

```
public async Task<FileUpload> FileUploadAsync(IFormFile file)
    {
```

```
        if (file == null || string.IsNullOrEmpty(file.FileName))
        {
            // Manejar el caso en el que file o file.FileName sean nulos
            return null;
        }
        string[] validExtensionsImg = { ".jpg", ".jpeg", ".png", ".gif",
".webp"  };
        string[] validDocumentExtensions = { ".pdf", ".doc", ".docx",
".txt", ".xlsx" };
        var extension = Path.GetExtension(file.FileName).ToLower();
        var filePath = "";
        int type = 0;
        if (validExtensionsImg.Contains(extension))
        {
            type = 1;
            string userFolder =
Environment.GetFolderPath(Environment.SpecialFolder.UserProfile);
            filePath = Path.Combine(userFolder, "Escritorio", "ProyUpload",
"BackEnd", "Persistence", "Data", "Archivos", "Img", file.FileName);
            Console.WriteLine("esste es el log de userfolder " +
userFolder);
            /*filePath =
"C:\\Users\\andre\\OneDrive\\Escritorio\\ProyUpload\\BackEnd\\Persistence\\D
ata\\Archivos\\Img\\" + file.FileName;*/
            filePath =
"D:\\Users\\dalgr\\Documents\\Campus\\Ciclo3\\NetCore\\ProyectoUploadFile\\P
royUpload\\BackEnd\\Persistence\\Data\\Archivos\\Img\\" + file.FileName;
            // Comprimir imágenes
            using (var memoryStream = new MemoryStream())
            {
                await file.CopyToAsync(memoryStream);
                var image = Image.FromStream(memoryStream);
                var quality = 50; // Ajusta la calidad de compresión según
tus necesidades
                var encoder = ImageCodecInfo.GetImageEncoders().First(c =>
c.FormatID == ImageFormat.Jpeg.Guid);
                var encoderParameters = new EncoderParameters(1);
                encoderParameters.Param[0] = new
EncoderParameter(Encoder.Quality, quality);
                image.Save(filePath, encoder, encoderParameters);
            }
        }
        else if (validDocumentExtensions.Contains(extension))
        {
            type = 2;
```

```
            string userFolder =
Environment.GetFolderPath(Environment.SpecialFolder.UserProfile);
            filePath = Path.Combine(userFolder, "Escritorio", "ProyUpload",
"BackEnd", "Persistence", "Data", "Archivos", "Doc", file.FileName);

            /* filePath =
"C:\\Users\\andre\\OneDrive\\Escritorio\\ProyUpload\\BackEnd\\Persistence\\D
ata\\Archivos\\Doc\\" + file.FileName; */
            filePath =
"D:\\Users\\dalgr\\Documents\\Campus\\Ciclo3\\NetCore\\ProyectoUploadFile\\P
royUpload\\BackEnd\\Persistence\\Data\\Archivos\\Doc\\" + file.FileName;
            using (var stream = File.Create(filePath))
            {
                await file.CopyToAsync(stream);
            }
        }
        else
        {
            return null;
        }
        double size = file.Length; //1.048.576
        size = size / 1048576;
        size = Math.Round(size, 2);

        FileUpload fileUpload = new FileUpload
        {
            Extension = Path.GetExtension(file.FileName).Substring(1),
            Name = Path.GetFileNameWithoutExtension(file.FileName),
            Size = size,
            Route = filePath,
            TypeFileFk = type
        };

        return fileUpload;
    }
```

Creación de los métodos en el Controller de la entidad

Metodo GET

Método GET genérico que obtener todos los archivos, de forma indistinta del tipo de archivo o extensión de este.

```
[HttpGet]
    [ProducesResponseType(StatusCodes.Status200OK)]
```

```
    [ProducesResponseType(StatusCodes.Status400BadRequest)]

    public async Task<ActionResult<IEnumerable<FileUploadDto>>> Get()
    {
        var files = await unitofwork.FileUploads.GetAllAsync();
        return mapper.Map<List<FileUploadDto>>(files);
    }
```

Método POST

método Post genérico distribuido con la lógica de 4 capas.

```
[HttpPost]
    [ProducesResponseType(StatusCodes.Status201Created)]
    [ProducesResponseType(StatusCodes.Status400BadRequest)]
    public async Task<ActionResult<IFormFile>> PostFiles(IFormFile file)
    {
        FileUpload fileObject = await
unitofwork.FileUploads.FileUploadAsync(file);
        if (fileObject == null)
        {
            return BadRequest();
        }
        var fileUpload = this.mapper.Map<FileUpload>(fileObject);
        if (fileUpload == null)
        {
            return BadRequest();
        }
        this.unitofwork.FileUploads.Add(fileUpload);
        await unitofwork.SaveAsync();
        fileObject.Id = fileUpload.Id;
        return CreatedAtAction(nameof(PostFiles), new { id = fileObject.Id
}, fileObject);
    }
```