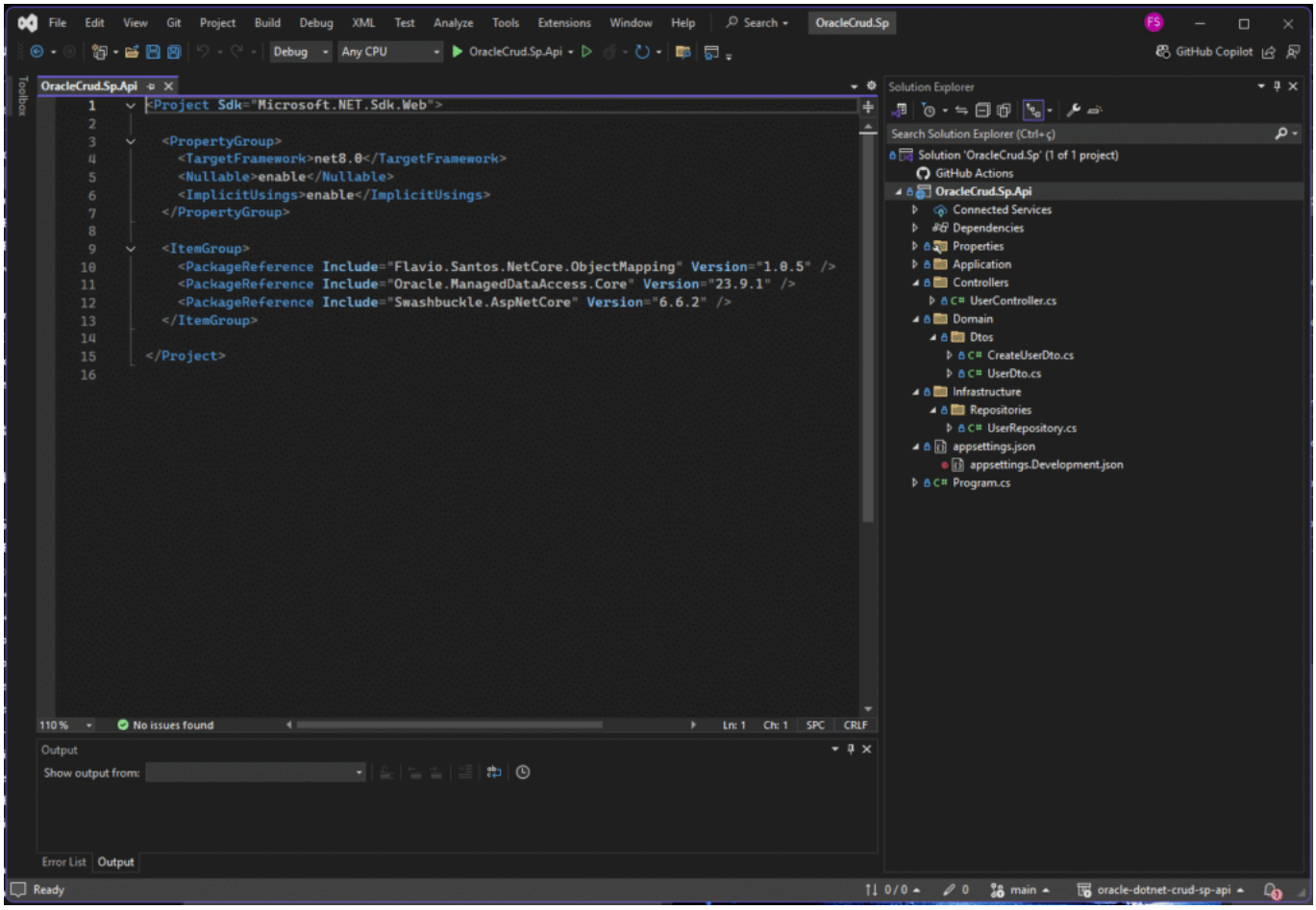
O que construímos?

Nessa série, você aprendeu a instalar o Oracle Database Express Edition (XE) no WSL2, executando-o em um contêiner Docker no Windows 11, sem depender do Docker Desktop. Também integrou o banco à sua API REST em .NET, utilizando stored procedures para as operações de dados.

O que é uma Solution no Visual Studio?

Uma Solution é um arquivo de orquestração (.sln) que diz ao Visual Studio quais projetos fazem parte do mesmo trabalho e onde eles estão. Pense nela como um índice/manifesto. Ela não é o código e não guarda configurações de execução da aplicação; apenas agrupa projetos, permite abrir tudo junto, definir qual projeto inicia ao depurar (Startup Project) e organizar pastas virtuais (solution folders).

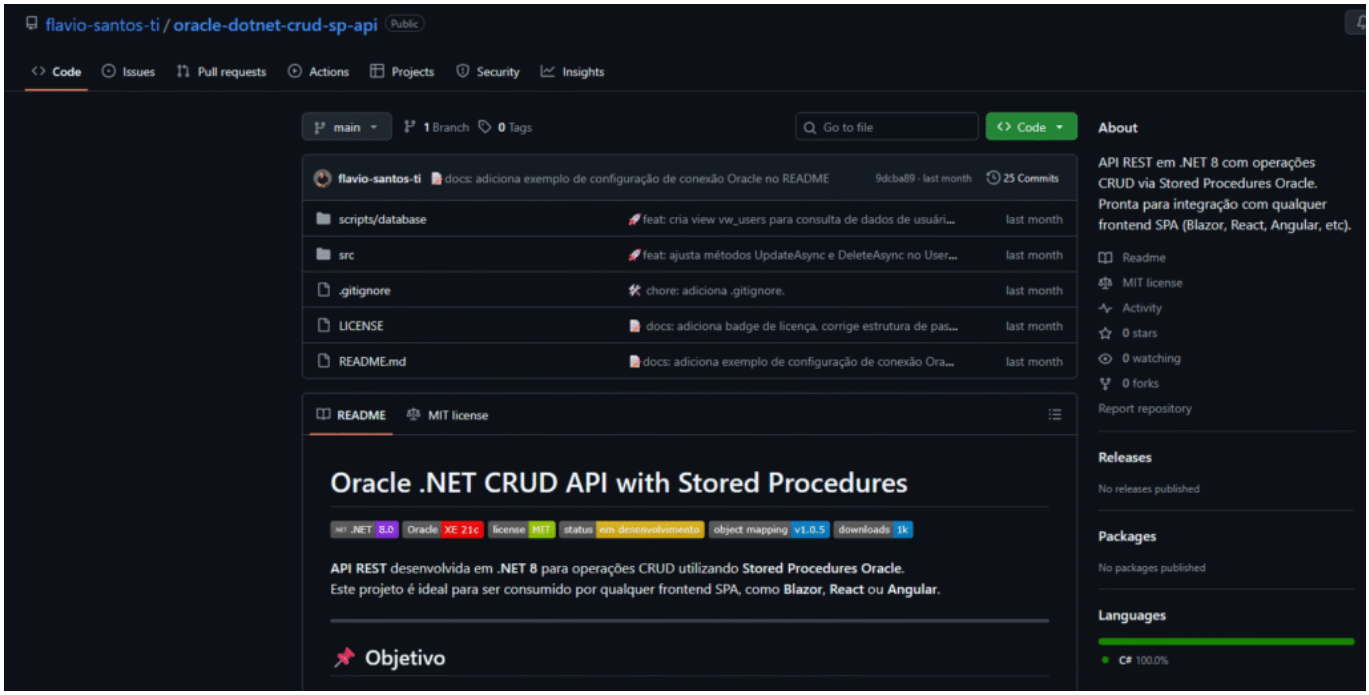
Neste tutorial, decidimos nomear a solução como OracleCrud.Sp.sln. Ela contém apenas um projeto, `OracleCrud.Sp.Api`. Quando mencionamos “camadas”, estamos falando somente da organização por pastas dentro desse único projeto (Application, Domain, Infrastructure). Não existem projetos ou assemblies separados. Tudo é compilado junto. Essa organização por pastas existe para manter o código claro, organizado e fácil de navegar.



Código completo do projeto para baixar e estudar

O que é

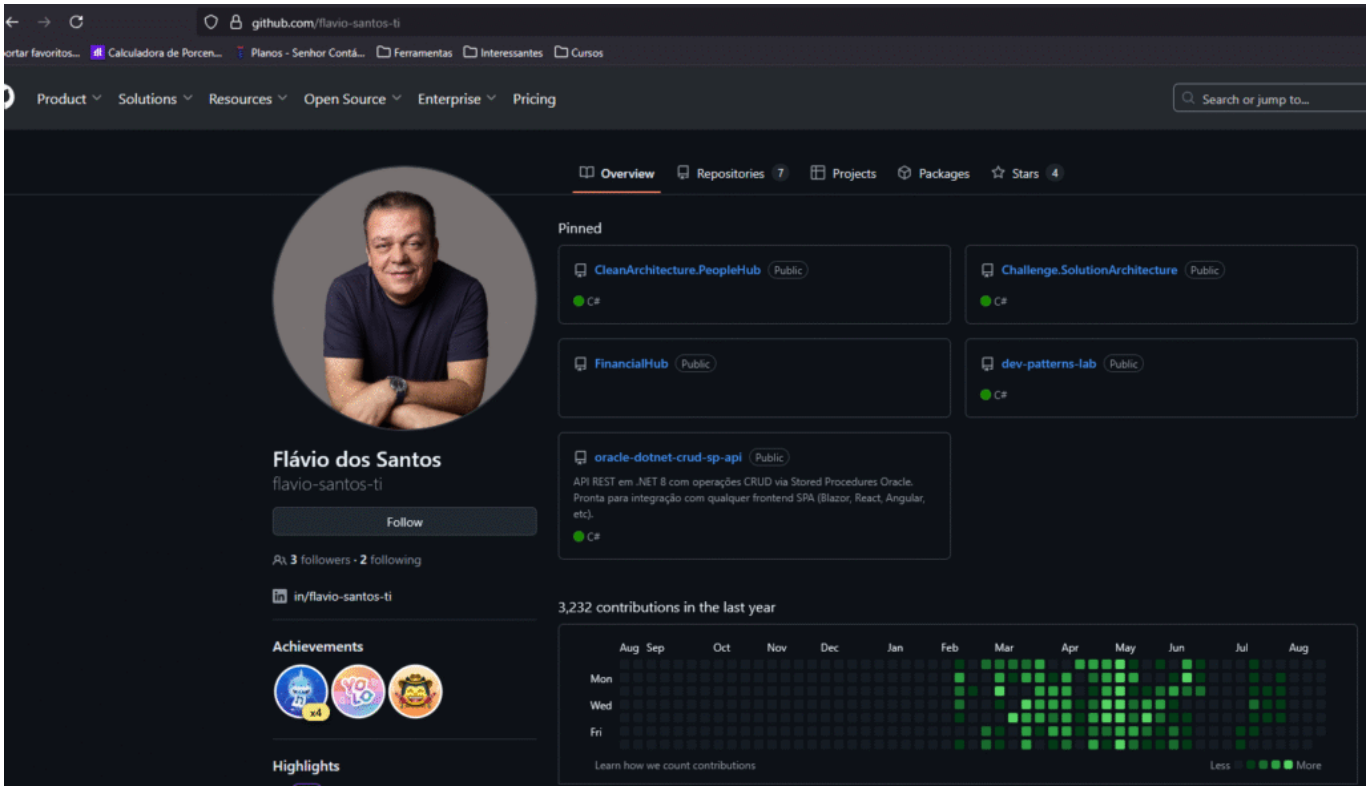
Este é o repositório público com o código completo do projeto usado no tutorial. Ele serve para você acompanhar os passos, comparar com o seu código e praticar.



Onde está

O código fica hospedado no GitHub e pode ser aberto direto no navegador, sem instalar nada.
Você também pode ir ao perfil flavio-santos-ti e procurar pelo repositório `oracle-dotnet-crud-sp-api`.
As pastas mais úteis para começar são `src` (contém a solution do Visual Studio) e `scripts/database` (scripts SQL).

Acesse por último, quando estiver pronto para abrir:
https://github.com/flavio-santos-ti/oracle-dotnet-crud-sp-api



O que você encontra lá

Ao abrir o repositório, comece pela pasta `src`. Dentro dela está o arquivo `OracleCrud.Sp.sln`, que é a solução a ser aberta no Visual Studio para carregar o projeto. A API foi organizada em pastas apenas para facilitar a leitura do código: `Application` reúne os controladores e endpoints, `Domain` guarda os modelos e DTOs, e `Infrastructure` concentra os repositórios que acessam o banco. Na pasta `scripts/database` você encontra os arquivos SQL usados no tutorial, incluindo criação de esquema, tabelas, views e stored procedures. Quando precisar de um passo a passo rápido para executar o projeto, consulte o arquivo `README.md` na raiz do repositório.

Como baixar sem usar Git

1. Abra o repositório no navegador;
2. Clique em Code e depois em Download ZIP;
3. Extraia o ZIP;
4. No Visual Studio 2022, abra `src/OracleCrud.Sp.sln` e execute a API.

Como clonar com Git (opcional)

```
git clone https://github.com/flavio-santos-ti/oracle-dotnet-crud-sp-api.git
```

Depois de clonar, abra `src/OracleCrud.Sp.sln` no Visual Studio e execute a API.

> **Dica:** Leia na ordem `scripts/database` → `Infrastructure` → `Domain` → `Application`. Use o Swagger para testar cada endpoint e observe como as stored procedures são chamadas no repositório. Sempre que surgir dúvida, compare o seu código com o do repositório.

Ponto de partida para evoluir padrões na sua equipe

Este tutorial prioriza didática e simplicidade. Em alguns pontos usamos decisões que podem ser consideradas anti patterns em produção. A intenção é entregar um exemplo funcional e fácil de entender que sirva como base. A partir dele você pode adaptar e evoluir até o padrão mais aderente ao seu time, à sua empresa ou ao seu projeto pessoal.

Algumas simplificações presentes incluem trabalhar com uma única solução e um único projeto organizado por pastas, chamar stored procedures diretamente no repositório e manter validação e tratamento de erros no nível essencial. Isso facilita o aprendizado inicial e reduz distrações técnicas.

Caminhos de evolução sugeridos

- Separar por projetos quando fizer sentido ou por limites de contexto;
- Introduzir uma camada de serviços e, se couber, padrões como CQRS ou Mediator;
- Centralizar erros com um middleware e padronizar respostas da API;
- Fortalecer validação e mapeamento de modelos e DTOs.;
- Investir em observabilidade com logs estruturados e métricas, além de testes de unidade e integração.

Use este projeto como base de estudo e como ponto de partida. Evolua cada parte conforme as práticas e padrões do seu ambiente.

Mapeamento de objetos com Flavio.Santos.NetCore.ObjectMapping(usado no projeto)

Neste projeto usamos a biblioteca Flavio.Santos.NetCore.ObjectMapping para mapear dados entre DTOs e entidades de forma automática. Ela reduz código repetitivo e torna a conversão entre objetos simples e previsível para quem está começando.

O que é esta biblioteca?

Uma biblioteca leve para copiar dados de um objeto para outro quando as propriedades têm o mesmo nome e tipos compatíveis. Não exige configuração nem DI. Oferece conversão automática entre `enum` e `string` e inclui métodos fluentes para listas e pequenas mutações.

Para que serve?

Evitar código repetido ao converter DTOs de entrada e saída e modelos internos da API. Você escreve menos código braçal e mantém o foco na regra de negócios. No projeto, ela é o recurso usado para esse mapeamento.

Como usar (exemplo)?

```
using FDS.NetCore.ObjectMapping.Extensions;

// 1) Objeto único
var readDto = entity.MapTo<ReadUserDto>();

// 2) Lista
var readList = entities.MapToList<User, ReadUserDto>();

// 3) Pequenas alterações encadeadas
var entityToSave = createDto
    .MapTo<User>()
    .Apply(u => u.Id = Guid.NewGuid());
```

Quais são as observações rápidas?

- Converte `enum ↔ string` automaticamente quando os nomes coincidem, sem se importar com

maiúsculas e minúsculas.
- Propriedades sem correspondências são ignoradas e o destino só recebe o que puder ser escrito.
- `MapTo<T>()` espera a origem não nula.

Conclusão da Série

Você saiu do zero e chegou a uma API REST em .NET 8 integrada ao Oracle XE, validando o CRUD completo com testes de endpoint. A jornada uniu preparação do banco no WSL2 + Docker, modelagem com stored procedures e uma API organizada por pastas, com serviços, repositórios e controllers funcionando ponta a ponta.

O que entregamos, de ponta a ponta

- **Infra & Banco:** você subiu o Oracle XE em contêiner (volume nomeado, portas `1521/5500`, imagem oficial 21c) e configurou acesso via **DBeaver** (host `localhost`, porta `1521`, serviço XEPDB1). Criou um usuário de aplicação (`app_user`), a tabela `users` e escreveu **stored procedures** para inserção e atualização (e demais operações do CRUD).
- **API & Arquitetura:** criou o projeto **Web API (.NET 8)** no Visual Studio, definiu **DTOs** (`CreateUserDto`, `UserDto`) e as **interfaces** (`IUserService`, `IUserRepository`) para manter baixo acoplamento, seguindo uma organização por pastas (`Domain`, `Application`, `Infrastructure`).
- **Camada de Aplicação:** implementou o **UserService** para orquestrar regras e delegar persistência ao repositório. Aqui entrou a **Flavio.Santos.NetCore.ObjectMapping** para mapear DTO ⇄ entidade sem código repetitivo (incluindo conversão automática entre `enum` e `string`).
- **Controllers & Testes:** expôs endpoints REST (GET/POST/PUT/DELETE), configurou DI e **validou o CRUD** com ferramentas como o Postman, confirmando a integração com as stored procedures.

> **Observação final:** embora tenhamos justificado o uso de algumas simplificações e *anti patterns* por se tratar de um trabalho base para iniciantes, a intenção é que você evolua essa base para uma arquitetura e práticas mais maduras, alinhadas ao seu time, à sua empresa ou ao seu projeto pessoal.

Observação final: embora tenhamos justificado o uso de algumas simplificações e *anti patterns* por se tratar de um trabalho base para iniciantes, a intenção é que você evolua essa base para uma arquitetura e práticas mais maduras, alinhadas ao seu time, à sua empresa ou ao seu projeto pessoal.

Obrigado por acompanhar até aqui. A intenção né só dar os primeiros passos. É um convite para olhar padrões e antipadrões com curiosidade e atenção ao contexto. Defender um certo ou errado universal costuma ser delicado, pois cada escolha nasce dos limites de custo, tempo, recursos e disponibilidade do seu cenário. Eu respeito ideias diferentes e as justificativas que as sustentam e valorizo toda contribuição que traga clareza e entendimento. A dúvida funciona como um sinal de alerta de que algo pede atenção pode precisar ser resolvido, mudado ou melhorado. Decidir bem também significa considerar o planejamento estratégico da sua empresa ou organização e buscar alinhamento com objetivos de negócio e de login prazo. Que este material ajude a enxergar alternativas, ponderar os trade offs com mais serenidade hoje e também quando o contexto mudar.