Passo a passo

1. Criar um projeto/clonar um repositório
    a. Solução em branco
    b. Selecionar um nome
2. Clicar com o direito em solução
    a. Adicionar
    b. Novo projeto
    c. API web do asp.net CORE
    d. Nome: nome_do_projeto.API
3. Clicar com o direito em solução
    a. Adicionar
    b. Novo projeto
    c. Biblioteca de classes
    d. Nome: nome_do_projeto.Services / nome_do_projeto.Repositories / nome_do_projeto.Domain /

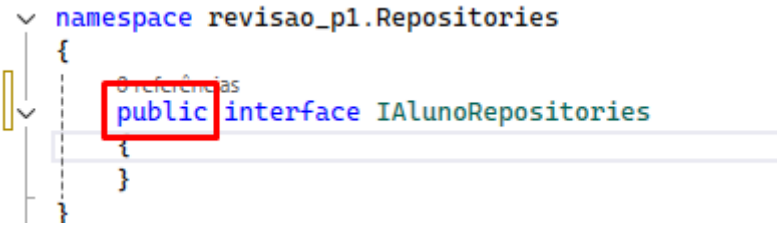
OBS: Renomear as classes pelo direito+renomear, para alterar também o nome da classe (Ex: AlunoDomain.cs)

4. **Adicionar as regras de negócio dentro do Domain**

```
namespace revisao_p1.Domain
{
    1 referência
    public class AlunoDomain
    {
        1 referência
        public string Nome { get; private set; } //setando a variável nome
        2 referências
        public decimal Nota { get; private set; } //setando a variável nota

        private const decimal NOTA_APROVACAO = 7.0m; //setando a constante da nota de
                                                     //aprovação para posterior comparação

        0 referências
        public AlunoDomain(string nome, decimal nota) //construtor da classe,
                                                      //PRECISA ter o mesmo nome da classe
                                                      //(AlunoDomain)
        {
            if (string.IsNullOrWhiteSpace(nome)) //verifica se a string nome é nula ou vazia
                throw new ArgumentException("O nome é obrigatório"); // exibe erro caso sim

            if (nota < 0 || nota > 10) // verifica se a nota é menor do que zero ou maior do que dez
                throw new ArgumentException("Nota inválida"); //exibe erro caso sim

            Nome = nome; //diz que os parâmetros usados são as variáveis setadas anteriormente
            Nota = nota; //diz que os parâmetros usados são as variáveis setadas anteriormente
        }
        0 referências
        public bool Aprovado() //outra função, para definir se o aluno está aprovado
        {
            return Nota >= NOTA_APROVACAO; //retorna true se a nota for maior ou igual
                                           //a variável setada e false, caso não seja
        }
    }
}
```

5. **Criar interface do Repositorie**

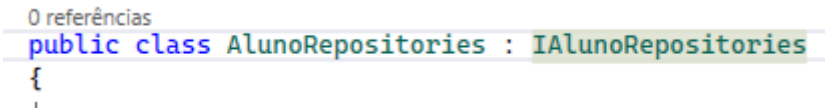
a. Clicar com o direito na biblioteca do repositorie
b. Adicionar > novo item
c. Interface (INome_do_repositorie)
d. Alterar a interface para “public”



e. Descrever os métodos do repositorie na interface

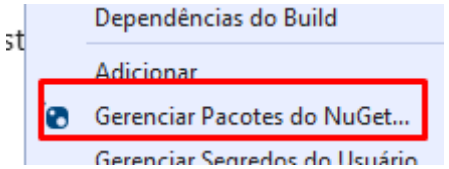
```
using revisao_p1.Domain;

namespace revisao_p1.Repositories
{
    0 referências
    public interface IAlunoRepositories
    {
        0 referências
        void Salvar(AlunoDomain aluno); //função para salvar/criar que não
                                        //retorna nada (void) e tem como
                                        //parâmetro o tipo AlunoDomain
        0 referências
        List<AlunoDomain> ObterAlunos(); //função que vai retornar um array
                                         //de objetos do tipo AlunoDomain

        0 referências
        AlunoDomain ObterporID(int id); //função que vai retornar um objeto do
                                        //tipo AlunoDomain quando buscado pelo
                                        //parâmetro id

        0 referências
        void Atualizar(AlunoDomain aluno); //função para atualizar e que não retorna
                                           //nada (void) e tem como parâmetro o tipo
                                           //AlunoDomain

        0 referências
        void Remover(int id); //função para remover e que não retorna nada (void) e
                              //tem como parâmetro o id

    }
}
```

6. **Desenvolver a classe Repositories com base no contrato da interface**



a. Passar o mouse sobre “IAlunoRepositories” e selecionar a opção “Mostrar possíveis correções” > “Implementar a interface”

7. **Conectar no banco:**
   a. Adicionar nova classe na classe de repositories: Clicar com o direito > novo item > classe
   b. Alterar o nome para “AppDbContext”
   c. Clicar com o direito na biblioteca > Gerenciar pacotes do NuGet



d. Procurar > Microsoft.EntityFrameworkCore > Instalar
e. Microsoft.EntityFrameworkCore.InMemory > Instalar

```
namespace revisao_p1.Repositories
{
    2 referências
    public class AppDbContext : DbContext //herdar da classe DbContext
    {
        private readonly DbContext _context; //crio variável _context
        0 referências
        public AppDbContext(DbContextOptions<AppDbContext> context): base(context){} //Construtor
        0 referências
        public DbSet<AlunoDomain> Alunos { get; set; } // Alunos é a tabela e o DbSet diz como a
                                                       // tabela vai ser estruturada, com as propriedades do AlunoDomain

        0 referências
        protected override void OnModelCreating(ModelBuilder modelBuilder) //método para especificar o tipo de cada
                                                                            //coluna da tabela
        {
            modelBuilder.Entity<AlunoDomain>(entity =>
            {
                entity.HasKey(u => u.Id); //tipificada por id
            });
        }
```

```
using revisao_p1.Domain;

namespace revisao_p1.Repositories
{
    1 referência
    public class AlunoRepositories : IAlunoRepositories
    {
        private readonly AppDbContext _context;//criando a variável pro contexto
                                               //do banco de dados
        0 referências
        public AlunoRepositories(AppDbContext context) //construindo a classe
        {
            _context = context; //atribuo o contexto na variável interna
        }
        2 referências
        public void Atualizar(int id, AlunoDomain aluno) //função para atualizar por id
        {
            var a = ObterporID(id); //guarda o valor obtido pelo id em uma variável
            a.Nota = aluno.Nota; //a variável obtida irá receber uma nota/nome
            a.Nome = aluno.Nome;
            _context.SaveChanges(); //salvar as mudanças no banco
        }

        1 referência
        public List<AlunoDomain> ObterAlunos() //função para obter um array com todos os alunos
        {
            return _context.Alunos.ToList(); //retorna a lista de alunos do banco
        }

        3 referências
        public AlunoDomain ObterporID(int id) //função para buscar o aluno pelo id
        {
            var aluno = _context.Alunos.FirstOrDefault(a => a.Id == id); //busco no banco
                                                                         //o valor correspondente ao id e
                                                                         //guardo em uma variável
            if (aluno == null) //se o valor for nulo, aviso que não existe
            {
                throw new Exception("Aluno não encontrado");
            }
            return aluno; //se o id for válido, retorno o aluno
        }
```

```
    1 referência
    public void Remover(int id) //função para remover um aluno
    {
        var aluno = ObterporID(id); //uso o método obter por id, já que a
                                    //função tem parâmetro id e guardo em uma variável
        _context.Alunos.Remove(aluno); //utilizo o método Remove, na variável
                                       //que foi encontrada pelo id
        _context.SaveChanges(); //salvo as alterações no banco
    }

    1 referência
    public void Salvar(AlunoDomain aluno) //função para salvar/adicionar no banco
    {
        _context.Alunos.Add(aluno); //utilizo o método Add para adicionar um
                                    //novo aluno na minha tabela Alunos
        _context.SaveChanges(); //salvo as alterações no banco
    }
}
```

8. **Fazer a interface do Services:**

```
using revisao_p1.Domain;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace revisao_p1.Services
{
    1 referência
    public interface IAlunoServices
    {
        1 referência
        void Atualizar(int id, decimal nota, string nome); //função para atualizar um aluno,
                                                           //não retorna nada
        1 referência
        void Salvar(decimal nota, string nome); //função para criar um aluno, não retorna nada
        1 referência
        void Remover(int id); //função para remover um aluno, não retorna nada
        1 referência
        List<AlunoDomain> ObterTodos(); //função para visualizar todos os alunos, retorna um
                                        //array com todos os objetos do tipo AlunoDomain


        1 referência
        AlunoDomain ObterPorId(int id); //função que retorna um AlunoDomain com base em uma
                                        //busca por id
```

9. **Fazer o services:**

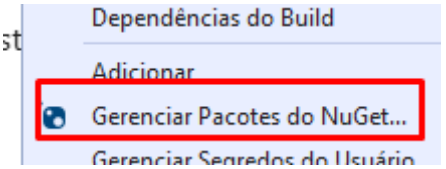
```
using revisao_p1.Domain;
using revisao_p1.Repositories;

namespace revisao_p1.Services
{
    1 referência
    public class AlunoServices : IAlunoServices
    {
        private readonly IAlunoRepositories _repository;
        0 referências
        public AlunoServices(IAlunoRepositories alunoRepositories) //construtor
        {
            _repository = alunoRepositories; //faz com que a variável interna receba
                                             //o parâmetro de AlunoServices
        }
        1 referência
        public void Atualizar(int id, decimal nota, string nome) //função de atualizar
        {
            var aluno = new AlunoDomain(nome, nota); //variável que guarda um valor do tipo
                                                     //AlunoDomain, utilizando os parâmetros
                                                     //de nome e nota
            _repository.Atualizar(id, aluno); //utiliza o método atualizar com o id e a
                                              //variável criada pelo tipo AlunoDomain
        }

        1 referência
        public AlunoDomain ObterPorId(int id) //função que utiliza o método ObterporId
                                              //do IAlunoRepositories
        {
            return _repository.ObterporID(id);
        }

        1 referência
        public List<AlunoDomain> ObterTodos() //função que utiliza o método ObterAlunos
                                              //do IAlunoRepositories
        {
          return _repository.ObterAlunos();
        }
```

```
    1 referência
    public void Remover(int id) //função que utiliza o método Remover do
                                //IAlunoRepositories
    {
        _repository.Remover(id);
    }

    1 referência
    public void Salvar(decimal nota, string nome) //função que utiliza o
                                                  //método Salvar do IAlunoRepositories
    {
        var aluno = new AlunoDomain(nome, nota); //crio uma variável que recebe o AlunoDomain
        _repository.Salvar(aluno); //utilizo o método salvar com a variável criada
    }
```

10. API

a. Instalar o NuGet na API

b. Clicar com o direito na API > Gerenciar pacotes do NuGet



c. Procurar > Microsoft.EntityFrameworkCore > Instalar

d. Microsoft.EntityFrameworkCore.InMemory > Instalar
e. Inserir injeção de dependência no arquivo “Program.cs”

```
builder.Services.AddControllers();
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();
builder.Services.AddScoped<IAlunoRepositories, AlunoRepositories>();
builder.Services.AddScoped<IAlunoServices, AlunoServices>();
```

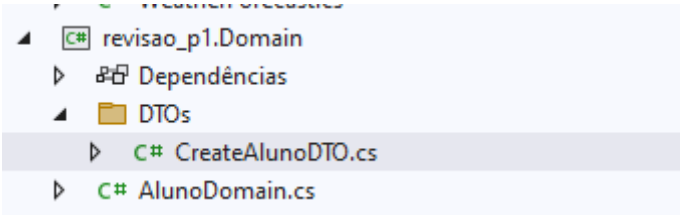
f. Configurar o banco in memory no mesmo arquivo

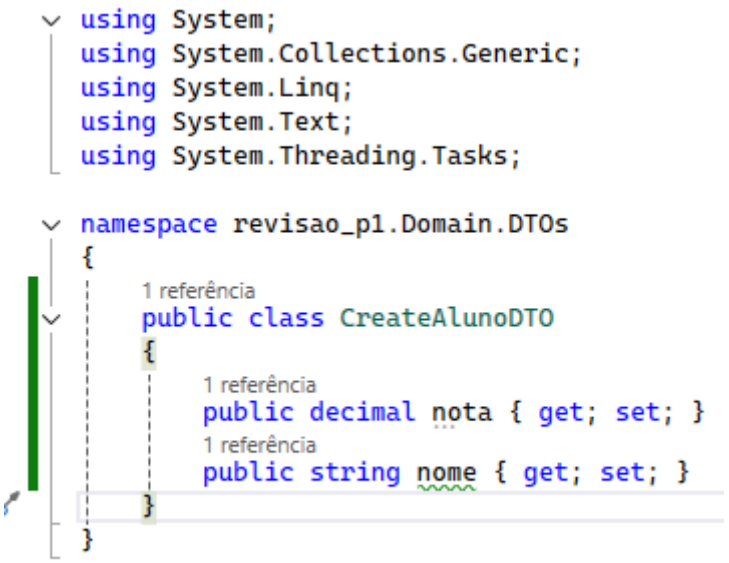
```
builder.Services.AddDbContext<AppDbContext>(options => options.UseInMemoryDatabase("Alunos"));
```

g. Criar o controller na pasta controllers
    i. Adicionar > Novo item (nomeController)

```
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using revisao_p1.Domain;
using revisao_p1.Domain.DTOs;
using revisao_p1.Services;

namespace revisao_p1.API.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    1 referência
    public class AlunoController : ControllerBase
    {
        private readonly IAlunoServices _alunoService; //criar variável interna do tipo IAlunoServices
                                                       //arq: controller > service > repository > banco
        0 referências
        public AlunoController(IAlunoServices alunoService) { //construtor da classe

            _alunoService = alunoService; //a variável INTERNA recebe o parâmetro do construtor
        }
        [HttpGet("{id}")] //tipo de requisição http
        0 referências
        public IActionResult ObterPorId(int id) //sempre um IActionResult, depois o nome da função e seu parâmetro
        {
            var aluno = _alunoService.ObterPorId(id); //coloco o resultado em uma variável para depois conseguir retornar
            return Ok(aluno); //resultado da requisição (200 ok) e a variável com o valor. Get sempre vai ter um return
        }
        [HttpGet] //tipo de requisição http
        0 referências
        public IActionResult ObterTodos() //sempre um IActionResult, depois o nome da função
        {
            var todos = _alunoService.ObterTodos(); //coloco o resultado em uma variável para depois conseguir retornar
            return Ok(todos); //resultado da requisição (200 ok) e a variável com o valor. Get sempre vai ter um return
        }
        [HttpDelete("{id}")] //tipo de requisição http
        0 referências
        public IActionResult Remover(int id) //sempre um IActionResult, depois o nome da função e o parâmetro
        {
            _alunoService.Remover(id); //chamando a função com o parâmetro
            return Ok(); //resultado da requisição (200 ok), nesse caso não temos algo para retornar
        }
    }
    [HttpPatch] //tipo de requisição http
    0 referências
    public IActionResult Atualizar([FromBody] AlunoDomain Aluno) //sempre um IActionResult,
                                                                 //depois o nome da função e o parâmetro.
                                                                 //Nesse caso, o parâmetro virá do domain, onde as
                                                                 //variáveis foram criadas. Deverá buscar o parâmetro
                                                                 //do corpo da requisição
    {
        _alunoService.Atualizar(Aluno.Id, Aluno.Nota, Aluno.Nome); //chamando a função, utilizando o parâmetro de acordo
                                                                   //com a variável Aluno, que puxa as variáveis do Domain
        return Ok(); //resultado da requisição (200 ok), nesse caso não temos algo para retornar
    }
    [HttpPost("create")] //tipo de requisição http
    0 referências
    public IActionResult Salvar([FromBody] CreateAlunoDTO Aluno) //sempre um IActionResult,
                                                                 //depois o nome da função e o parâmetro.
                                                                 //Nesse caso, o parâmetro virá do CreateAlunoDTO, onde as
                                                                 //variáveis foram criadas. Deverá buscar o parâmetro
                                                                 //do corpo da requisição
    {
        _alunoService.Salvar(Aluno.nota, Aluno.nome); //chamando a função, utilizando o parâmetro de acordo
                                                      //com a variável Aluno, que puxa as variáveis do CreateAlunoDTO
        return Ok(); //resultado da requisição (200 ok), nesse caso não temos algo para retornar
    }
```

- CreateAlunoDTO: Foi criado como uma classe dentro de uma pasta DTO, que estava dentro do Domain, para guardar as variáveis que seriam utilizadas no controller





```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace revisao_p1.Domain.DTOs
{
    1 referência
    public class CreateAlunoDTO
    {
        1 referência
        public decimal nota { get; set; }
        1 referência
        public string nome { get; set; }
    }
}
```