Criação de Solução com ASP.NET Core 6 e Clean Architecture

ROTEIRO DE AULA PRÁTICA

Blank Soluticon – HotelSol
Class Library – HotelSol.Domain (entidades do domínio do app)
Class Library – HotelSol.Application (interfaces, serviços, regras de negócio)
Class Library – HotelSol.Infra.Data (acesso aos dados)
Class Library – HotelSol.Infra.IoC (centralização da injeção de dependência)
ASP.NET Core Web App – HotelSol.MVC (Interface do Usuário)

DEPENDÊNCIAS

HotelSol.Domain → sem dependência
HotelSol.Application → Domain
HotelSol.Infra.Data → Domain
HotelSol.Infra.IoC → Domain, Applicarion, Infra.Data
HotelSol.MVC → Infra.Data e Infra.IoC

PARA CRIAR A ESTRUTURA
`$ mkdir HotelSol` (Criar diretório para solução)
`$ cd HotelSol` (Ir para o diretório da solução)
`$ dotnet new sln --name HotelSol` (Criar arquivo de solução .net core)
`$ mkdir HotelSol.NomeProjeto` (Criar diretório para cada projeto da solução)
`$ dotnet new classlib` (Criar projetos dentro de cada diretório, exceto MVC)
Apagar as classes class1.cs dos projetos

CRIAR PROJETO MVC COM IDENTITY
```
$ dotnet new mvc --auth individual
```

ADICIONAR PROJETOS NA SOLUÇÃO (FAZER COM TODOS OS PROJETOS)
```
$ dotnet sln add HotelSol.Domain/HotelSol.Domain.csproj
```

CRIAR AS DEPENDÊNCIAS ENTRE OS PROJETOS (CONFORME DEFINIÇÃO ACIMA)
```
$ dotnet add reference ..path to csproj
```

FAZER RESTORE E BUILD DA SOLUÇÃO
```
$ dotnet restore
$ dotnet build
```

FAZER MIGRATION DO PROJETO MVC
```
$ dotnet ef migrations add initialcreate
$ dotnet ef database update
```

CRIAR AS ENTIDADES DO DOMÍNIO DO SISTEMA
```
criar pasta models/entities no projeto domain
dentro da pasta model, criar uma nova classe que representa um
modelo do sistema, conforme exemplo
```

```
using system;
using system.collections.generic;
using system.text;

namespace projectname.domain.models
{
    public class book
    {
        public int id { get; set; }
        public string name { get; set; }
        public string isbn { get; set; }
        public string authorname { get; set; }
    }
}
```

PREPARAR A CAMADA DE INFRA.DATA PARA CONEXÃO COM BANCO DE DADOS
Adicionar o pacotes necessário do EntityFramework Core

```
$ dotnet add package Microsoft.EntityFrameworkCore
$ dotnet add package Microsoft.EntityFrameworkCore.Design
$ dotnet add package Microsoft.EntityFrameworkCore.SqlServer
$ dotnet add package Microsoft.EntityFrameworkCore.Sqlite
$ dotnet add package Microsoft.EntityFrameworkCore.Tools
```

Criar uma pasta Context
Dentro da pasta Context criar uma classe para ProjetoDbContext, conforme exemplo abaixo

```
using Project.Domain.Models;
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Text;

namespace Project.Infra.Data.Context
{
    public class ProjectDbContext : DbContext
    {
        public ProjectDbContext(DbContextOptions options) :
          base(options) { }

        public DbSet<Model> Models { get; set; }

    }
}
```

REGISTRAR O NOVO DBCONTEXT NOS ARQUIVOS PROGRAM.CS E APPSETINGS.JSON DA CAMADA PROJETO.MVC

PROGRAM.CS

```
var connectionString2 =
builder.Configuration.GetConnectionString("ProjectConnection");
builder.Services.AddDbContext<ProjectDbContext>(options =>
    options.UseSqlite(connectionString2));
```

APPSETTINGS.JSON

```
"ConnectionStrings": {
    "DefaultConnection": "DataSource=app.db;Cache=Shared",

"ProjectConnection":"DataSource=../Project.Infra.Data/project.db;C
ache=Shared"
}
```

CRIAR A DBCONTEXT FACTORY

```
using Microsoft.Extensions.DependencyInjection;
using System;
using System.Collections.Generic;
using System.Text;
using TesteSol.Infra.Data.Context;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Design;
using Microsoft.Extensions.Configuration;


namespace TesteSol.Infra.Data
{
    public class LibraryDbContextFactory:
IDesignTimeDbContextFactory<LibraryDbContext>
    {
        public LibraryDbContext CreateDbContext(string[] args)
        {
            var optionsBuilder = new
DbContextOptionsBuilder<LibraryDbContext>();
            optionsBuilder.UseSqlite("Data Source=library.db");

            return new LibraryDbContext(optionsBuilder.Options);
        }
    }
}
```

FAZER A MIGRATION DO NOVO BANCO DE DADOS

```
$ dotnet ef migrations add InicialData -c ProjectDbContext
$ dotnet ef database update
```

CRIAR A PASTAS Interfaces, Services e ViewModel no projeto de Application

Criar a classe BookViewModel na pasta ViewModel para preparar os dados que serão enviados para a camada de visualização

```
using CleanArchitecture.Domain.Models;
using System;
using System.Collections.Generic;
using System.Text;

namespace CleanArchitecture.Application.ViewModels
{
    public class BookViewModel
    {
        public IEnumerable<Book> Books { get; set; }
    }
}
```

Sob a pasta interfaces, criar uma interface para retornar uma lista de objetos do ViewModel

```
using CleanArchitecture.Application.ViewModels;
using System;
using System.Collections.Generic;
using System.Text;

namespace CleanArchitecture.Application.Interfaces
{
    public interface IBookService
    {
        BookViewModel GetBooks();
    }
}
```

Criar o “Repository Design Patterns” para mediar o acesso entre o banco de dados e a camada Domain

Criar a pasta Interfaces no projeto Domain e adicionar a interface para repositório

```
using CleanArchitecture.Domain.Models;
using System;
using System.Collections.Generic;
using System.Text;

namespace CleanArchitecture.Domain.Interfaces
{
    public interface IBookRepository
    {
        IEnumerable<Book> GetBooks();
    }
}
```

Fazer a implementação de IbookService no projeto de application, na paste service, aproveite para fazer a injeção de dependência do repositório IBookRepository

```
using CleanArchitecture.Application.Interfaces;
using CleanArchitecture.Application.ViewModels;
using CleanArchitecture.Domain.Interfaces;
using System;
using System.Collections.Generic;
using System.Text;

namespace CleanArchitecture.Application.Services
{
    public class BookService : IBookService
    {
        public IBookRepository _bookRepository;
        public BookService(IBookRepository bookRepository)
        {
            _bookRepository = bookRepository;
        }
        public BookViewModel GetBooks()
        {
            throw new NotImplementedException();
        }
    }
}
```

Implementar o Repositório no projeto de Infra.Data criando a pasta Repositories e a classe BookRepository

```
using CleanArchitecture.Domain.Interfaces;
using CleanArchitecture.Domain.Models;
using CleanArchitecture.Infra.Data.Context;
using System;
using System.Collections.Generic;
using System.Text;

namespace CleanArchitecture.Infra.Data.Repositories
{
    public class BookRepository : IBookRepository
    {
        public LibraryDbContext _context;
        public BookRepository(LibraryDbContext context)
        {
            _context = context;
        }
        public IEnumerable<Book> GetBooks()
        {
            throw new NotImplementedException();
        }
    }
}
```

Implementar os métodos BookService e BookRepository
BookRepository

```
namespace TesteSol.Infra.Data.Repositories
{
    public class BookRepository: IBookRepository
    {
        public LibraryDbContext _context;
        public BookRepository(LibraryDbContext context)
        {
            _context = context;
        }
        public IEnumerable<Book> GetBooks(){
            return _context.Books;
        }
    }
}
```

BookService

```
public class BookService : IBookService
    {
        public IBookRepository _bookRepository;
        public BookService(IBookRepository bookRepository)
        {
            _bookRepository = bookRepository;
        }

        public BookViewModel GetBooks()
        {
            return new BookViewModel()
            {
                Books = _bookRepository.GetBooks()
            };
        }
    }
```

INVERSÃO DE CONTROLE
Adicionar o pacote DependencyInjection para centralizar e dividir nossas dependecias
Na pasta do projeto Infra.IoC, use o comando para adicionar o pacote

```
$ dotnet add package Microsoft.Extensions.DependencyInjection
```

Criar uma classe DependencyContainer para ser nosso container de dependencias sob o projeto Infra.IoC

```
using TesteSol.Application.Interfaces;
using TesteSol.Application.Services;
using TesteSol.Domain.Interfaces;
using TesteSol.Infra.Data.Repositories;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using System;
```

```
using System.Collections.Generic;
using System.Text;

namespace TesteSol.Infra.IoC
{
    public static class DependencyContainer
    {
        public static IServiceCollection RegisterServices(this
IServiceCollection services,
        IConfiguration configuration)
        {
            //TesteSol.Application
            services.AddScoped<IBookService, BookService>();

            //TesteSol.Domain.Interfaces |
TesteSol.Infra.Data.Repository
            services.AddScoped<IBookRepository, BookRepository>();

            return services;


        }
    }
}
```

Fazer a injeção de dependência no arquivo program.cs do Projeto.MVC

```
builder.Services.RegisterServices(builder.Configuration);
```

Adicionar dados no Banco de dados

Adicionar um controlador de modelo para a pasta Controllers no Projeto.MVC

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using TesteSol.Application.Interfaces;
using Microsoft.AspNetCore.Mvc;
using TesteSol.Application.ViewModels;

namespace TesteSol.MVC.Controllers
{
    public class BookController : Controller
    {
        private IBookService _bookService;
        public BookController(IBookService bookService)
        {
            _bookService = bookService;
        }
        public IActionResult Index()
        {
```

```
            BookViewModel model = _bookService.GetBooks();
            return View(model);
        }
    }
}
```

Na pasta View, criar a pasta Book e sob ela, criar um arquivo Razor View chamado Index.cshtml com o seguinte código

```
@model TesteSol.Application.ViewModels.BookViewModel

<table id="book" class="table table-bordered table-hover">
    <thead>
        <tr class="">
            <th>Name</th>
            <th>ISBN</th>
            <th>Author</th>
        </tr>
    </thead>
    <tbody>
        @foreach (var book in Model.Books)
        {
            <tr style="height: 60px;">
                <td>@book.Name</td>
                <td>@book.ISBN</td>
                <td>@book.AuthorName</td>
            </tr>
        }
    </tbody>
</table>
```

Em Views/Shared/_Layout.cshtml, adicionar um novo item de menu com o código abaixo

```
<li class="nav-item">
    <a class="nav-link text-dark" asp-area="" asp-
controller="Book" asp-action="Index">Books</a>
</li>
```

Fazer um build e testar a aplicação

Adicionar autenticação para acesso
No arquivo de Controller, adicionar a diretiva e o atributo [Authorize] no ação desejada

```
using Microsoft.AspNetCore.Authorization;

[Authorize]
public IActionResult Index()
{
    ...
}
```