C# SOFTWARE DEVELOPMENT

FIAP **CP2** **16/10/2025**

Nome: _____ Nota: _____

A prova toda irá se basear em um sistema de uma loja.

A Sangue & Algodão é uma loja de camisas de bandas com vendas presenciais e online. O sistema opera com:

- Banco de dados para produtos e pedidos;
- Arquivos diários para integração e auditoria de estoque e compras;
- Web API interna para consultas e acionamento de fluxos.

O ciclo diário consiste em: abrir o estoque, registrar compras ao longo do dia e fechar o estoque com reconciliação e atualização no banco.

Além do Id, a tupla (IdBanda, Tamanho, Modelo) identifica unicamente uma camisa.

Entidades do banco:

Banda
+ Id
+ Nome
+ País
+ Gênero

Camisa
+ Id
+ IdBanda
+ Tamanho
+ Modelo
+ Cor
+ Tecido
+ Preço
+ QuantidadeEmEstoque
+ QuantidadeMinimaAlerta

Pedido
+ Id
+ Data
+ CpfCliente
+ Status

ItemPedido
+ Id
+ IdPedido
+ IdCamisa

1. Escreva uma classe para representar a entidade `Camisa`.

2. **Acesso ao banco com Entity Framework.**

   Assuma que já existem:

   - A classe `Camisa` da Questão 1.
   - Um `DbContext` chamado `LojaContext`, com `DbSet<Camisa> Camisas`.
   - A conexão com o banco já configurada.

   Escreva um código que:

   1. Busque **todas as camisas** do banco usando **Entity Framework**;
   2. Ordene por `Modelo` e selecione os campos `Id, Modelo, Tamanho, QuantidadeEmEstoque`;
   3. Imprima cada linha no console no formato: `Id - Modelo (Tamanho) : QuantidadeEmEstoque`.

```
1  using Microsoft.EntityFrameworkCore;
2  using var db = new LojaContext();
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
```

3. **Arquivos diários de estoque.**

   Convenção de nomes: use o padrão `<dia_de_hoje>` no formato ddMMyyyy. Ex.: hoje 16/10/2025, seria `16102025`.

   Descrição dos arquivos

   - `<dia_de_hoje>_estoque_inicial.txt`
     Formato (sem cabeçalho), linhas no padrão:
     `IdCamisa;NomeBanda;QuantidadeInicial`
     Ex.: `1;Iron Maiden;30`
   - `<dia_de_hoje>_compras.txt`
     Temos 3 tipos de linha, uma linha com apenas #, onde indica que o último pedido chegou ao fim e começamos outro,
     na linha logo após, temos o formato
     `IdPedido;CpfCliente;hh:mm`
     Ex.: `1;01234567890;10:49`
     e depois temos 1 ou mais linhas descrevendo os itens do pedido,
     `Banda;Tamanho;Modelo;Preço`
     Ex.: `Led Zeppelin;M;tour1970;75.00`
   - `<dia_de_hoje>_estoque_final.txt`
     Padrão semelhante a `<dia_de_hoje>_estoque_inicial.txt`
     `IdCamisa;NomeBanda;QuantidadeFinal`
     Ex.: `1;Iron Maiden;18`

   (a) **(Geração do estoque inicial)** Monte o conteúdo do arquivo `<dia_de_hoje>_estoque_inicial.txt` e salve-o no diretório atual. Você pode assumir que temos todas as entradas do banco de cada tabela em listas do seu modelo.

```
using System;
using System.Globalization;
using DatabaseService;
using Models;

List<Banda> bandas   = DatabaseService.GetAllBandas();
```

```
List<Camisa> camisas = DatabaseService.GetAllCamisas();
```

(b) **(Cálculo do estoque final)** Considerando que existem no diretório atual os arquivos:
`<dia_de_hoje>_estoque_inicial.txt`
e `<dia_de_hoje>_compras.txt`, leia eles usando a static class File e crie o arquivo `<dia_de_hoje>_estoque_final.txt`
Além disso, imprima no console o total de receita obtida no dia.

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
```

(c) **(Persistência no banco)** Atualize a tabela `Camisas` no banco de dados com a quantidade final calculada no dia, ou baseada nas compras feitas.

```
using Microsoft.EntityFrameworkCore;
using System.Globalization;

var hoje = DateTime.Today.ToString(
            "ddMMyyyy", CultureInfo.InvariantCulture);
```

(d) Baseado no arquivo `<dia_de_hoje>_compras.txt`, adicione as entradas na tabela Pedidos e ItemPedidos.

4. **Web API – Itens com baixo estoque.**

   (a) Implemente um endpoint de uma Web API que retorne as camisas em **baixo estoque**, isto é, aquelas em que `QuantidadeMinimaAlerta` $\geq$ `QuantidadeEmEstoque`.

   A resposta deve ser ActionResult<EstoqueCriticoDTO> onde "quantidade" é a quantidade de camisas com estoque crítico, e "camisas" seja a lista das camisas com estoque crítico.

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);
builder.Services.AddDbContext<LojaContext>();
builder.Services.AddControllers();
var app = builder.Build();
app.MapControllers();
app.Run();

public class EstoqueCriticoDTO
{
    public int Quantidade {get; set;}
    public List<Camisa> Camisas {get; set;}
}

[ApiController]
[Route("api/[controller]")]
public class CamisasController : ControllerBase
{
    private readonly LojaContext _db;
    public CamisasController(LojaContext db) => _db = db;

    [HttpGet("estoque-critico")]
    public async Task<ActionResult<EstoqueCriticoDTO>>
```

```
    Get(CancellationToken ct)
26  {
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53  }
54 }
```

(b) Para fazer uma requisição ao endpoint desenvolvido em a), qual deve ser a url? Pode só dizer o Path.

System.IO

```
File.ReadAllLines(path)
File.ReadLines(path)
File.WriteAllLines(path, lines)
File.AppendAllText(path, text)
File.Exists(path)
Directory.CreateDirectory(path)
Path.Combine(a, b, ...)
StreamReader(path)
StreamWriter(path, append:false)
File.Replace(temp, final, null)
File.Move(temp, final, overwrite:true)
```

LINQ

```
Where(...)
Select(...)
SelectMany(...)
OrderBy(...), ThenByDescending(...)
GroupBy(...)
Join(...), GroupJoin(...)
DefaultIfEmpty()
Distinct()
ToList(), ToDictionary(...)
Sum(...), Count(...)
Any(...), All(...)
First(), FirstOrDefault()
```

Parsing

```
line.Split(
';', StringSplitOptions.TrimEntries)
string.IsNullOrWhiteSpace(s)
int.TryParse(s, out _)
decimal.TryParse
(s, CultureInfo.InvariantCulture, out _)
DateTimeOffset.Parse(s)
```

Entity Framework

```
var set = context.Set<TEntity>();

// leitura (lista)
await set.AsNoTracking()
  .Where(e => /* cond */)
  .Select(e => new { /* proj */ })
  .OrderBy(e => /* key */)
  .ToListAsync(ct);

// leitura (por chave)
var ent = await set.FindAsync(key, ct);

// inclusão simples
await set.AddAsync(entity, ct);
await context.SaveChangesAsync(ct);

// inclusão em lote
await set.AddRangeAsync(list, ct);
await context.SaveChangesAsync(ct);

// atualização simples
set.Update(entity);
await context.SaveChangesAsync(ct);

// transação
await using var tx =
await context.Database.BeginTransactionAsync(ct);
/* ... várias operações ... */
await context.SaveChangesAsync(ct);
await tx.CommitAsync(ct);

// include
await context.Set<TRoot>()
  .Include(r => r.Nav)
  .ThenInclude(n => n.Sub)
  .ToListAsync(ct);
```

Console / Data

```
Console.WriteLine(...)
Console.Error.WriteLine(...)
string.Format(...)
$"{...}"
DateTime.Today.ToString(
"ddMMyyyy", CultureInfo.InvariantCulture)
```

ASP.NET Core Controllers

```
[ApiController]
[Route("api/[controller]")]
public class XController : ControllerBase { /* ... */ }

[HttpGet], [HttpPost], [HttpPut], [HttpDelete]
ActionResult<T>
[FromRoute], [FromQuery], [FromBody]
CancellationToken

Ok(obj), NotFound(), BadRequest()
Created(uri, obj), NoContent(), Problem(...)
```

RASCUNHO

Boa prova!

"A simplicidade é pré-requisito para a confiabilidade."
— Edsger W. Dijkstra