Fase 7 — Repository JSON — persistência em JSON

Título para o ClassHero: Fase 7 — Repository JSON (System.Text.Json)

Descrição (para o ClassHero): Evoluam o Repository para **persistir em JSON** (ex.: `books.json`) usando **System.Text.Json**. Implementem as mesmas operações (**Add/Get/List/Update/Remove**), mantendo o **cliente dependente apenas do Repository** (não acessar arquivo direto). Tratem **arquivo ausente/vazio**, definam opções de serialização (**camelCase**, ignorar `null`) e documentem decisões no README. Incluam **testes de integração** com arquivo temporário.

Entregáveis (repo da equipe)

- Pasta: `src/fase-06-repository-json/`.
- Arquivo `.md` com diagrama leve e **snippets C#** do Repository JSON + uso por um serviço cliente.
- **Testes** com temp file/diretório, cobrindo operações e arquivo inexistente/vazio.
- **README (raiz)** atualizado: **composição da equipe (nomes e RAs)**, sumário com link e como executar os testes.

Snippets C# (modelo mínimo)

```
using System.Text.Json;
using System.Text.Json.Serialization;

public sealed class JsonBookRepository : IRepository<Book, int>
{
    private readonly string _path;
    private static readonly JsonSerializerOptions _opts = new()
    {
        PropertyNamingPolicy = JsonNamingPolicy.CamelCase,
        DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingNull,
        WriteIndented = true
    };

    public JsonBookRepository(string path) => _path = path;

    public Book Add(Book e) { var list = Load(); list.RemoveAll(b => b.Id ==
e.Id); list.Add(e); Save(list); return e; }
    public Book? GetById(int id) => Load().FirstOrDefault(b => b.Id == id);
    public IReadOnlyList<Book> ListAll() => Load();
    public bool Update(Book e) { var list = Load(); var i = list.FindIndex(b
=> b.Id == e.Id); if (i < 0) return false; list[i] = e; Save(list); return
true; }
```

```
    public bool Remove(int id) { var list = Load(); var ok =
list.RemoveAll(b => b.Id == id) > 0; if (ok) Save(list); return ok; }

    private List<Book> Load()
    {
        if (!File.Exists(_path)) return new();
        var json = File.ReadAllText(_path);
        if (string.IsNullOrWhiteSpace(json)) return new();
        return JsonSerializer.Deserialize<List<Book>>(json, _opts) ?? new();
    }

    private void Save(List<Book> list)
    {
        var json = JsonSerializer.Serialize(list, _opts);
        File.WriteAllText(_path, json);
    }
}
```

> **Observação didática:** mantenha **regra de negócio** fora do Repository; ele deve cuidar **apenas do acesso a dados**.

Testes (ideias rápidas)

- `ListAll()` com arquivo inexistente → **lista vazia**.
- `Add()` cria arquivo e persiste; `GetById()` encontra.
- `Update()` retorna `false` se id não existe; `true` quando atualiza.
- `Remove()` exclui item e persiste lista.
- Conteúdo vazio/whitespace no arquivo → comportar-se como lista vazia.

Dica: use `Path.GetTempFileName()` ou um diretório temporário por teste e **limpe ao final**.

Critérios (0–10)

- Persistência JSON correta (System.Text.Json + opções) — **0–3**
- Operações Add/Get/List/Update/Remove — **0–3**
- Testes de integração com temp file — **0–2**
- README e documentação de formato/decisões — **0–2**

Tempo sugerido

25–30 minutos

Pitfalls

- Esquecer `JsonSerializerOptions` (gerando PascalCase sem querer) — padronize **camelCase**.
- Não tratar arquivo **ausente/vazio** explicitamente.
- Misturar **regra de negócio** no Repository.

Peso sugerido

12/100 — consolida persistência estruturada, preparando refatorações (mappers, validações e políticas).