Clean Architecture .NET 8 – API de Colaboradores + Scripts (Cache em Memória)

Solução exemplo com Clean Architecture, SOLID, Repository Pattern, Refit + Polly (retry/circuit-breaker/timeout), cache em memória (IMemoryCache) e testes unitários e de integração.

> **Cenário:** - Buscar colaboradores em uma API A e scripts de perguntas por cargo em uma API B. - Gravar os dados em cache em memória para consultas posteriores via endpoints internos. - Endpoints internos: - `GET /colaboradores` - `GET /colaboradores/{cpf}` - `GET /scripts/{cargo}` - `GET /colaboradores/{cpf}/scripts` (atalho que deriva o cargo pelo CPF) - Refresh automático do cache via Hosted Service + TTL configurável.

Estrutura de Pastas

```
src/
  Company.CollabScripts.Api/              # Web API (apresentação)
  Company.CollabScripts.Application/      # Casos de uso, portas
(interfaces)
  Company.CollabScripts.Domain/           # Entidades, VOs, regras básicas
  Company.CollabScripts.Infrastructure/   # Adapters: Refit + Polly, cache,
repos

tests/
  Company.CollabScripts.UnitTests/
  Company.CollabScripts.IntegrationTests/
```

Domain

`Domain/Entities/Colaborador.cs`

```csharp
namespace Company.CollabScripts.Domain.Entities;

public sealed class Colaborador
{
    public string Nome { get; }
    public string Cpf { get; } // como string para preservar zeros à esquerda
    public string Cargo { get; }
    public string Email { get; }
    public StatusColaborador Status { get; }

    public Colaborador(string nome, string cpf, string cargo, string email,
StatusColaborador status)
```

```
    {
        Nome = nome;
        Cpf = cpf;
        Cargo = cargo;
        Email = email;
        Status = status;
    }
}
```

`Domain/Entities/ScriptPerguntas.cs`

```
namespace Company.CollabScripts.Domain.Entities;

public sealed class ScriptPerguntas
{
    public string Cargo { get; }
    public IReadOnlyList<string> Perguntas { get; }
    public ScriptPerguntas(string cargo, IEnumerable<string> perguntas)
    {
        Cargo = cargo;
        Perguntas = perguntas.ToList().AsReadOnly();
    }
}
```

`Domain/Enums/StatusColaborador.cs`

```
namespace Company.CollabScripts.Domain.Entities;

public enum StatusColaborador
{
    Ativo = 1,
    Inativo = 2
}
```

Application (Portas e Casos de Uso)

Portas (interfaces)

`Application/Abstractions/IColaboradorReadRepository.cs`

```
using Company.CollabScripts.Domain.Entities;

namespace Company.CollabScripts.Application.Abstractions;
```

```
public interface IColaboradorReadRepository
{
    Task<IReadOnlyList<Colaborador>> ListAsync(CancellationToken ct);
    Task<Colaborador?> GetByCpfAsync(string cpf, CancellationToken ct);
}
```

`Application/Abstractions/IScriptReadRepository.cs`

```
using Company.CollabScripts.Domain.Entities;

namespace Company.CollabScripts.Application.Abstractions;

public interface IScriptReadRepository
{
    Task<ScriptPerguntas?> GetByCargoAsync(string cargo, CancellationToken
ct);
}
```

`Application/Abstractions/IExternalSyncService.cs`

```
namespace Company.CollabScripts.Application.Abstractions;

public interface IExternalSyncService
{
    Task SyncAllAsync(CancellationToken ct);
}
```

Casos de uso (exemplos)

`Application/UseCases/GetColaboradoresQuery.cs`

```
using Company.CollabScripts.Application.Abstractions;
using Company.CollabScripts.Domain.Entities;

namespace Company.CollabScripts.Application.UseCases;

public sealed class GetColaboradoresQuery
{
    private readonly IColaboradorReadRepository _repo;
    public GetColaboradoresQuery(IColaboradorReadRepository repo) => _repo =
repo;
    public Task<IReadOnlyList<Colaborador>> Handle(CancellationToken ct) =>
_repo.ListAsync(ct);
}
```

Application/UseCases/GetColaboradorByCpfQuery.cs

```
using Company.CollabScripts.Application.Abstractions;
using Company.CollabScripts.Domain.Entities;

namespace Company.CollabScripts.Application.UseCases;

public sealed class GetColaboradorByCpfQuery
{
    private readonly IColaboradorReadRepository _repo;
    public GetColaboradorByCpfQuery(IColaboradorReadRepository repo) =>
_repo = repo;
    public Task<Colaborador?> Handle(string cpf, CancellationToken ct) =>
_repo.GetByCpfAsync(cpf, ct);
}
```

Application/UseCases/GetScriptsByCargoQuery.cs

```
using Company.CollabScripts.Application.Abstractions;
using Company.CollabScripts.Domain.Entities;

namespace Company.CollabScripts.Application.UseCases;

public sealed class GetScriptsByCargoQuery
{
    private readonly IScriptReadRepository _repo;
    public GetScriptsByCargoQuery(IScriptReadRepository repo) => _repo =
repo;
    public Task<ScriptPerguntas?> Handle(string cargo, CancellationToken ct)
=> _repo.GetByCargoAsync(cargo, ct);
}
```

Application/UseCases/GetScriptsByCpfQuery.cs

```
using Company.CollabScripts.Application.Abstractions;

namespace Company.CollabScripts.Application.UseCases;

public sealed class GetScriptsByCpfQuery
{
    private readonly GetColaboradorByCpfQuery _getColab;
    private readonly GetScriptsByCargoQuery _getScripts;
    public GetScriptsByCpfQuery(GetColaboradorByCpfQuery getColab,
GetScriptsByCargoQuery getScripts)
    { _getColab = getColab; _getScripts = getScripts; }

    public async Task<(string Cargo, IReadOnlyList<string> Perguntas)?>
```

```
Handle(string cpf, CancellationToken ct)
    {
        var colab = await _getColab.Handle(cpf, ct);
        if (colab is null) return null;
        var scripts = await _getScripts.Handle(colab.Cargo, ct);
        return scripts is null ? null : (scripts.Cargo, scripts.Perguntas);
    }
}
```

Infrastructure

Refit Clients + Polly Policies

`Infrastructure/External/ColaboradoresApiClient.cs`

```
using Refit;

namespace Company.CollabScripts.Infrastructure.External;

public interface IColaboradoresApi
{
    [Get("/colaboradores")] Task<List<ColaboradorDto>>
GetColaboradoresAsync();
}

public sealed record ColaboradorDto(string nome, string cpf, string cargo,
string email, string status);
```

`Infrastructure/External/ScriptsApiClient.cs`

```
using Refit;

namespace Company.CollabScripts.Infrastructure.External;

public interface IScriptsApi
{
    // Retorna perguntas por cargo
    [Get("/scripts/{cargo}")] Task<ScriptDto> GetScriptByCargoAsync(string
cargo);
}

public sealed record ScriptDto(string cargo, List<string> perguntas);
```

`Infrastructure/DependencyInjection/RefitPollySetup.cs`

```
using Microsoft.Extensions.DependencyInjection;
using Polly;
using Polly.Contrib.WaitAndRetry;
using Polly.Extensions.Http;
using System.Net;
using Company.CollabScripts.Infrastructure.External;

namespace Company.CollabScripts.Infrastructure.DependencyInjection;

public static class RefitPollySetup
{
    public static IServiceCollection AddExternalApis(this IServiceCollection
services, IConfiguration cfg)
    {
        var collabBase = cfg["ExternalApis:Colaboradores"] ?? throw new
InvalidOperationException("Missing Colaboradores base URL");
        var scriptsBase = cfg["ExternalApis:Scripts"] ?? throw new
InvalidOperationException("Missing Scripts base URL");

        var jitter =
Backoff.DecorrelatedJitterBackoffV2(TimeSpan.FromMilliseconds(200),
retryCount: 5);
        var retry = HttpPolicyExtensions
            .HandleTransientHttpError()
            .OrResult(msg => msg.StatusCode ==
HttpStatusCode.TooManyRequests)
            .WaitAndRetryAsync(jitter);
        var breaker = HttpPolicyExtensions
            .HandleTransientHttpError()
            .CircuitBreakerAsync(handledEventsAllowedBeforeBreaking: 5,
durationOfBreak: TimeSpan.FromSeconds(30));
        var timeout = Policy.TimeoutAsync<HttpResponseMessage>(10);

        services
            .AddRefitClient<IColaboradoresApi>()
            .ConfigureHttpClient(c => c.BaseAddress = new Uri(collabBase))
            .AddPolicyHandler(retry)
            .AddPolicyHandler(breaker)
            .AddPolicyHandler(timeout);

        services
            .AddRefitClient<IScriptsApi>()
            .ConfigureHttpClient(c => c.BaseAddress = new Uri(scriptsBase))
            .AddPolicyHandler(retry)
            .AddPolicyHandler(breaker)
            .AddPolicyHandler(timeout);

        return services;
    }
}
```

Cache + Repositórios (Read)

`Infrastructure/Repositories/InMemoryColaboradorRepository.cs`

```
using Company.CollabScripts.Application.Abstractions;
using Company.CollabScripts.Domain.Entities;
using Company.CollabScripts.Infrastructure.External;
using Microsoft.Extensions.Caching.Memory;

namespace Company.CollabScripts.Infrastructure.Repositories;

public sealed class InMemoryColaboradorRepository :
IColaboradorReadRepository
{
    private readonly IMemoryCache _cache;
    private readonly IColaboradoresApi _api;
    private readonly MemoryCacheEntryOptions _opts;
    private const string CacheKey = "collab:list";

    public InMemoryColaboradorRepository(IMemoryCache cache,
IColaboradoresApi api, IOptions<CacheSettings> settings)
    {
        _cache = cache; _api = api;
        _opts = new MemoryCacheEntryOptions {
AbsoluteExpirationRelativeToNow =
TimeSpan.FromMinutes(settings.Value.TtlMinutes) };
    }

    public async Task<IReadOnlyList<Colaborador>>
ListAsync(CancellationToken ct)
    {
        if (_cache.TryGetValue(CacheKey, out List<Colaborador> cached))
            return cached;

        var dtos = await _api.GetColaboradoresAsync();
        var mapped = dtos.Select(d => new Colaborador(d.nome, d.cpf,
d.cargo, d.email,
            d.status.Equals("ativo", StringComparison.OrdinalIgnoreCase) ?
StatusColaborador.Ativo : StatusColaborador.Inativo)).ToList();
        _cache.Set(CacheKey, mapped, _opts);
        return mapped;
    }

    public async Task<Colaborador?> GetByCpfAsync(string cpf,
CancellationToken ct)
    {
        var list = await ListAsync(ct);
        return list.FirstOrDefault(x => string.Equals(x.Cpf, cpf,
StringComparison.OrdinalIgnoreCase));
```

```
    }
}
```

`Infrastructure/Repositories/InMemoryScriptRepository.cs`

```
using Company.CollabScripts.Application.Abstractions;
using Company.CollabScripts.Domain.Entities;
using Company.CollabScripts.Infrastructure.External;
using Microsoft.Extensions.Caching.Memory;

namespace Company.CollabScripts.Infrastructure.Repositories;

public sealed class InMemoryScriptRepository : IScriptReadRepository
{
    private readonly IMemoryCache _cache;
    private readonly IScriptsApi _api;
    private readonly MemoryCacheEntryOptions _opts;

    public InMemoryScriptRepository(IMemoryCache cache, IScriptsApi api,
IOptions<CacheSettings> settings)
    {
        _cache = cache; _api = api;
        _opts = new MemoryCacheEntryOptions {
AbsoluteExpirationRelativeToNow =
TimeSpan.FromMinutes(settings.Value.TtlMinutes) };
    }

    public async Task<ScriptPerguntas?> GetByCargoAsync(string cargo,
CancellationToken ct)
    {
        var key = $"scripts:{cargo.ToLowerInvariant()}";
        if (_cache.TryGetValue(key, out ScriptPerguntas found))
            return found;

        var dto = await _api.GetScriptByCargoAsync(cargo);
        var mapped = new ScriptPerguntas(dto.cargo, dto.perguntas);
        _cache.Set(key, mapped, _opts);
        return mapped;
    }
}
```

Sincronização Periódica

`Infrastructure/Sync/ExternalSyncService.cs`

```
using Company.CollabScripts.Application.Abstractions;
using Microsoft.Extensions.Caching.Memory;
```

```
namespace Company.CollabScripts.Infrastructure.Sync;

public sealed class ExternalSyncService : IExternalSyncService
{
    private readonly IColaboradorReadRepository _colabRepo;
    private readonly IScriptReadRepository _scriptRepo;

    public ExternalSyncService(IColaboradorReadRepository colabRepo,
IScriptReadRepository scriptRepo)
    { _colabRepo = colabRepo; _scriptRepo = scriptRepo; }

    public async Task SyncAllAsync(CancellationToken ct)
    {
        // Precarrega a lista de colaboradores
        var list = await _colabRepo.ListAsync(ct);
        // Precarrega scripts por cargo distinto
        var cargos = list.Select(x =>
x.Cargo).Distinct(StringComparer.OrdinalIgnoreCase);
        foreach (var cargo in cargos)
        {
            _ = await _scriptRepo.GetByCargoAsync(cargo, ct);
        }
    }
}
```

`Infrastructure/Sync/CacheWarmupHostedService.cs`

```
using Company.CollabScripts.Application.Abstractions;
using Microsoft.Extensions.Hosting;

namespace Company.CollabScripts.Infrastructure.Sync;

public sealed class CacheWarmupHostedService : BackgroundService
{
    private readonly IExternalSyncService _sync;
    private readonly ILogger<CacheWarmupHostedService> _logger;
    private readonly TimeSpan _interval;

    public CacheWarmupHostedService(IExternalSyncService sync,
IOptions<CacheSettings> settings, ILogger<CacheWarmupHostedService> logger)
    { _sync = sync; _logger = logger; _interval =
TimeSpan.FromMinutes(settings.Value.RefreshMinutes); }

    protected override async Task ExecuteAsync(CancellationToken
stoppingToken)
    {
        while (!stoppingToken.IsCancellationRequested)
        {
            try { await _sync.SyncAllAsync(stoppingToken); }
```

```
            catch (Exception ex) { _logger.LogError(ex, "Erro ao sincronizar
caches"); }
            await Task.Delay(_interval, stoppingToken);
        }
    }
}
```

Settings + DI

`Infrastructure/Settings/CacheSettings.cs`

```
namespace Company.CollabScripts.Infrastructure;

public sealed class CacheSettings
{
    public int TtlMinutes { get; set; } = 30;       // TTL por item
    public int RefreshMinutes { get; set; } = 15;   // job de refresh
}
```

`Infrastructure/DependencyInjection/InfraModule.cs`

```
using Company.CollabScripts.Application.Abstractions;
using Company.CollabScripts.Infrastructure.Repositories;
using Company.CollabScripts.Infrastructure.Sync;
using Microsoft.Extensions.DependencyInjection;

namespace Company.CollabScripts.Infrastructure.DependencyInjection;

public static class InfraModule
{
    public static IServiceCollection AddInfrastructure(this
IServiceCollection services, IConfiguration cfg)
    {
        services.AddMemoryCache();
        services.Configure<CacheSettings>(cfg.GetSection("Cache"));

        services.AddExternalApis(cfg);

        services.AddScoped<IColaboradorReadRepository,
InMemoryColaboradorRepository>();
        services.AddScoped<IScriptReadRepository,
InMemoryScriptRepository>();

        services.AddScoped<IExternalSyncService, ExternalSyncService>();
        services.AddHostedService<CacheWarmupHostedService>();

        return services;
```

```
    }
}
```

API (Apresentação)

`Api/Program.cs`

```
using Company.CollabScripts.Application.Abstractions;
using Company.CollabScripts.Application.UseCases;
using Company.CollabScripts.Infrastructure.DependencyInjection;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

builder.Services.AddInfrastructure(builder.Configuration);

// Application services
builder.Services.AddScoped<GetColaboradoresQuery>();
builder.Services.AddScoped<GetColaboradorByCpfQuery>();
builder.Services.AddScoped<GetScriptsByCargoQuery>();
builder.Services.AddScoped<GetScriptsByCpfQuery>();

var app = builder.Build();

if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.MapGet("/health", () => Results.Ok(new { status = "ok" }));

app.MapGet("/colaboradores", async (GetColaboradoresQuery q,
CancellationToken ct) =>
{
    var result = await q.Handle(ct);
    return Results.Ok(result.Select(x => new { nome = x.Nome, cpf = x.Cpf,
cargo = x.Cargo, email = x.Email, status = x.Status.ToString() }));
});

app.MapGet("/colaboradores/{cpf}", async (string cpf,
GetColaboradorByCpfQuery q, CancellationToken ct) =>
{
    var r = await q.Handle(cpf, ct);
    return r is null ? Results.NotFound() : Results.Ok(new { r.Nome, r.Cpf,
r.Cargo, r.Email, status = r.Status.ToString() });
```

```
});

app.MapGet("/scripts/{cargo}", async (string cargo, GetScriptsByCargoQuery
q, CancellationToken ct) =>
{
    var r = await q.Handle(cargo, ct);
    return r is null ? Results.NotFound() : Results.Ok(new { cargo =
r.Cargo, perguntas = r.Perguntas });
});

app.MapGet("/colaboradores/{cpf}/scripts", async (string cpf,
GetScriptsByCpfQuery q, CancellationToken ct) =>
{
    var r = await q.Handle(cpf, ct);
    return r is null ? Results.NotFound() : Results.Ok(new { cargo =
r.Value.Cargo, perguntas = r.Value.Perguntas });
});

app.Run();
```

`Api/appsettings.json` (exemplo)

```
{
  "ExternalApis": {
    "Colaboradores": "https://api.exemplo.com",
    "Scripts": "https://api.exemplo.com"
  },
  "Cache": {
    "TtlMinutes": 30,
    "RefreshMinutes": 15
  },
  "Logging": {
    "LogLevel": { "Default": "Information", "Microsoft.AspNetCore":
"Warning" }
  },
  "AllowedHosts": "*"
}
```

Testes

Unit Tests (xUnit)

`UnitTests/GetColaboradoresQueryTests.cs`

```
using Company.CollabScripts.Application.Abstractions;
using Company.CollabScripts.Application.UseCases;
```

```
using Company.CollabScripts.Domain.Entities;
using Moq;

public class GetColaboradoresQueryTests
{
    [Fact]
    public async Task DeveRetornarLista()
    {
        var repo = new Mock<IColaboradorReadRepository>();
        repo.Setup(x => x.ListAsync(It.IsAny<CancellationToken>()))
            .ReturnsAsync(new List<Colaborador> {
new("Ana","000","Dev","ana@x.com", StatusColaborador.Ativo) });
        var q = new GetColaboradoresQuery(repo.Object);
        var result = await q.Handle(CancellationToken.None);
        Assert.Single(result);
    }
}
```

UnitTests/GetScriptsByCpfQueryTests.cs

```
using Company.CollabScripts.Application.UseCases;
using Company.CollabScripts.Domain.Entities;
using Moq;
using Company.CollabScripts.Application.Abstractions;

public class GetScriptsByCpfQueryTests
{
    [Fact]
    public async Task DeveRetornarPerguntasPeloCpf()
    {
        var repoColab = new Mock<IColaboradorReadRepository>();
        repoColab.Setup(x => x.GetByCpfAsync("123",
It.IsAny<CancellationToken>()))
            .ReturnsAsync(new Colaborador("Bob","123","Dev","bob@x.com",
StatusColaborador.Ativo));

        var repoScript = new Mock<IScriptReadRepository>();
        repoScript.Setup(x => x.GetByCargoAsync("Dev",
It.IsAny<CancellationToken>()))
            .ReturnsAsync(new ScriptPerguntas("Dev", new []{"Pergunta 1"}));

        var getColab = new GetColaboradorByCpfQuery(repoColab.Object);
        var getScript = new GetScriptsByCargoQuery(repoScript.Object);
        var sut = new GetScriptsByCpfQuery(getColab, getScript);

        var res = await sut.Handle("123", CancellationToken.None);
        Assert.NotNull(res);
        Assert.Equal("Dev", res?.Cargo);
        Assert.Single(res?.Perguntas!);
```

```
    }
}
```

Integration Tests (WebApplicationFactory)

`IntegrationTests/ApiTests.cs`

```
using System.Net;
using Microsoft.AspNetCore.Mvc.Testing;
using Microsoft.Extensions.DependencyInjection;
using Refit;
using Company.CollabScripts.Infrastructure.External;

public class ApiTests : IClassFixture<WebApplicationFactory<Program>>
{
    private readonly WebApplicationFactory<Program> _factory;
    public ApiTests(WebApplicationFactory<Program> factory)
    {
        _factory = factory.WithWebHostBuilder(builder =>
        {
            builder.ConfigureServices(services =>
            {
                // Substitui Refit por stubs em memória
                services.AddSingleton<IColaboradoresApi>(new StubColabApi());
                services.AddSingleton<IScriptsApi>(new StubScriptsApi());
            });
        });
    }

    [Fact]
    public async Task GetColaboradores_DeveRetornarOk()
    {
        var client = _factory.CreateClient();
        var resp = await client.GetAsync("/colaboradores");
        Assert.Equal(HttpStatusCode.OK, resp.StatusCode);
    }
}

internal sealed class StubColabApi : IColaboradoresApi
{
    public Task<List<ColaboradorDto>> GetColaboradoresAsync() =>
        Task.FromResult(new List<ColaboradorDto>{
new("Ana","111","Dev","ana@x.com","ativo") });
}

internal sealed class StubScriptsApi : IScriptsApi
{
    public Task<ScriptDto> GetScriptByCargoAsync(string cargo) =>
```

```
        Task.FromResult(new ScriptDto(cargo, new(){"O que é SOLID?"}));
}
```

Observação: marque o projeto `Company.CollabScripts.Api` com a classe `Program` como `public partial class Program { }` caso use `WebApplicationFactory` no .NET 8 (para descoberta do entrypoint nos testes).

`Api/Program.cs` **(linha final para testes)**

```
public partial class Program { }
```

Observabilidade e Erros

- **Logs**: use `ILogger<T>` em repositórios/hosted services. Configure níveis por categoria em `appsettings`.
- **Polly**: retry exponencial com jitter + circuit breaker + timeout.
- **Métricas**: adicione `EventCounters`/Prometheus (ex.: `prometheus-net`) se desejar.
- **Erros de domínio**: retorne `404` quando não encontrar, `503` se APIs externas indisponíveis (padrão por exceção filtrada + ProblemDetails).

Filtro de exceptions para ProblemDetails (opcional)

```
app.Use(async (ctx, next) =>
{
    try { await next(); }
    catch (HttpRequestException ex)
    {
        ctx.Response.StatusCode = StatusCodes.Status503ServiceUnavailable;
        await ctx.Response.WriteAsJsonAsync(new { title = "Service
Unavailable", detail = ex.Message });
    }
});
```

Como Rodar

```
# na raiz
dotnet new sln -n CollabScripts
# criar projetos conforme estrutura e adicionar referências
# dotnet add src/Company.CollabScripts.Api package Swashbuckle.AspNetCore
# dotnet add src/Company.CollabScripts.Infrastructure package Refit
# dotnet add src/Company.CollabScripts.Infrastructure package Polly
Polly.Extensions.Http Polly.Contrib.WaitAndRetry
# dotnet add tests/* package Microsoft.AspNetCore.Mvc.Testing Moq xunit
```

```
# executar
cd src/Company.CollabScripts.Api
dotnet run
```

Extensões Futuras

- **Evitando thundering herd**: use `SemaphoreSlim` ao popular cache por chave.
- **Cache distribuído**: mudar para Redis (`IDistributedCache`) mantendo interfaces.
- **Autorização**: incluir JWT/KeyAuth no gateway.
- **Versionamento**: `/v1/...` nas rotas.
- **Paginação** em `/colaboradores`.