PROJETO ACCOUNTOWNER

Prof. Ms. José Antonio Gallo Junior

1. Crie um Repositório com o nome **AccountOwner** no seu **GitHub.**

2. Clone o Repositório **AccountOwner** na **Área de Trabalho**.

3. Abra a pasta do Repositório **AccountOwner** no **Visual Studio Code**.

4. Crie as seguintes pastas no repositório:
   - **backend**
   - **database**
   - **frontend**
   - **tutorial**

Caso prefira a criação pode ser realizada pelo terminal, desde que esteja dentro da pasta do repositório:

```
mkdir backend
mkdir database
mkdir frontend
mkdir tutorial
```

5. Através deste link, faça o download da pasta **AccountOwnerFiles**, que possui os arquivos necessários ao desenvolvimento do projeto. Realize as seguintes ações após o download:
   a. Copie o arquivo **DbScript.sql** para a **database**;
   b. Copie o arquivo **AccountOwner.pdf** para a pasta **tutorial**.

6. No terminal, modifique o caminho atual acessando a pasta **backend**. Isso pode ser realizado com o comando:

```
cd backend
```

7. Digite os comandos abaixo na ordem apresentada, para criar a solução e os projetos necessários ao desenvolvimento do **backend** do projeto:

```
dotnet new solution --name AccountOwnerServer
dotnet new webapi --o AccountOwnerServer -f net6.0
dotnet sln add AccountOwnerServer\AccountOwnerServer.csproj
dotnet new classlib -o Contracts -f net6.0
dotnet sln add Contracts\Contracts.csproj
dotnet new classlib -o LoggerService -f net6.0
dotnet sln add LoggerService\LoggerService.csproj
dotnet new classlib -o Entities -f net6.0
dotnet sln add Entities\Entities.csproj
dotnet new classlib -o Repository -f net6.0
dotnet sln add Repository\Repository.csproj
```

8. Abra o arquivo **AccountOwnerServer.csproj** e altere a linha 5, desabilitando a **tag <Nullable>**:

```
<Nullable>disable</Nullable>
```

9. Adicione uma pasta com o nome **Extensions** na pasta do projeto **AccountOwnerServer**.

```
cd AccountOwnerServer
mkdir Extensions
```

10. Adicione na pasta **Extensions** uma classe com o nome **ServiceExtensions**, em seguida faça as alterações a classe conforme o código a seguir:

OBS: Para criar uma classe, clique com o botão direito do mouse sobre a pasta onde quer criar a classe e selecione a opção **New C#** e **Class.**

```
namespace AccountOwnerServer.Extensions;

public static class ServiceExtensions
{
    public static void ConfigureCors(this IServiceCollection services)
    {
        services.AddCors(options =>
        {
            options.AddPolicy("CorsPolicy",
                builder => builder.AllowAnyOrigin()
                .AllowAnyMethod()
                .AllowAnyHeader());
        });
    }

    public static void ConfigureIISIntegration(this IServiceCollection services)
    {
        services.Configure<IISOptions>(options =>
        {
        });
    }
}
```

11. Abra o arquivo **Program.cs** do projeto **AccountOwnerServer** e faça as alterações abaixo:

```
using AccountOwnerServer.Extensions;
using Microsoft.AspNetCore.HttpOverrides;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.ConfigureCors();
builder.Services.ConfigureIISIntegration();
```

```
builder.Services.AddControllers();

// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
    app.UseDeveloperExceptionPage();
}
else
    app.UseHsts();

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseForwardedHeaders(new ForwardedHeadersOptions
{
    ForwardedHeaders = ForwardedHeaders.All
});

app.UseCors("CorsPolicy");

app.UseAuthorization();

app.MapControllers();

app.Run();
```

12. Abra o arquivo **WeatherForecast.cs** e altere a linha 11 trocando **string?** por **string**, conforme mostra o código abaixo:

```
    public string Summary { get; set; }
```

13. Salve todos os arquivos e no terminal, confirmando que está na pasta **backend**, compile o projeto para verificar possíveis erros de desenvolvimento. Para isso execute o comando:

```
dotnet build
```

14. Abra a pasta do projeto **Contracts** e apague o arquivo **Class1.cs**

15. Abra o arquivo **Contracts.csproj** e altere a linha 6, desabilitando a **tag <Nullable>**:

```
<Nullable>disable</Nullable>
```

16. Crie no projeto **Contracts** uma interface com o nome **ILoggerManager** e faça as alterações abaixo:

OBS: Para criar uma interface, clique com o botão direito do mouse sobre a pasta onde quer criar a classe e selecione a opção **New C#** e **Interface.**

```
namespace Contracts;

public interface ILoggerManager
{
    void LogInfo(string message);
    void LogWarn(string message);
    void LogDebug(string message);
    void LogError(string message);
}
```

17. Abra a pasta do projeto **LoggerService** e apague o arquivo **Class1.cs**

18. Abra o arquivo **LoggerService.csproj** e altere a linha 6, desabilitando a **tag <Nullable>**:

```
<Nullable>disable</Nullable>
```

19. Abra o terminal, confira se o caminho está na pasta do **backend** e execute o comando abaixo para adicionar uma dependência no projeto principal (**AccountOwnerServer**) do projeto **LoggerService**:

```
dotnet add AccountOwnerServer\AccountOwnerServer.csproj reference
LoggerService\LoggerService.csproj
```

OBS: O comando acima deve estar em uma única linha quando for executado no terminal, se copiar e colar é provável que ocorra erro de execução, pois o terminal irá entender como dois comandos distintos**.**

20. Abra o terminal, confira se o caminho está na pasta do **backend** e execute o comando abaixo para adicionar uma dependência no projeto **LoggerService** do projeto **Contracts**:

```
dotnet add LoggerService\LoggerService.csproj reference Contracts\Contracts.csproj
```

21. Abra o terminal e execute os comandos abaixo para acessar a pasta do projeto **LoggerService** e instalar a biblioteca padrão de logs da **Microsoft NLog**:

```
cd LoggerService
dotnet add package NLog.Extensions.Logging --version 5.2.1
```

22. Abra o terminal e execute os comandos para voltar a pasta **backend** para compilar o projeto antes de continuar:

```
cd..
dotnet build
```

OBS: Caso não seja apresentada a mensagem **Complicação com êxito**, confira as mensagens em vermelho e reveja este documento para localizar o problema e corrigi-lo**.**

23. Crie no projeto **LoggerService** uma classe com o nome **LoggerManager** e faça as alterações abaixo:

```
using Contracts;
using NLog;

namespace LoggerService;

public class LoggerManager : ILoggerManager
{
    private static ILogger logger = LogManager.GetCurrentClassLogger();
    public void LogDebug(string message) => logger.Debug(message);
    public void LogError(string message) => logger.Error(message);
    public void LogInfo(string message) => logger.Info(message);
    public void LogWarn(string message) => logger.Warn(message);
}
```

24. Crie no projeto **AccountOwnerServer** um arquivo com o nome **nlog.config**, e faça as alterações abaixo:

OBS: Lembre-se de alterar os caminhos das pastas do projeto de acordo com a localização do projeto em seu computador. Mantenha os textos dos caminhos em uma única linha cada.

```
<?xml version="1.0" encoding="utf-8" ?>
<nlog xmlns="http://www.nlog-project.org/schemas/NLog.xsd"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      autoReload="true"
      internalLogLevel="Trace"
      internalLogFile="C:\Users\Gallo\Desktop\AccountOwner\backend\AccountOwnerServer\logs\
internal\internallog.txt">

  <targets>
    <target name="logfile" xsi:type="File"
            fileName="C:/Users/Gallo/Desktop/AccountOwner/backend/AccountOwnerServer/logs/
api/${shortdate}_logfile.txt"
            layout="${longdate} ${level:uppercase=true} ${message}"/>
  </targets>

  <rules>
    <logger name="*" minlevel="Debug" writeTo="logfile" />
  </rules>
</nlog>
```

25. Abra o arquivo **Extensions\ServiceExtensions.cs** do projeto **AccountOwnerServer** e adicione ao final classe o método abaixo:

```
public static void ConfigureLoggerService(this IServiceCollection services)
{
    services.AddSingleton<ILoggerManager, LoggerManager>();
}
```

OBS: Para o código acima não apresentar erros e funcionar corretamente é necessário adicionar ao começo do código as linhas abaixo:

```
using Contracts;
using LoggerService;
```

26. Abra o arquivo **Program.cs** do projeto **AccountOwnerServer** e realize a inclusão das linhas em destaque abaixo:

```
using AccountOwnerServer.Extensions;
using Microsoft.AspNetCore.HttpOverrides;
using NLog;

var builder = WebApplication.CreateBuilder(args);

LogManager.LoadConfiguration(string.Concat(Directory.GetCurrentDirectory(), "/nlog.config"));

// Add services to the container.

builder.Services.ConfigureCors();
builder.Services.ConfigureIISIntegration();
builder.Services.ConfigureLoggerService();

builder.Services.AddControllers();
```

27. Abra o arquivo **Controllers\WeatherForecastController.cs** do projeto **AccountOwnerServer** e faça as alterações abaixo:

```
using Contracts;
using Microsoft.AspNetCore.Mvc;

namespace AccountOwnerServer.Controllers;

[ApiController]
[Route("[controller]")]
public class WeatherForecastController : ControllerBase
{

    private readonly ILoggerManager _logger;
```

```
    public WeatherForecastController(ILoggerManager logger)
    {
        _logger = logger;
    }

    [HttpGet]
    public IEnumerable<string> Get()
    {
        _logger.LogInfo("Testando um log de informação a partir de um Controller.");
        _logger.LogDebug("Testando um log de debug a partir de um Controller.");
        _logger.LogWarn("Testando um log de aviso a partir de um Controller.");
        _logger.LogError("Testando um log de erro a partir de um Controller.");
        return new string[] { "value1", "value2" };
    }
}
```

28. No terminal execute os comandos na ordem apresentada, a partir do caminho da pasta do **backend**:

```
dotnet build
cd AccountOwnerServer
dotnet watch run
```

29. No navegador ao ser apresentada a documentação da **API** pelo **Swagger**, clique no método em azul **[GET] /WeatherForecast**, em seguida no botão **[Try it out]** (canto superior direito) e depois no botão azul **[Execute].** Será apresentado o **StatusCode 200** e um vetor com os **valores [“value1”, values2]**. Agora basta fechar o navegador, para a execução no terminal com as teclas **[Ctrl + C]**, e então verificar que dentro da pasta do projeto **AccountOwnerServer**, foi criada uma pasta **logs**,e nesta pasta outras duas pastas **api** e **internal**, cada uma contendo um arquivo de log especifico.

30. Abra a pasta do projeto **Entities** e apague o arquivo **Class1.cs**.

31. Abra o arquivo **Entities.csproj** e altere a linha 6, desabilitando a **tag <Nullable>**:

```
<Nullable>disable</Nullable>
```

32. Crie no projeto **Entities** uma pasta com o nome **Models** e outra com o nome **DataTransferObjects.**

OBS: Tome o cuidado de não criar uma pasta dentro da outra.

33. Crie no projeto **Entities** pasta **Models** uma classe com o nome **Owner**, e faça a codificação abaixo:

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace Entities.Models;

[Table("owner")]
public class Owner
{
```

```
    [Column("OwnerId")]
    public Guid Id { get; set; }

    [Required]
    [StringLength(60)]
    public string Name { get; set; }

    [Required]
    [DataType(DataType.Date)]
    public DateTime DateOfBirth { get; set; }

    [Required]
    [StringLength(100)]
    public string Address { get; set; }

    public ICollection<Account> Accounts { get; set; }
}
```

OBS: Ignore o erro que será apresentado na linha 23, informando que o projeto desconhece a classe **Account**.

34. Crie no projeto **Entities** pasta **Models** uma classe com o nome **Account**, e faça a codificação abaixo:

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace Entities.Models;

[Table("Account")]
public class Account
{
    [Column("AccountId")]
    public Guid Id { get; set; }

    [Required]
    public DateTime DateCreated { get; set; }

    [Required]
    public string AccountType { get; set; }

    [ForeignKey(nameof(Owner))]
    [Required]
    public Guid OwnerId { get; set; }
    public Owner Owner { get; set; }
}
```

35. Abra o terminal e execute os comandos abaixo para acessar a pasta do projeto **Entities** e instalar a biblioteca de manipulação de dados da **Microsoft** o **EntityFrameworkCore.**

```
cd Entities
dotnet add package Microsoft.EntityFrameworkCore --version 6.0.14
```

36. Crie no projeto **Entities** uma classe com o nome **RepositoryContext** e faça a codificação abaixo:

```
using Entities.Models;
using Microsoft.EntityFrameworkCore;

namespace Entities;

public class RepositoryContext : DbContext
{
    public RepositoryContext(DbContextOptions options) : base(options)
    {
    }

    public DbSet<Owner> Owners { get; set; }
    public DbSet<Account> Accounts { get; set; }
}
```

37. Abra o terminal e execute os comandos abaixo para acessar a pasta do projeto **AccountOwnerServer** e instalar a biblioteca de conexão a bancos de dados **MySQL** o **Pomelo**.

```
cd..
cd AccountOwnerServer
dotnet add package Pomelo.EntityFrameworkCore.MySql --version 6.0.2
```

38. Abra o arquivo **appsettings.json** do projeto **AccountOwnerServer** e faça as alterações abaixo:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Warning",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "mysqlconnection": {
    "connectionString":"server=localhost;port=3306;uid=root;pwd='';database=accountowner"
  },
  "AllowedHosts": "*"
}
```

39. Abra o terminal e execute os comandos abaixo para voltar a pasta do **backend**, onde está o arquivo da solução, e criar as **referências** necessárias entre os projetos.

```
cd..
dotnet add Contracts\Contracts.csproj reference Entities\Entities.csproj
dotnet add Repository\Repository.csproj reference Contracts\Contracts.csproj
dotnet add AccountOwnerServer\AccountOwnerServer.csproj reference
Repository\Repository.csproj
```

40. Salve todos os arquivos e no terminal, confirmando que está na pasta **backend**, compile o projeto para verificar possíveis erros de desenvolvimento. Para isso execute o comando:

```
dotnet build
```

41. Abra o arquivo **ServiceExtensions.cs** da pasta **Extensions** do projeto **AccountOwnerServer** e adicione ao final classe o método abaixo:

```
public static void ConfigureMySqlContext(this IServiceCollection services, IConfiguration
config)
{
    var connectionString = config["mysqlconnection:connectionString"];
    var serverVersion = ServerVersion.AutoDetect(connectionString);
    services.AddDbContext<RepositoryContext>(o =>
         o.UseMySql(connectionString, serverVersion));
}
```

OBS: Para o código acima não apresentar erros e funcionar corretamente é necessário adicionar ao começo do **usings** abaixo:

```
using Contracts;
using Entities;
using LoggerService;
using Microsoft.EntityFrameworkCore;
```

42. Abra o arquivo **Program.cs** do projeto **AccountOwnerServer** e adicione a linha em destaque abaixo:

```
builder.Services.ConfigureLoggerService();
builder.Services.ConfigureMySqlContext(builder.Configuration);

builder.Services.AddControllers();
```

43. Crie no projeto **Contracts** uma interface com o nome **IRepositoryBase** e faça a codificação abaixo:

```
using System.Linq.Expressions;
namespace Contracts;

public interface IRepositoryBase<T>
{
    IQueryable<T> FindAll();
    IQueryable<T> FindByCondition(Expression<Func<T, bool>> expression);
    void Create(T entity);
    void Update(T entity);
    void Delete(T entity);
}
```

44. Abra a pasta do projeto **Repository** e apague o arquivo **Class1.cs**.

45. Altere no arquivo **Repository.csproj** a linha 6 desabilitando o **Nullable**, como feito nos demais projetos.

46. Crie no projeto **Repository** uma classe com o nome **RepositoryBase** e faça a codificação abaixo:

```
using Contracts;
using Entities;
using Microsoft.EntityFrameworkCore;
using System.Linq.Expressions;
namespace Repository;

public abstract class RepositoryBase<T> : IRepositoryBase<T> where T : class
{
    protected RepositoryContext RepositoryContext { get; set; }
    public RepositoryBase(RepositoryContext repositoryContext)
    {
        RepositoryContext = repositoryContext;
    }

    public IQueryable<T> FindAll() => RepositoryContext.Set<T>().AsNoTracking();

    public IQueryable<T> FindByCondition(Expression<Func<T, bool>> expression) =>
        RepositoryContext.Set<T>().Where(expression).AsNoTracking();

    public void Create(T entity) => RepositoryContext.Set<T>().Add(entity);

    public void Update(T entity) => RepositoryContext.Set<T>().Update(entity);

    public void Delete(T entity) => RepositoryContext.Set<T>().Remove(entity);
}
```

47. Crie no projeto **Contracts** uma interface com o nome **IOwnerRepository** e faça a codificação abaixo:

```
using Entities.Models;
namespace Contracts;

public interface IOwnerRepository : IRepositoryBase<Owner>
{
}
```

48. Crie no projeto **Contracts** uma interface com o nome **IAccountRepository** e faça a codificação abaixo:

```
using Entities.Models;
namespace Contracts;

public interface IAccountRepository : IRepositoryBase<Account>
{
}
```

49. Crie no projeto **Repository** uma classe com o nome **OwnerRepository** e faça a codificação abaixo:

```
using Contracts;
using Entities;
using Entities.Models;
namespace Repository;

public class OwnerRepository : RepositoryBase<Owner>, IOwnerRepository
{
    public OwnerRepository(RepositoryContext repositoryContext) : base(repositoryContext)
    {
    }
}
```

50. Crie no projeto **Repository** uma classe com o nome **AccountRepository** e faça a codificação abaixo:

```
using Contracts;
using Entities;
using Entities.Models;

namespace Repository;

public class AccountRepository : RepositoryBase<Account>, IAccountRepository
{
    public AccountRepository(RepositoryContext repositoryContext) : base(repositoryContext)
    {
    }
}
```

51. Crie no projeto **Contracts** uma interface com o nome **IRepositoryWrapper** e faça a codificação abaixo:

```
namespace Contracts;

public interface IRepositoryWrapper
{
    IOwnerRepository Owner { get; }
    IAccountRepository Account { get; }
    void Save();
}
```

52. Crie no projeto **Repository** uma classe com o nome **RepositoryWrapper** e faça a codificação abaixo:

```
using Contracts;
using Entities;

namespace Repository;
```

```
public class RepositoryWrapper : IRepositoryWrapper
{
    private RepositoryContext _repoContext;
    private IOwnerRepository _owner;
    private IAccountRepository _account;

    public IOwnerRepository Owner
    {
        get
        {
            if (_owner == null)
            {
                _owner = new OwnerRepository(_repoContext);
            }

            return _owner;
        }
    }

    public IAccountRepository Account
    {
        get
        {
            if (_account == null)
            {
                _account = new AccountRepository(_repoContext);
            }

            return _account;
        }
    }

    public RepositoryWrapper(RepositoryContext repositoryContext)
    {
        _repoContext = repositoryContext;
    }

    public void Save()
    {
        _repoContext.SaveChanges();
    }
}
```

53. Abra o arquivo **ServiceExtensions.cs** da pasta **Extensions** do projeto **AccountOwnerServer** e adicione ao final classe o método abaixo:

```
    public static void ConfigureRepositoryWrapper(this IServiceCollection services)
    {
        services.AddScoped<IRepositoryWrapper, RepositoryWrapper>();
    }
```

OBS: Para o código acima não apresentar erros e funcionar corretamente é necessário adicionar o **using** em destaque abaixo:

```
using Microsoft.EntityFrameworkCore;
using Repository;
```

54. Abra o arquivo **Program.cs** do projeto **AccountOwnerServer** e adicione a linha em destaque abaixo:

```
builder.Services.ConfigureMySqlContext(builder.Configuration);
builder.Services.ConfigureRepositoryWrapper();

builder.Services.AddControllers();
```

55. Abra o arquivo **Controllers\WeatherForecastController.cs** do projeto **AccountOwnerServer** e faça as alterações abaixo:

```
using Contracts;
using Microsoft.AspNetCore.Mvc;

namespace AccountOwnerServer.Controllers;

[ApiController]
[Route("[controller]")]
public class WeatherForecastController : ControllerBase
{
    private IRepositoryWrapper _repository;

    public WeatherForecastController(IRepositoryWrapper repository)
    {
        _repository = repository;
    }

    [HttpGet]
    public IEnumerable<string> Get()
    {
        var domesticAccounts = _repository.Account
            .FindByCondition(x => x.AccountType.Equals("Domestic"));
        var owners = _repository.Owner.FindAll();
        return new string[] { "value1", "value2" };
    }
}
```

56. No Windows abra o **XAMPP** e clique no **[Start]** do **MySQL**. A partir deste ponto do desenvolvimento, as operações a serem testadas e executadas precisam do banco de dados do projeto, desta forma, o uso do **XAMPP** é obrigatório.

57. No Windows abra o **MySQL Workbench**, acesse o servidor **localhost**, abra e execute o arquivo **DbScript.sql** localizado na pasta **database** do projeto **AccountOwner**.

58. No terminal execute os comandos na ordem apresentada, a partir do caminho da pasta do **backend**:

```
dotnet build
cd AccountOwnerServer
dotnet watch run
```

59. No navegador ao ser apresentada a documentação da **API** pelo **Swagger**, clique no método em azul **[GET] /WeatherForecast**, em seguida no botão **[Try it out]** (canto superior direito) e depois no botão azul **[Execute].** Será apresentado o **StatusCode 200** e um vetor com os **valores ["value1", values2]**. Agora basta fechar o navegador, para a execução no terminal com as teclas **[Ctrl + C]**. Ainda é necessário implementar os métodos de exibição de dados do **RepositoryWrapper**.

60. Abra o arquivo **IOwnerRepository.cs** do projeto **Contracts** e faça a inclusão do método **GetAllOwners()** na interface **IOwnerRepository** conforme código abaixo:

```
public interface IOwnerRepository : IRepositoryBase<Owner>
{
    IEnumerable<Owner> GetAllOwners();
}
```

61. Abra o arquivo **OwnerRepository.cs** do projeto **Repository** e faça a implementação do método **GetAllOwners()** na classe **OwnerRepository**, conforme determina a interface **IOwnerRepository**, através da inclusão do código em destaque

```
using Contracts;
using Entities;
using Entities.Models;
namespace Repository;

public class OwnerRepository : RepositoryBase<Owner>, IOwnerRepository
{
    public OwnerRepository(RepositoryContext repositoryContext) : base(repositoryContext)
    {
    }

    public IEnumerable<Owner> GetAllOwners()
    {
        return FindAll()
            .OrderBy(ow => ow.Name)
            .ToList();
    }
}
```

62. Crie no projeto **AccountOwnerServer** na pasta **Controllers** um **API Controller** com o nome **OwnerController** e faça a codificação abaixo:

OBS: Para criar um **API Controller**, clique com o botão direito do mouse sobre a pasta onde quer criar o controlador classe e selecione a opção **New C#** e **API Controller.** Em seguida informe o nome do novo controlador e pressione **[Enter]**.

```
using Contracts;
using Microsoft.AspNetCore.Mvc;
namespace AccountOwnerServer.Controllers;

[Route("api/owner")]
[ApiController]
public class OwnerController : ControllerBase
{
    private ILoggerManager _logger;
    private IRepositoryWrapper _repository;

    public OwnerController(ILoggerManager logger, IRepositoryWrapper repository)
    {
        _logger = logger;
        _repository = repository;
    }

    [HttpGet]
    public IActionResult GetAllOwners()
    {
        try
        {
            var owners = _repository.Owner.GetAllOwners();
            _logger.LogInfo($"Retornando todos os owners do banco de dados.");
            return Ok(owners);
        }
        catch (Exception ex)
        {
            _logger.LogError($"Ocorreu um erro no método GetAllOwners: {ex.Message}");
            return StatusCode(500, "Erro Interno do Servidor");
        }
    }
}
```

63. Abra a pasta **Controllers** do **AccountOwnerServer** e apague o arquivo **WeatherForecastController.cs.**

64. Abra o terminal e execute o comando abaixo baixar e instalar o pacote da biblioteca **AutoMaper** no projeto **AccountOwnerServer**. Lembre-se de acessar a pasta no terminal antes de executar o comando:

```
dotnet add package AutoMapper.Extensions.Microsoft.DependencyInjection --version 12.0.0
```

65. Abra o arquivo **Program.cs** do projeto **AccountOwnerServer** e adicione a linha em destaque abaixo:

```
builder.Services.ConfigureRepositoryWrapper();

builder.Services.AddAutoMapper(typeof(Program));

builder.Services.AddControllers();
```

66. Crie uma classe na pasta **DataTransferObjects** do projeto **Entities** com o nome **OwnerDto** e faça a codificação abaixo:

```
namespace Entities.DataTransferObjects;

public class OwnerDto
{
    public Guid Id { get; set; }
    public string Name { get; set; }
    public DateTime DateOfBirth { get; set; }
    public string Address { get; set; }
}
```

67. Crie uma classe na pasta raiz do projeto **AccountOwnerServer** com o nome **MappingProfile**, e faça a codificação abaixo:

```
using AutoMapper;
using Entities.DataTransferObjects;
using Entities.Models;

public class MappingProfile : Profile
{
    public MappingProfile()
    {
        CreateMap<Owner, OwnerDto>();
    }
}
```

68. Abra o arquivo **OwnerController.cs** da pasta **Controllers** do projeto **AccountOwnerServer** e faça as alterações em destaque:

```
using AutoMapper;
using Contracts;
using Entities.DataTransferObjects;
using Microsoft.AspNetCore.Mvc;
namespace AccountOwnerServer.Controllers;

[Route("api/owner")]
[ApiController]
public class OwnerController : ControllerBase
{
    private ILoggerManager _logger;
    private IRepositoryWrapper _repository;
    private IMapper _mapper;

    public OwnerController(ILoggerManager logger, IRepositoryWrapper repository, IMapper
mapper)
    {
        _logger = logger;
```

```
        _repository = repository;
        _mapper = mapper;
    }

    [HttpGet]
    public IActionResult GetAllOwners()
    {
        try
        {
            var owners = _repository.Owner.GetAllOwners();

            _logger.LogInfo($"Retornando todos os owners do banco de dados.");

            var ownersResult = _mapper.Map<IEnumerable<OwnerDto>>(owners);
            return Ok(ownersResult);
        }
        catch (Exception ex)
        {
            _logger.LogError($"Ocorreu um erro no método GetAllOwners: {ex.Message}");
            return StatusCode(500, "Erro Interno do Servidor");
        }
    }
}
```

69. Abra o arquivo **IOwnerRepository.cs** do projeto **Contracts** e faça a inclusão do método **GetOwnerById()** na interface **IOwnerRepository** conforme código abaixo:

```
public interface IOwnerRepository : IRepositoryBase<Owner>
{
    IEnumerable<Owner> GetAllOwners();
    Owner GetOwnerById(Guid ownerId);
}
```

70. Abra o arquivo **OwnerRepository.cs** do projeto **Repository** e faça a implementação do método **GetOwnerById ()** na classe **OwnerRepository**, conforme determina a interface **IOwnerRepository**, através da inclusão do código em destaque:

```
    public IEnumerable<Owner> GetAllOwners()
    {
        return FindAll()
            .OrderBy(ow => ow.Name)
            .ToList();
    }

    public Owner GetOwnerById(Guid ownerId)
    {
        return FindByCondition(owner => owner.Id.Equals(ownerId))
                .FirstOrDefault();
    }
}
```

71. Abra o arquivo **OwnerController.cs** da pasta **Controllers** do projeto **AccountOwnerServer** e faça as alterações em destaque:

```
    [HttpGet]
    public IActionResult GetAllOwners()
    {
        try
        {
            var owners = _repository.Owner.GetAllOwners();

            _logger.LogInfo($"Retornando todos os owners do banco de dados.");

            var ownersResult = _mapper.Map<IEnumerable<OwnerDto>>(owners);
            return Ok(ownersResult);
        }

        catch (Exception ex)
        {
            _logger.LogError($"Ocorreu um erro no método GetAllOwners: {ex.Message}");
            return StatusCode(500, "Erro Interno do Servidor");
        }
    }
    [HttpGet("{id}")]
    public IActionResult GetOwnerById(Guid id)
    {
        try
        {
            var owner = _repository.Owner.GetOwnerById(id);

            if (owner is null)
            {
                _logger.LogError($"Owner com Id: {id}, não encontrado.");
                return NotFound();
            }
            else
            {
                _logger.LogInfo($"Retornando o owner com Id: {id}");

                var ownerResult = _mapper.Map<OwnerDto>(owner);
                return Ok(ownerResult);
            }
        }
        catch (Exception ex)
        {
            _logger.LogError($"Ocorreu um erro no método GetOwnerById: {ex.Message}");
            return StatusCode(500, "Erro Interno do Servidor");
        }
    }
}
```

72. Crie uma classe na pasta **DataTransferObjects** do projeto **Entities** com o nome **AccountDto** e faça a codificação abaixo:

```
namespace Entities.DataTransferObjects;

public class AccountDto
{
    public Guid Id { get; set; }
    public DateTime DateCreated { get; set; }
    public string AccountType { get; set; }
}
```

73. Abra a classe **OwnerDto**, localizada na pasta **DataTransferObjects** do projeto **Entities**, e faça a alteração abaixo:

```
namespace Entities.DataTransferObjects;

public class OwnerDto
{
    public Guid Id { get; set; }
    public string Name { get; set; }
    public DateTime DateOfBirth { get; set; }
    public string Address { get; set; }

    public IEnumerable<AccountDto> Accounts { get; set; }
}
```

74. Abra o arquivo **IOwnerRepository.cs** do projeto **Contracts** e faça a inclusão do método **GetOwnerWithDetails()** na interface **IOwnerRepository** conforme código abaixo:

```

public interface IOwnerRepository : IRepositoryBase<Owner>
{
    IEnumerable<Owner> GetAllOwners();
    Owner GetOwnerById(Guid ownerId);
    Owner GetOwnerWithDetails(Guid ownerId);
}
```

75. Abra o arquivo **OwnerRepository.cs** do projeto **Repository** e faça a implementação do método **GetOwnerWithDetails()** na classe **OwnerRepository**, conforme determina a interface **IOwnerRepository**, através da inclusão do código em destaque:

```
        return FindByCondition(owner => owner.Id.Equals(ownerId))
                .FirstOrDefault();
    }

    public Owner GetOwnerWithDetails(Guid ownerId)
    {
        return FindByCondition(owner => owner.Id.Equals(ownerId))
            .Include(ac => ac.Accounts)
            .FirstOrDefault();
    }
}
```

OBS: Para o código acima não apresentar erros é necessário adicionar o **using** abaixo no começo do arquivo:

```
using Microsoft.EntityFrameworkCore;
```

76. Abra o arquivo **MappingProfile.cs** do projeto **AccountOwnerServer** e faça a inclusão da linha em destaque conforme mostra o código abaixo:

```
    public MappingProfile()
    {
        CreateMap<Owner, OwnerDto>();

        CreateMap<Account, AccountDto>();
    }
```

77. Abra o arquivo **OwnerController.cs** da pasta **Controllers** do projeto **AccountOwnerServer** e faça as alterações em destaque:

```
        catch (Exception ex)
        {
            _logger.LogError($"Ocorreu um erro no método GetOwnerById: {ex.Message}");
            return StatusCode(500, "Erro Interno do Servidor");
        }
    }

    [HttpGet("{id}/account")]
    public IActionResult GetOwnerWithDetails(Guid id)
    {
        try
        {
            var owner = _repository.Owner.GetOwnerWithDetails(id);
            if (owner == null)
            {
                _logger.LogError($"Owner com Id: {id}, não encontrado.");
                return NotFound();
            }
            else
            {
                _logger.LogInfo($"Retornando o owner com detalhes e Id: {id}");

                var ownerResult = _mapper.Map<OwnerDto>(owner);
                return Ok(ownerResult);
            }
        }
        catch (Exception ex)
        {
            _logger.LogError($"Ocorreu um erro no método GetOwnerWithDetails: {ex.Message}");
            return StatusCode(500, "Erro Interno do Servidor");
        }
    }
}
```

78. Salve todos os arquivos e no terminal execute os comandos na ordem apresentada, a partir do caminho da pasta do **backend**, para testar as funcionalidades desenvolvidas até o momento:

```
dotnet build
cd AccountOwnerServer
dotnet watch run
```

79. Abra o arquivo **OwnerController.cs** da pasta **Controllers** do projeto **AccountOwnerServer** e faça a seguinte alteração:

```
    [HttpGet("{id}", Name = "OwnerById")]
    public IActionResult GetOwnerById(Guid id)
```

80. Crie uma classe na pasta **DataTransferObjects** do projeto **Entities** com o nome **OwnerForCreationDto** e faça a codificação abaixo:

```
using System.ComponentModel.DataAnnotations;
namespace Entities.DataTransferObjects;

public class OwnerForCreationDto
{
    [Required(ErrorMessage = "Campo obrigatório: Nome")]
    [StringLength(60, ErrorMessage = "O Nome não pode ter mais de 60 caracteres")]
    public string Name { get; set; }

    [Required(ErrorMessage = "Campo obrigatório: Data de Nascimento")]
    public DateTime DateOfBirth { get; set; }

    [Required(ErrorMessage = "Campo obrigatório: Endereço")]
    [StringLength(100, ErrorMessage = "O Endereço não pode ter mais de 100 caracteres")]
    public string Address { get; set; }
}
```

81. Abra o arquivo **IOwnerRepository.cs** do projeto **Contracts** e faça a inclusão do método **CreateOwner()** na interface **IOwnerRepository** conforme código abaixo:

```
public interface IOwnerRepository : IRepositoryBase<Owner>
{
    IEnumerable<Owner> GetAllOwners();
    Owner GetOwnerById(Guid ownerId);
    Owner GetOwnerWithDetails(Guid ownerId);
    void CreateOwner(Owner owner);
}
```

82. Abra o arquivo **OwnerRepository.cs** do projeto **Repository** e faça a implementação do método **CreateOwner()** na classe **OwnerRepository**, conforme determina a interface **IOwnerRepository**, através da inclusão do código em destaque:

```
    public Owner GetOwnerWithDetails(Guid ownerId)
    {
        return FindByCondition(owner => owner.Id.Equals(ownerId))
            .Include(ac => ac.Accounts)
            .FirstOrDefault();
    }

    public void CreateOwner(Owner owner)
    {
        Create(owner);
    }
}
```

83. Abra o arquivo **MappingProfile.cs** do projeto **AccountOwnerServer** e faça a inclusão da linha em destaque conforme mostra o código abaixo:

```
    public MappingProfile()
    {
        CreateMap<Owner, OwnerDto>();

        CreateMap<Account, AccountDto>();

        CreateMap<OwnerForCreationDto, Owner>();
    }
```

84. Abra o arquivo **OwnerController.cs** da pasta **Controllers** do projeto **AccountOwnerServer** e faça as alterações em destaque:

```
        catch (Exception ex)
        {
            _logger.LogError($"Ocorreu um erro no método GetOwnerWithDetails: {ex.Message}");
            return StatusCode(500, "Erro Interno do Servidor");
        }
    }

    [HttpPost]
    public IActionResult CreateOwner([FromBody] OwnerForCreationDto owner)
    {
        try
        {
            if (owner is null)
            {
                _logger.LogError("Objeto Owner enviado está nulo.");
                return BadRequest("Objeto Owner é nulo");
            }

            if (!ModelState.IsValid)
            {
                _logger.LogError("Objeto owner enviado é inválido.");
                return BadRequest("Objeto de modelo inválido");
            }
```

```
            var ownerEntity = _mapper.Map<Owner>(owner);

            _repository.Owner.CreateOwner(ownerEntity);
            _repository.Save();

            var createdOwner = _mapper.Map<OwnerDto>(ownerEntity);

            return CreatedAtRoute("OwnerById", new { id = createdOwner.Id }, createdOwner);
        }
        catch (Exception ex)
        {
            _logger.LogError($"Ocorreu um erro no método CreateOwner: {ex.Message}");
            return StatusCode(500, "Erro Interno do Servidor");
        }
    }
}
```

OBS: Para o código acima não apresentar erros é necessário adicionar o **using** abaixo no começo do arquivo:

```
using Entities.Models;
```

85. Crie uma classe na pasta **DataTransferObjects** do projeto **Entities** com o nome **OwnerForUpdateDto** e faça a codificação abaixo:

```
using System.ComponentModel.DataAnnotations;
namespace Entities.DataTransferObjects;

public class OwnerForUpdateDto
{
    [Required(ErrorMessage = "Campo obrigatório: Nome")]
    [StringLength(60, ErrorMessage = "O Nome não pode ter mais de 60 caracteres")]
    public string Name { get; set; }

    [Required(ErrorMessage = "Campo obrigatório: Data de Nascimento")]
    public DateTime DateOfBirth { get; set; }

    [Required(ErrorMessage = "Campo obrigatório: Endereço")]
    [StringLength(100, ErrorMessage = "O Endereço não pode ter mais de 100 caracteres")]
    public string Address { get; set; }
}
```

86. Abra o arquivo **MappingProfile.cs** do projeto **AccountOwnerServer** e faça a inclusão da linha em destaque conforme mostra o código abaixo:

```
        CreateMap<OwnerForCreationDto, Owner>();

        CreateMap<OwnerForUpdateDto, Owner>();
    }
}
```

87. Abra o arquivo **IOwnerRepository.cs** do projeto **Contracts** e faça a inclusão do método **UpdateOwner()** na interface **IOwnerRepository** conforme código abaixo:

```
public interface IOwnerRepository : IRepositoryBase<Owner>
{
    IEnumerable<Owner> GetAllOwners();
    Owner GetOwnerById(Guid ownerId);
    Owner GetOwnerWithDetails(Guid ownerId);
    void CreateOwner(Owner owner);
    void UpdateOwner(Owner owner);
}
```

88. Abra o arquivo **OwnerRepository.cs** do projeto **Repository** e faça a implementação do método **UpdateOwner ()** na classe **OwnerRepository**, conforme determina a interface **IOwnerRepository**, através da inclusão do código em destaque:

```
    public void CreateOwner(Owner owner)
    {
        Create(owner);
    }

    public void UpdateOwner(Owner owner)
    {
        Update(owner);
    }
}
```

89. Abra o arquivo **OwnerController.cs** da pasta **Controllers** do projeto **AccountOwnerServer** e faça as alterações em destaque:

```
            _logger.LogError($"Ocorreu um erro no método CreateOwner: {ex.Message}");
            return StatusCode(500, "Erro Interno do Servidor");
        }
    }

    [HttpPut("{id}")]
    public IActionResult UpdateOwner(Guid id, [FromBody] OwnerForUpdateDto owner)
    {
        try
        {
            if (owner is null)
            {
                _logger.LogError("Objeto Owner enviado está nulo.");
                return BadRequest("Objeto Owner é nulo");
            }

            if (!ModelState.IsValid)
            {
                _logger.LogError("Objeto owner enviado é inválido.");
                return BadRequest("Objeto de modelo inválido");
            }
```

```
            var ownerEntity = _repository.Owner.GetOwnerById(id);
            if (ownerEntity is null)
            {
                _logger.LogError($"Owner com Id: {id}, não encontrado.");
                return NotFound();
            }

            _mapper.Map(owner, ownerEntity);

            _repository.Owner.UpdateOwner(ownerEntity);
            _repository.Save();

            return NoContent();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Ocorreu um erro no método UpdateOwner: {ex.Message}");
            return StatusCode(500, "Erro Interno do Servidor");
        }
    }
}
```

90. Abra o arquivo **IOwnerRepository.cs** do projeto **Contracts** e faça a inclusão do método **DeleteOwner()** na interface **IOwnerRepository** conforme código abaixo:

```
public interface IOwnerRepository : IRepositoryBase<Owner>
{
    IEnumerable<Owner> GetAllOwners();
    Owner GetOwnerById(Guid ownerId);
    Owner GetOwnerWithDetails(Guid ownerId);
    void CreateOwner(Owner owner);
    void UpdateOwner(Owner owner);
    void DeleteOwner(Owner owner);
}
```

91. Abra o arquivo **OwnerRepository.cs** do projeto **Repository** e faça a implementação do método **DeleteOwner ()** na classe **OwnerRepository**, conforme determina a interface **IOwnerRepository**, através da inclusão do código em destaque:

```
    public void UpdateOwner(Owner owner)
    {
        Update(owner);
    }

    public void DeleteOwner(Owner owner)
    {
        Delete(owner);
    }
}
```

92. Abra o arquivo **IAccountRepository.cs** do projeto **Contracts** e faça a inclusão do método **AccountsByOwner()** na interface **IAccountRepository** conforme código abaixo:

```
public interface IAccountRepository : IRepositoryBase<Account>
{
    IEnumerable<Account> AccountsByOwner(Guid ownerId);
}
```

93. Abra o arquivo **AccountRepository.cs** do projeto **Repository** e faça a implementação do método **AccountsByOwner ()** na classe **AccountRepository**, conforme determina a interface **IAccountRepository**, através da inclusão do código em destaque:

```
public class AccountRepository : RepositoryBase<Account>, IAccountRepository
{
    public AccountRepository(RepositoryContext repositoryContext) : base(repositoryContext)
    {
    }

    public IEnumerable<Account> AccountsByOwner(Guid ownerId)
    {
        return FindByCondition(a => a.OwnerId.Equals(ownerId)).ToList();
    }
}
```

94. Abra o arquivo **OwnerController.cs** da pasta **Controllers** do projeto **AccountOwnerServer** e faça as alterações em destaque:

```
        {
            _logger.LogError($"Ocorreu um erro no método UpdateOwner: {ex.Message}");
            return StatusCode(500, "Erro Interno do Servidor");
        }
    }

    [HttpDelete("{id}")]
    public IActionResult DeleteOwner(Guid id)
    {
        try
        {
            var owner = _repository.Owner.GetOwnerById(id);
            if (owner == null)
            {
                _logger.LogError($"Owner com Id: {id}, não encontrado.");
                return NotFound();
            }

            if(_repository.Account.AccountsByOwner(id).Any())
            {
                _logger.LogError($"O owner com id: {id}, não pode ser excluído, pois possuir
contas relacionadas. Exclua as contas primeiro.");
                return BadRequest("Não é possível excluir o owner. Possui contas
relacionadas. Exclua as contas primeiro.");
            }
```

```
            _repository.Owner.DeleteOwner(owner);
            _repository.Save();

            return NoContent();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Ocorreu um erro no método DeleteOwner: {ex.Message}");
            return StatusCode(500, "Erro Interno do Servidor");
        }
    }
}
```

95. Salve todos os arquivos e no terminal execute os comandos na ordem apresentada, a partir do caminho da pasta do **backend**, para testar as funcionalidades desenvolvidas até o momento:

```
dotnet build
cd AccountOwnerServer
dotnet watch run
```

IMPORTANTE: Verifique todas as ações disponibilizadas pelo **OwnerController**, testando pesquisas, inclusão, alteração e exclusão. Para desenvolvimento de todas as ações do **OwnerController** começamos no **passo 60 e finalizamos no 94**, desta forma, para criar o **AccountController**, é necessário repetir esses passos fazendo as alterações de classes e interfaces de **Owner** para **Account**.

IMPORTANTE: A próxima parte deste material irá focar no desenvolvimento do front-end utilizando o **framework Javascript ANGULAR**.