Justificativa para o uso dessa arquitetura

Embora as arquiteturas modernas privilegiem regras de negócio na aplicação e sigam princípios *DDD* e *Clean Architecture*, ainda existem cenários legítimos em que Stored Procedures são utilizadas para encapsular lógica diretamente no banco de dados. Isso pode acontecer por motivos como:

- Restrições técnicas de sistemas legados;
- Demandas específicas de performance;
- Políticas internas de segurança ou governança;
- Decisões estratégicas da equipe ou da empresa.

Nesta série de artigos sobre a construção de uma API REST com Stored Procedures, o objetivo não é incentivar o uso indiscriminado dessa abordagem, mas sim mostrar como aplicá-la de forma estruturada e organizada quando ela for necessária. Em muitos projetos reais, é preciso adaptar a solução à realidade do ambiente, respeitando restrições e práticas já existentes.

Criando um Novo Projeto no Visual Studio 2022

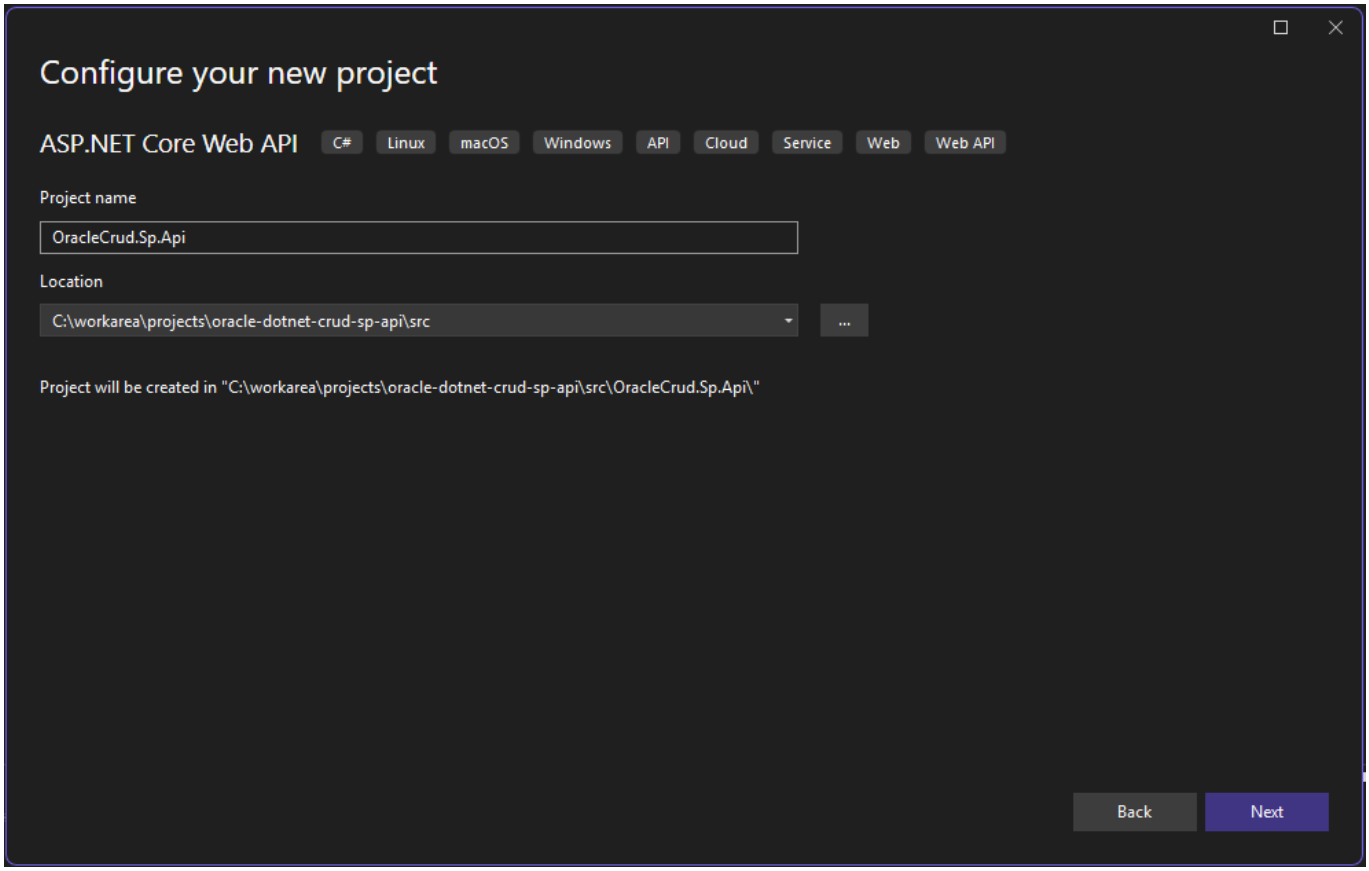
Para começarmos a construção da nossa API REST com .NET 8, o primeiro passo é criar um novo projeto no Visual Studio 2022. Essa será a IDE utilizada em toda a implementação. Siga os passos abaixo para chegar até a tela de seleção de templates:

1. Com o Visual Studio 2022 aberto, clique no menu `File > New - Project`;
2. Isso abrirá uma nova janela com o título “Create a new project” ou “Add a new project” (dependendo da sua versão/idioma).
3. No campo de busca, digite “API” e selecione o template “ASP.NET Core Web API”;
4. Verifique se a linguagem está definida como C#;
5. Clique em **Next** para prosseguir com a configuração do projeto.

Esta etapa define a base de nossa aplicação que será uma API REST desacoplada utilizando Stored Procedures com Oracle XE 21c.

Configurando o Nome do Projeto

pós selecionar o template ASP.NET Core Web API, o Visual Studio exibirá a tela “Configure your new project”. Essa etapa é responsável por definir o nome do seu projeto e o local onde ele será salvo.



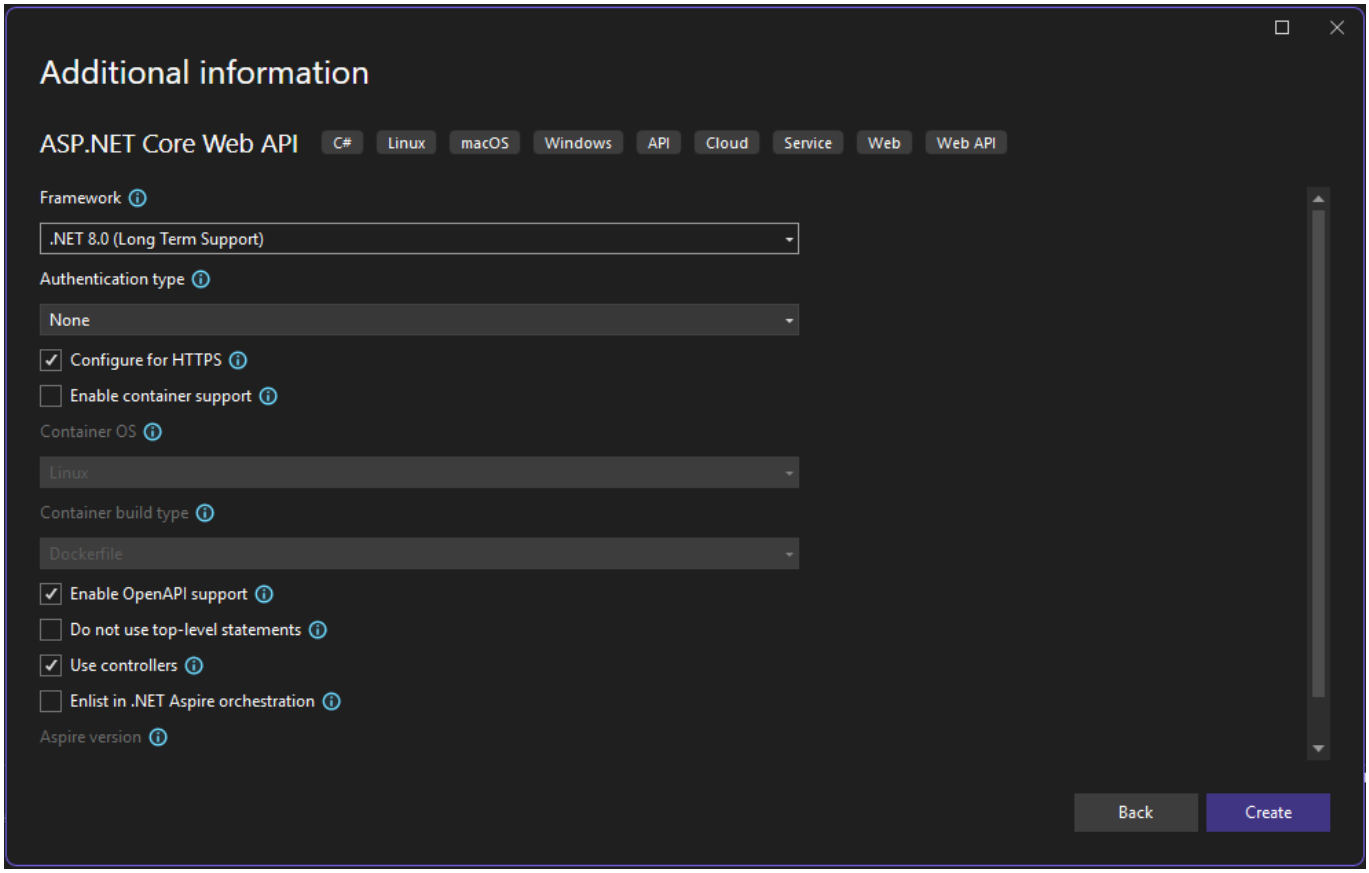
- Em Project name, insira um nome descritivo. Neste exemplo, utilizamos, `OracleCrud.Sp.Api`, indicando que o projeto é um CRUD com Oracle utilizando Stored Procedures;
- Em Location, escolha a pasta onde o projeto será armazenado. No nosso caso, estamos organizando dentro da estrutura `src/` do repositório.

> **Dica:** Manter os projetos dentro de uma pasta src é uma boa prática para repositórios que podem ter múltiplos projetos no futuro.

Após configurar, clique em **Next** para prosseguir com a escolha da versão do .NET.

Definindo as Configurações Adicionais do Projeto

Na última etapa da criação do projeto, temos a tela “Additional information”, onde configuramos detalhes importantes da nossa ASP.NET Core Web API. Veja abaixo as opções recomendadas para o nosso cenário com .NET 8 e Oracle XE:



- **Framework:** Selecione o `.NET 8 (Long Term Support)`, que oferece suporte de longo prazo e é ideal para novos projetos.
- **Authentication type:** Marque como None, já que neste exemplo não implementamos autenticação.
- **Configure for HTTPS:** Mantenha marcado para garantir a segurança nas requisições.
- **Enable OpenAPI support:** Marque essa opção para que o Swagger seja configurado automaticamente, facilitando testes na API.
- **Use controllers:** Certifique-se de que está marcado, pois usaremos controllers no estilo tradicional (e não minimal APIs).

Por fim, clique em Create para gerar sua API REST. Pronto! Agora temos a estrutura inicial do nosso projeto ASP.NET Core com suporte a Swagger e pronto para integrar com Oracle via Stored Procedures.

Limpando a Estrutura Inicial do Projeto

Após a criação da PAI, o Visual Studio gera alguns arquivos de exemplo que não serão utilizados no projeto. Para manter a organização e focar apenas no que é essencial, recomendamos excluir os seguites

arquivos:

Arquivos a serem removidos:

- `OracleCrud.Sp.Api.http`: Usado para testes HTTP via Visual Studio, mas será desnecessário no nosso fluxo.
- `Controllers/WeatherForecastController.cs`: Um controller gerado automaticamente com lógica fictícia.
- `WeatherForecast.cs`: Classe modelo de exemplo que não utilizaremos.

Para excluir, no **Solution Explorer**, clique com o botão direito do mouse em cada arquivo, selecione a opção **Delete** e confirme a exclusão se solicitado.

Separando responsabilidades com DTOs distintos

Para manter a clareza e a responsabilidade única de cada classe em nossa API, utilizamos dois **Data Transfer Objects** (**DTOs**) distintos: O `CreateUserDto` e o `UserDto`. O `CreateUserDto` é usado exclusivamente para representar os dados recebidos do frontend no momento da criação de um novo usuário. Ele não inclui o Id, pois esse campo é gerado internamente pela API ou pelo banco de dados. Já o `UserDto`, representa o objeto completo, incluindo o `Id`, e é utilizado nas respostas ou em operações internas onde o identificador já está presente. Essa separação melhora a coesão do código e facilita a manutenção e os testes da aplicação.

```
// DTO recebido do Frontend (para criação de usuário)
namespace OracleCrud.Sp.Api.Domain.Dtos;

public class CreateUserDto
{
    public string Name { get; set; } = default!;
    public string Email { get; set; } = default!;
}
```

```
// DTO utilizado internamente (com Id)
namespace OracleCrud.Sp.Api.Domain.Dtos;
```

```
public class UserDto
{
    public string Id { get; set; } = default!;
    public string Name { get; set; } = default!;
    public string Email { get; set; } = default!;
}
```

Por que os DTOs estão na pasta Domain?

Colocamos os **DTOs** na pasta **Domain** porque, nesse projeto, não estamos utilizando entidades de domínio mapeadas com **ORM** como o **Entity Framework**. Em vez disso, nossos objetos centrais de comunicação com a API(como `UserDto` e `CreateUserDto`) representam o "contrato" de dados que a aplicação conhece e manipula, sendo parte da camada de domínio. Isso reforça a ideia de que o domínio não é só composto por entidades persistidas, mas também pelos tipos que expressam regras e estruturas de negócio, mesmo que simplificadas.

Definindo Contratos com Interfaces: Separando Regras de Negócio e Persistência

Na estrutura desta API, criamos duas interfaces fundamentais para garantir a separação de responsabilidade `IUserService` e `IUserRepository`. A `IUserService` representa o contrato da camada de serviço, responsável por orquestrar as regras de negócio da aplicação. Já a `IUserRepository` define o contrato da camada de persistência de dados, isolando a forma como acessamos o Oracle Database através de stored procedures.

```
using OracleCrud.Sp.Api.Domain.Dtos;

namespace OracleCrud.Sp.Api.Application.Interfaces;

public interface IUserService
{
    Task<IEnumerable<UserDto>> GetAllAsync();
    Task<string> InsertAsync(CreateUserDto user);
    Task<string> UpdateAsync(UserDto user);
    Task<string> DeleteAsync(string id);
}
```

Ambas as interfaces foram posicionadas na pasta `Application/Interfaces`, pois essa camada representa a lógica da aplicação (application layer), isto é , o ponto de comunicação entre as regras de negócio e as implementações concretas.

```
using OracleCrud.Sp.Api.Domain.Dtos;

namespace OracleCrud.Sp.Api.Application.Interfaces;

public interface IUserRepository
{
    Task<IEnumerable<UserDto>> GetAllAsync();
    Task<string> InsertAsync(UserDto user);
    Task<string> UpdateAsync(UserDto user);
    Task<string> DeleteAsync(string id);
}
```

Manter as interfaces nesse local centraliza os contratos da aplicação, facilitando a leitura, os testes e futuras substituições de implementação (por exemplo, trocar Oracle por outro banco, sem impactar as camadas superiores). Essa decisão segue os princípios da arquitetura limpa promovendo uma baixo acoplamento e e facilitando a manutenção da solução.

Conclusão

Nesta **Parte 1** da série *Construindo a API REST em .NET com Oracle XE e Stored Procedures*, iniciamos o desenvolvimento da aplicação partindo totalmente do zero. Realizamos a seguintes etapas:

- Criamos o projeto com o template ASP.NET Core Web API no Visual Studio 2022;
- Limpamos a estrutura inicial removendo arquivos de exemplo como WeatherForecast.cs e WeatherForecastController.cs;
- Organizamos a base do projeto como uma estrutura em camadas: `Domain`, `Application`, `Infrastructure` e `Controllers`;
- Definimos os DTOs `CreateUserDto` e `UserDto`, explicando seus papéis e porque estão posicionados dentro da camada de domínio;
- Criamos as interfaces `IUserService` e `IUserRepository`, que representam os contratos da camada de aplicação.
- Justificamos a separação das interfaces em Application/Interfaces, mantendo o foco em uma

arquitetura limpa e desacoplada.

Na **Parte 2**, vamos dar continuidade à implementação, criando os repositórios concretos, integrando com o banco **Oracle XE** por meio de Stored Procedures e finalizando os endpoints da nossa **API REST**.