Empréstimos Online

Documentação da Arquitetura

Histórico de Versões

Data	Versão	Descrição	Autor	Revisor
06/05/2021	1.0	Versão inicial	Túlio Paim	Túlio Paim

1. Sumário

2. Introdução

2.1. Finalidade

Documentar a arquitetura do sistema Empréstimos Online.

2.2. Definições

2.2.1. Idioma

Em relação ao idioma utilizado no desenvolvimento do projeto, a estratégia utilizada é utilizar o Inglês para termos técnicos específicos e Português para os termos do Domínio.

Ex: FornecedorRepository, sendo Fornecedor um termo próprio do Domínio, escrito em português, e Repository um termo técnico, escrito em inglês.

2.2.2. Projetos

A nomenclatura segue o padrão EO. [Nome Camada], sendo EO a sigla para Empréstimos Online.

Ex: EO. Presentation, EO. Application, EO. Domain, EO. Infra ....

2.2.3. Nomenclaturas

- **Entidades:** Classes de negócio do nosso domínio, possuem suas próprias tabelas no banco de dados.
- **ViewModels:** Classes utilizadas para trafegar apenas as informações necessárias em determinado input/output no sistema.

2.3. Visão Geral

O Empréstimos Online é um sistema que visa intermediar pessoas que procuram empréstimos e pessoas que estão dispostas a emprestar o dinheiro com determinada taxa de juros.

3. Representação Arquitetural

A aplicação possui o Postgres como banco de dados e é dividida em 4 camadas:

- **Presentation:** Camada de apresentação, responsável pela comunicação com o usuário, UI.
- **Application:** Camada de aplicação, responsável por orquestrar as camadas de domínio e infra, definir as classes de ViewModels que serão utilizadas pela camada de apresentação e realizar o mapeamento das classes do Domínio para as classes de ViewModels
- **Domain:** Camada de domínio da aplicação, camada central da arquitetura, responsável pelas Entidades, regras de negócios e definição dos contratos.
- **Infra:** Camada de infra, responsável pelo acesso ao banco de dados, implementa os contratos definidos na camada de domínio.

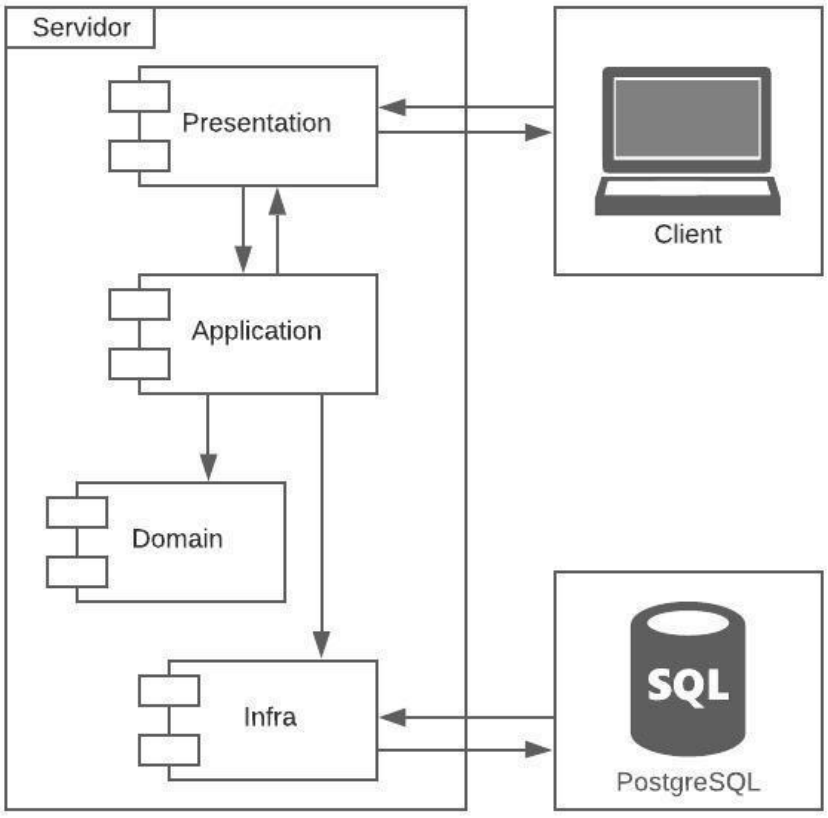


Figura 1 - Diagrama de Fluxo

4. Metas e Restrições da Arquitetura

A aplicação foi definida como uma aplicação web, disponível apenas em browsers, com possibilidade de expansão para o universo mobile.

O projeto será desenvolvido na plataforma .NET, plataforma madura e robusta para o desenvolvimento web. O projeto de apresentação (site) será um WebApp MVC, possibilitando a criação do site e futuros endpoints para servir como API. As demais camadas serão bibliotecas de classe.

Para o acesso ao banco de dados será utilizado o ORM Entity Framework Core 5, um framework da própria Microsoft que abstrai e facilita o acesso ao banco. O banco de dados utilizado é o Postgres SQL.

A motivação para a escolha destas tecnologias é o conhecimento do time, maturidade da plataforma, grande quantidade de bibliotecas disponíveis e o tamanho da comunidade.

5. Visão de Casos de Uso

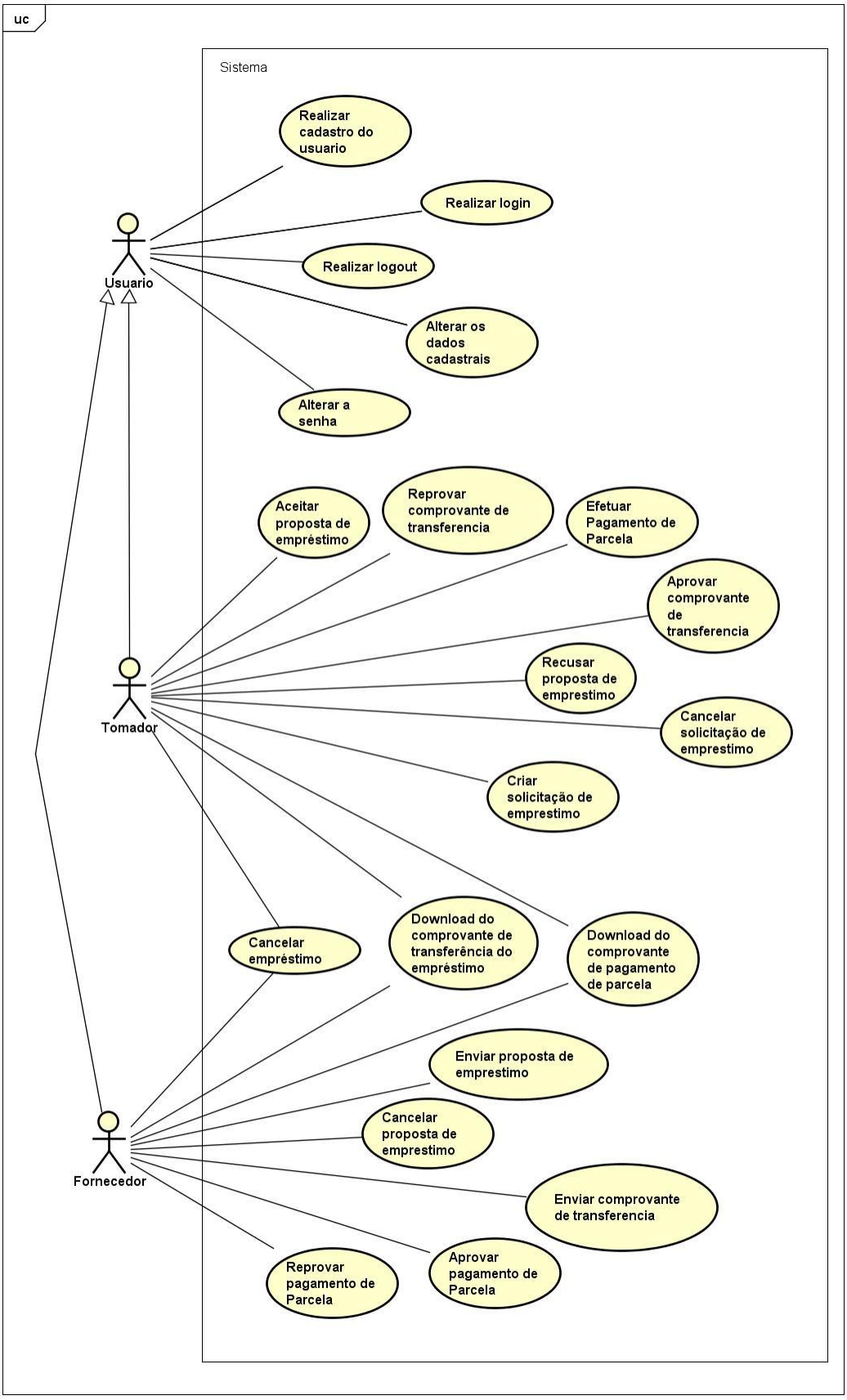


Figura 2 - Visão de Casos de Uso

6. Visão Lógica

6.1. Visão Geral

O design arquitetural em camadas foi escolhido pela escalabilidade e baixo acoplamento entre as camadas, inspirado nos conceitos de DDD e Clean Architecture.

6.2. Pacotes

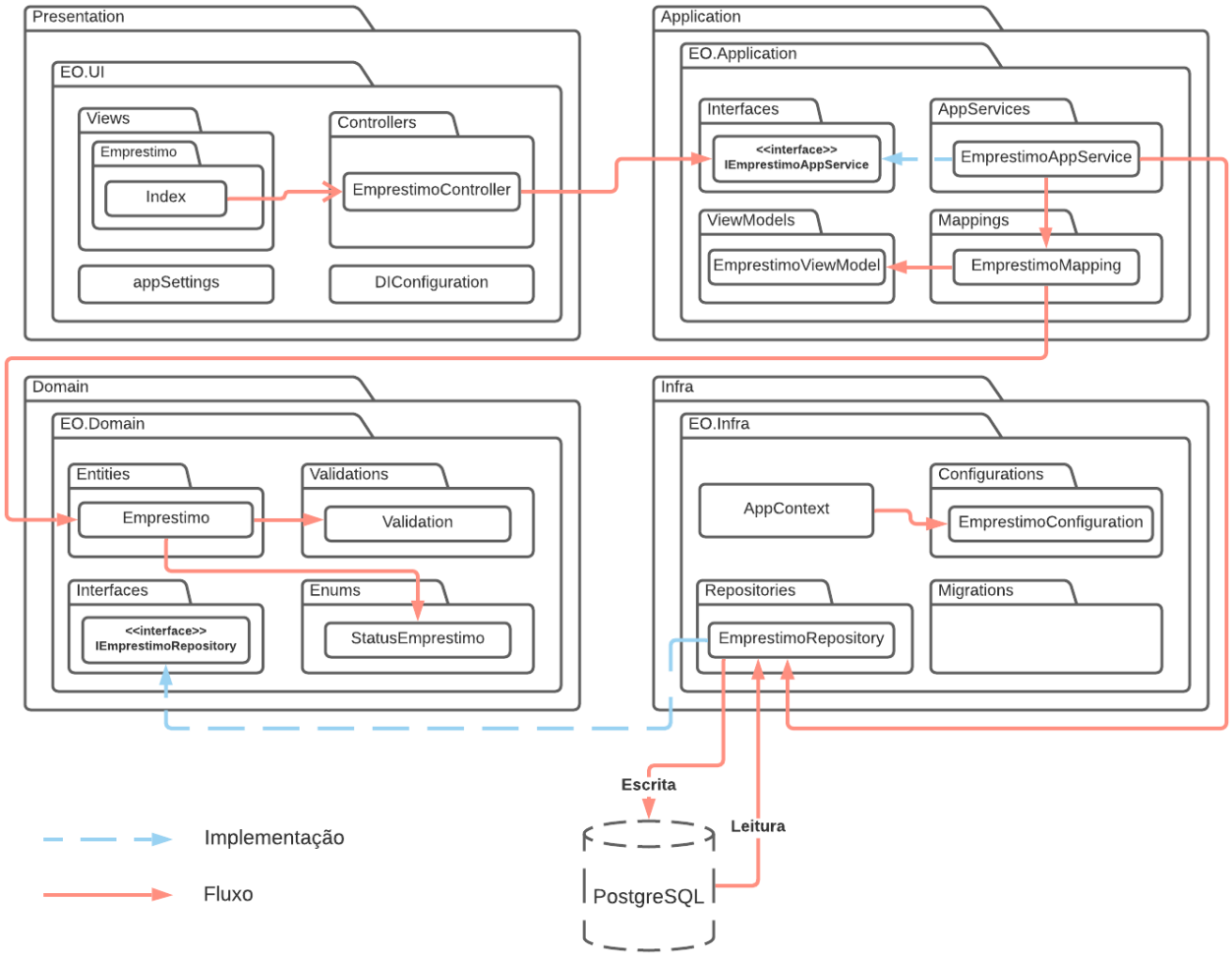


Figura 3 - Visão Geral de Implementação e Fluxo

7. Visão de Processos

Como exibido na *Figura 3,* o processo de fluxo de dados de uma ação do Usuário é feito da seguinte maneira:

Tomando como exemplo o **CDU11 – Cancelar Empréstimo,** o usuário entra na **View** de detalhe de empréstimo e clica em "*Cancelar Empréstimo*".

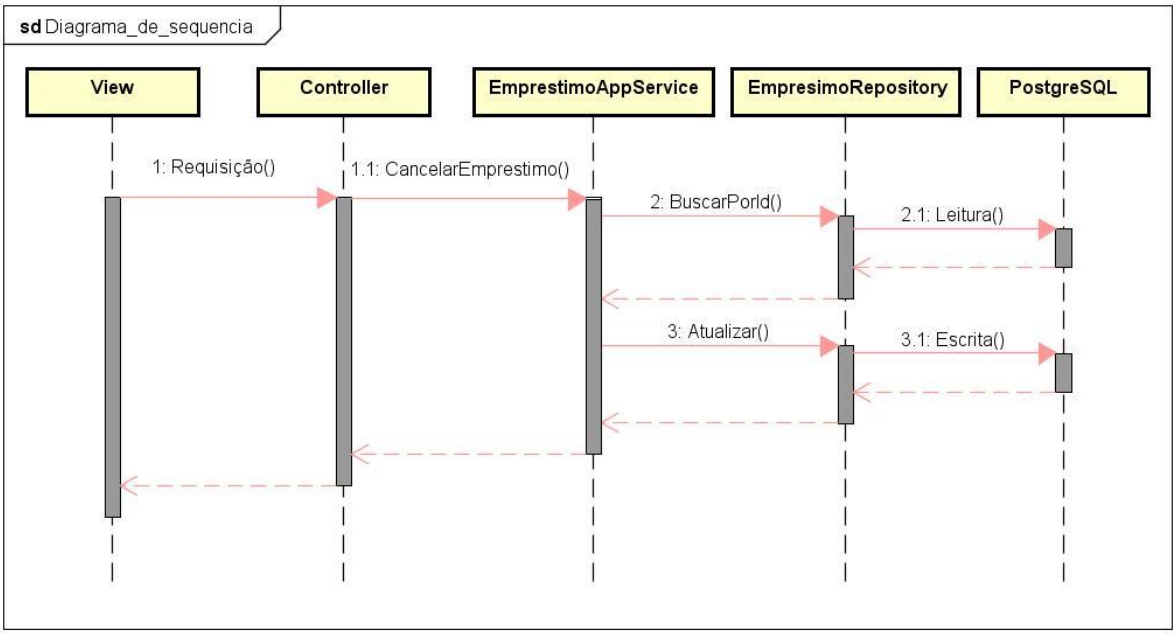


Figura 4 - Diagrama de Sequencia

A **View** envia uma requisição para o **Controller**, que por sua vez chama o método "**CancelarEmpréstimo",** presente na Interface **IEmprestimoAppService** implementada pela classe **EmprestimoAppService.**

A função por sua vez chama o método "**BuscarPorId"**, presente na interface **IEmprestimoRepository** implementada pela classe **EmprestimoRepository.** O método "**BuscarPorId"** recebe o Id do empréstimo e realiza uma consulta no banco de dados, retornado a **Entidade Empréstimo**.

O status do Empréstimo é validado, e se tudo estiver de acordo com as regras de negócio, é invocado o método "**Cancelar",** que altera o **StatusEmprestimo** para **Cancelado.**

Após isso, a Entidade Empréstimo é enviada para o método “**Atualizar”** da interface **IEmprestimoRepository**, que grava a atualização no banco de dados.

O **EmprestimoAppService** retorna que a operação foi bem sucedida para o **Controller,** que redireciona o usuário para a **View** de Listagem de Emprestimos.

8. Visão de Implantação

Um container para a Aplicação (WebApp .NET Core) se conectando com o container do banco de dados (PostgreSQL).

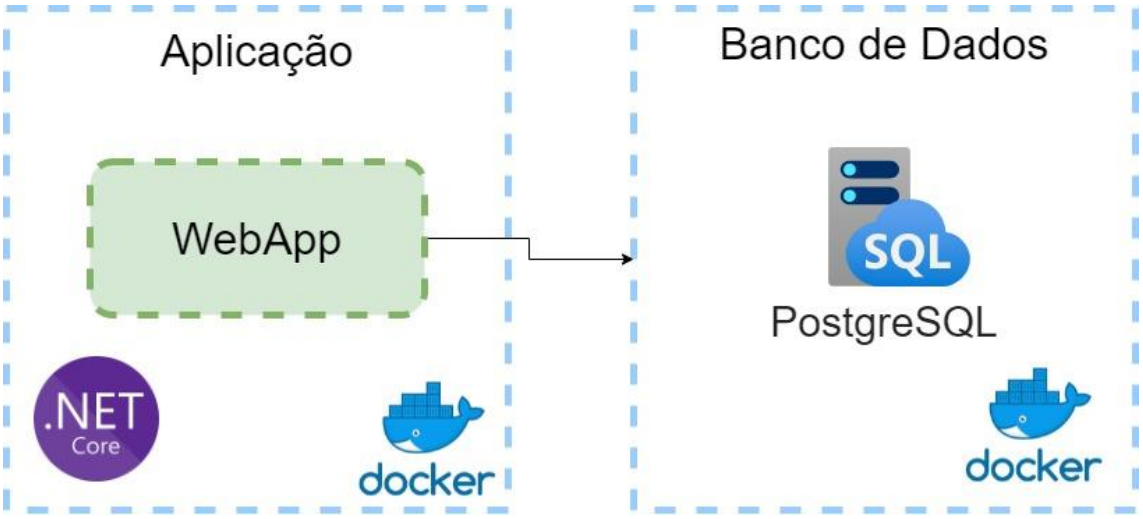


Figura 5 - Visão de Implantação

9. Visão de Implementação

9.1. Visão Geral

Uma visão detalhada sobre cada uma das camadas da aplicação e suas responsabilidades.

9.2. Camadas

9.2.1. Camada de Presentation (Apresentação)

Na camada de apresentação temos as **Views**, **Controllers** e as **Configurations**.

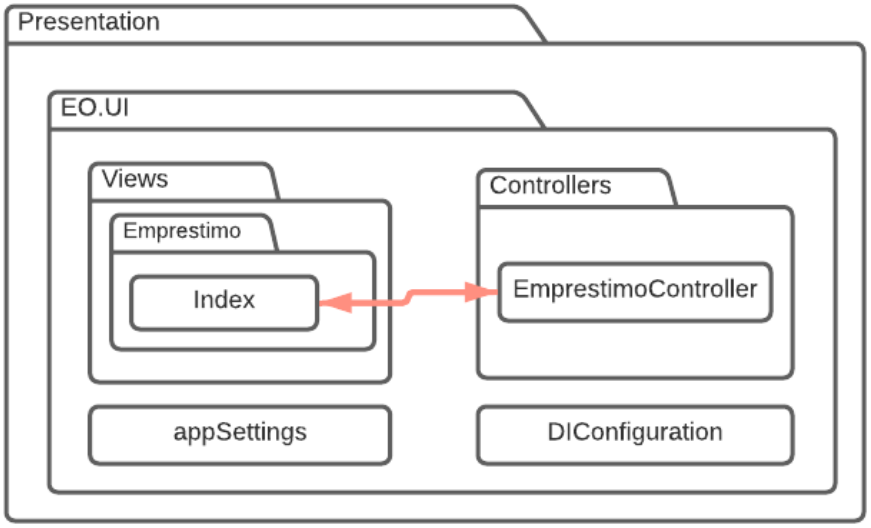


Figura 6 - Diagrama de Presentation

As **Views** são as telas de nosso site, responsáveis por exibir dados e receber inputs do usuário. As views enviam estes inputs para os **Controllers**, que por sua vez retornam as informações para a **View**.

Nas **Configurations** ficarão as configurações do nosso site, como por exemplo as configurações de autenticação e autorização, a injeção do contexto e as injeções de dependência.

O próprio framework fornece um container de injeção de dependência, onde podemos injetar implementações para cada

interface, e utilizar estas interfaces para ter acesso às instancias das implementações em outras camadas.

9.2.2. Camada de Application (Aplicação)

A camada de aplicação é responsável por orquestrar as operações de leitura e escrita da aplicação, ela contém as **Interfaces**, **AppServices**, **ViewModels** e **Mappings**.

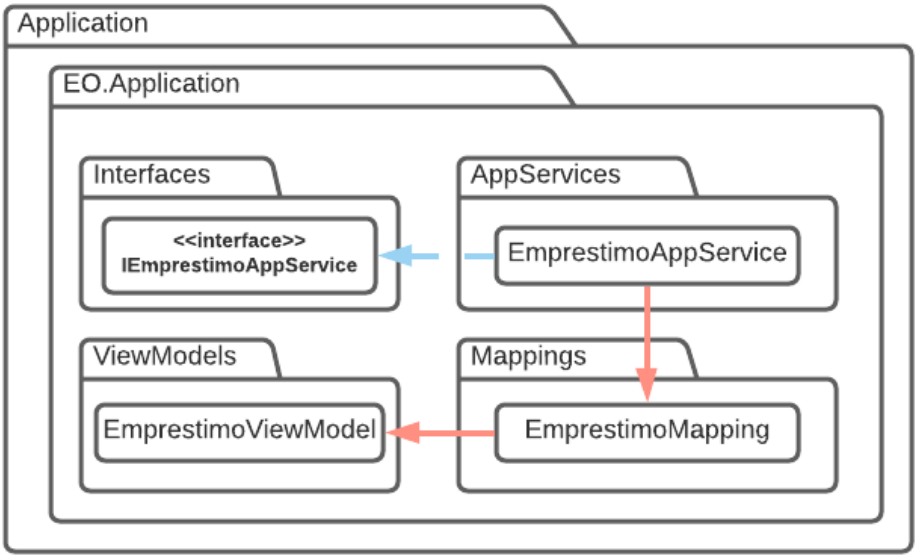


Figura 7 - Diagrama de Application

O **controller** da camada de apresentação acessa a camada de Application por meio das **Interfaces,** implementadas pelos **AppServices**.

Os **AppServices** são responsáveis por invocar as interfaces de repositório, contidas na camada de domínio, e mapear as entidades retornadas para as ViewModels esperadas na camada de apresentação.

As configurações de mapeamento entre as Entidades e as ViewModels ficam na pasta **Mappings.**

A camada de aplicação depende apenas da camada de Domínio, que disponibiliza interfaces de repositório para acesso indireto a camada de infra.

9.2.3. Camada de Domain (Domínio)

Na camada de domínio temos as **Entities, Enums, Validations e Interfaces.**

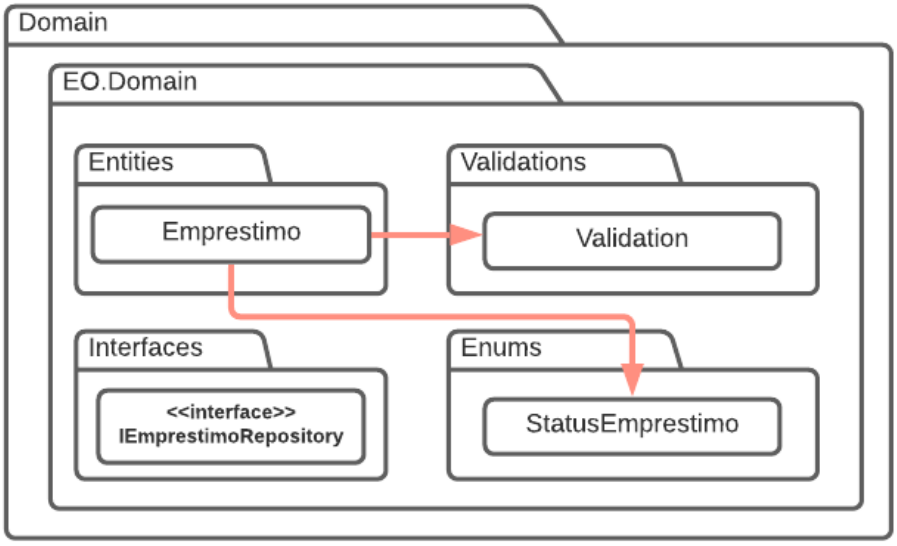


Figura 8 - Diagrama do Domain

As **Entities** representam os objetos de negócio, como Usuário, Fornecedor e Empréstimo. Cada entidade deve conter regras de negócio para sua criação, manipulação e validação.

Para realizar a validação, a entidade deve fazer referência a sua classe de validação contida em **Validations,** as classes de validação utilizam do FluentValidation (pacote) para definir suas regras.

Os **Enums** representam os enumeradores, como StatusEmprestimo, StatusParcela e etc.

Nas **Interfaces** estão contidos os contratos para os repositórios, a camada de aplicação conversa com a camada de infra através destes contratos, mantendo o desacoplamento entre a camada de infra e o restante da aplicação.

A camada de domínio não depende de nenhuma outra camada.

9.2.4. Camada de Infra

Na camada de infra é responsável pelo acesso ao banco de dados, nela temos o **Context, Configurations, Migrations e Repositories.**

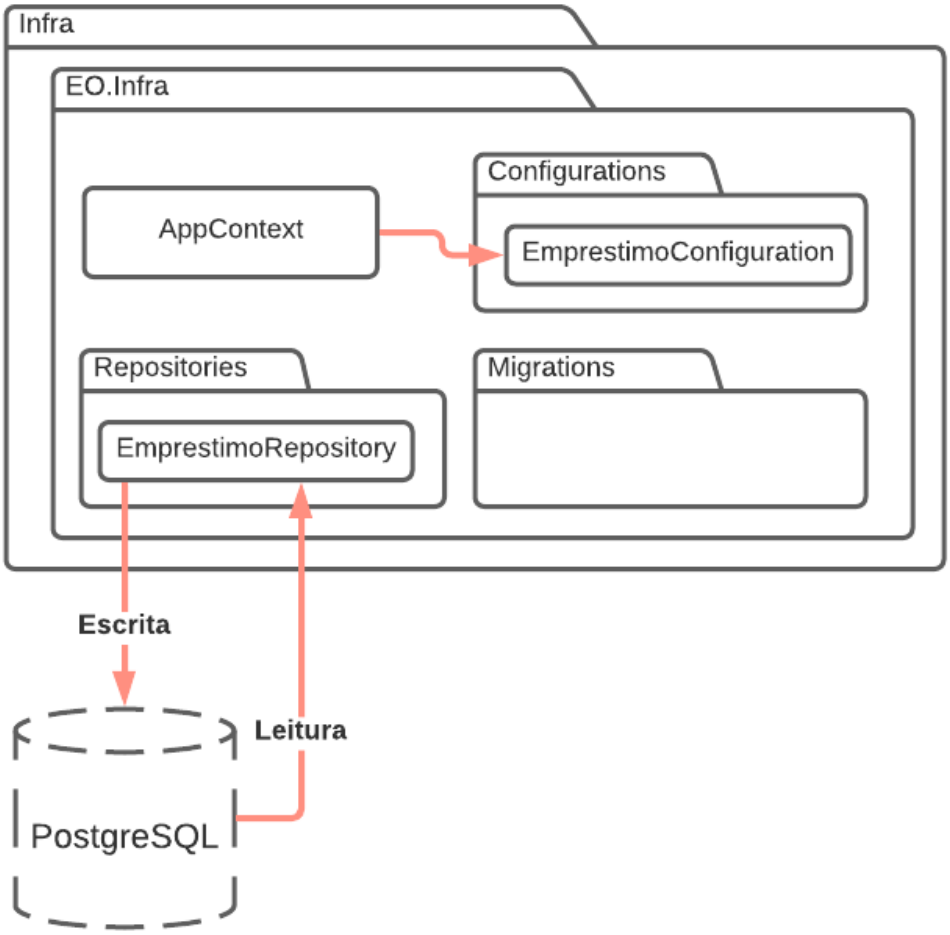


Figura 9- Diagrama de Infra

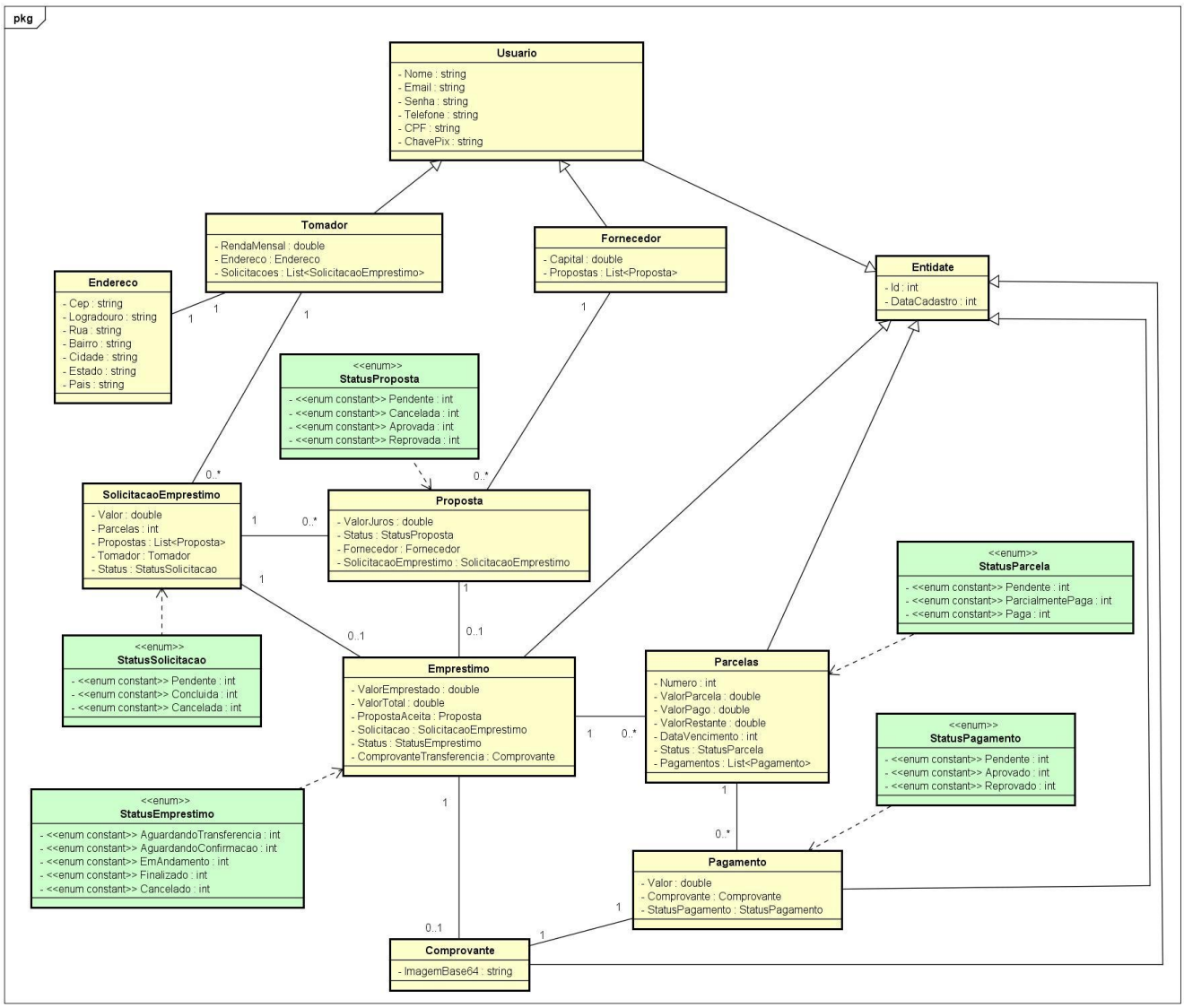
A camada de infra faz uso do **ORM Entity Framework** para o acesso a banco. Temos a classe **Context,** que representa o contexto da aplicação com o banco, nela definimos quais as entidades são mapeadas e chamamos cada classe de configuração.

Em **Configurations,** temos as classes de configuração de mapeamento de cada entidade com o banco, é nela que definimos, por exemplo, o tipo de cada coluna, nome da tabela, relacionamento entre tabelas e etc.

O Entity framework gera os scripts de migração automaticamente a partir da linha de comando e os armazena na pasta **Migrations.**

Em **Repositories** temos a implementação das interfaces da camada de domínio, onde estão armazenadas as funções de leitura e escrita no banco.

10. Visão de Dados



11. Tamanho e Desempenho

Por se tratar de uma aplicação web o tamanho da mesma não é definido, e seu desempenho deve seguir o que foi especificado no documento de requisitos.

12. Qualidade

A qualidade da aplicação deve seguir o que foi proposto nos requisitos não funcionais no documento de requisitos.

13. Referências

- **Clean Architecture com ASP.NET Core**
  https://www.luisdev.com.br/2020/09/29/clean-architecture-com-asp-net-core-parte-1/
- **Uma arquitetura, em .Net Core, baseada nos princípios do DDD**
  https://alexalvess.medium.com/criando-uma-api-em-net-core-baseado-na-arquitetura-ddd-2c6a409c686
- **Arquivo exemplo - Documento Arquitetura de Referência - Aplicações Demoiselle**