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. Introdução

1.1. Finalidade

Documentar a arquitetura do sistema Empréstimos Online.

1.2. Definições

1.2.1. 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.

1.2.2. 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 ....

1.2.3. Nomenclaturas

Entidades: Classes de negócio do nosso domínio, possuem próprias no banco de dados.

ViewModels: Classes utilizadas para trafegar apenas as informações necessárias em determinado input/output no sistema.

1.3. 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.

2. 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.

3. 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 projeto 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.

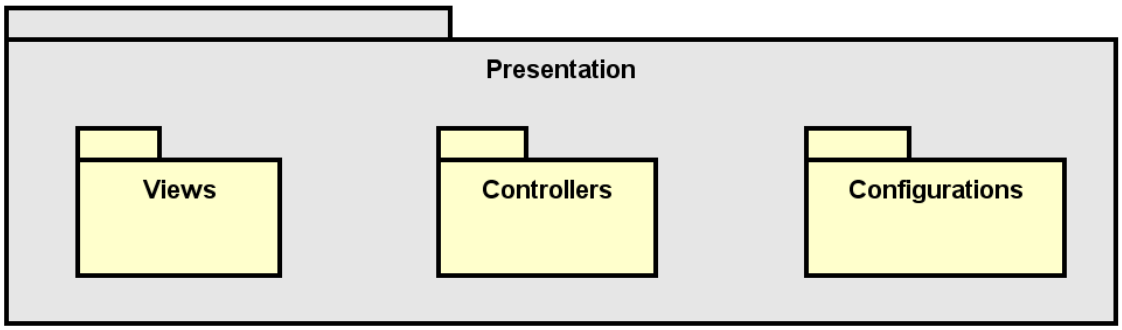
4. Visão Lógica

4.1. 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.

4.2. Camada de Presentation

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



As **Views** são as telas de nosso site, elas exibem os dados e recebem os inputs do usuário.

As **Views** enviam as informações para os **Controllers**, e eles por sua vez retornam as informações para a **View**.

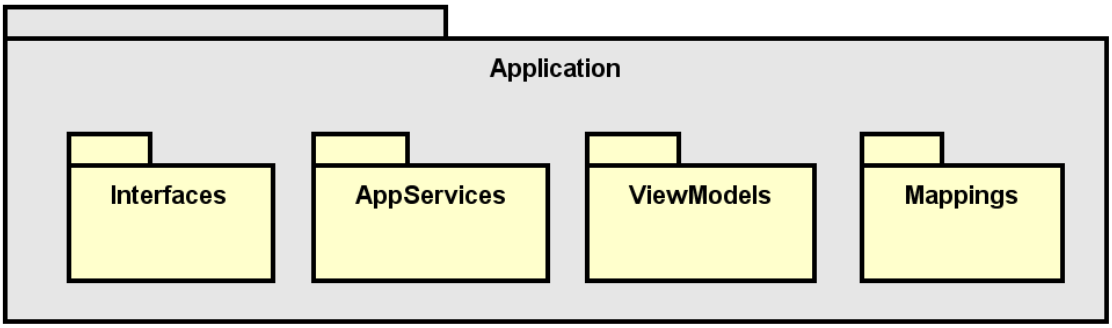
Nas **Configurations** ficarão as configurações do nosso site, como por exemplo as configurações de autenticação e autorização, a injeção do contexto e as injeções de dependência.

O próprio framework fornece um container de injeção de dependência, onde podemos injetar implementações para interfaces, e utilizar estas interfaces para ter acesso às instancias das implementações em outras camadas.

A camada de Presentation faz referência a todas as camadas do sistema.

4.3. Camada de Application

Na camada de aplicação temos as **Interfaces, AppServices, ViewModels e Mappings.**



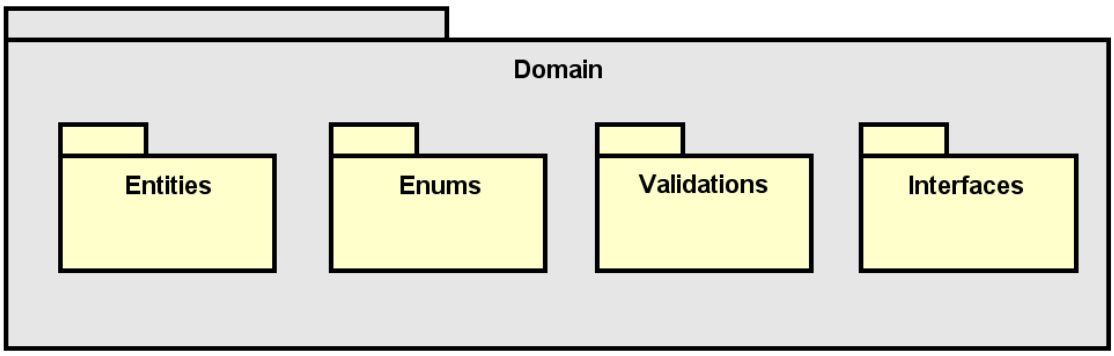
O controller da camada de apresentação utiliza as **Interfaces** para realizar qualquer procedimento ou leitura no sistema.

As **Interfaces** definem o contrato implementado pelos **AppServices**, que por sua vez mapeiam as **ViewModels** para as entidades e vice e versa, orquestrando as leituras e escritas no banco.

A camada de aplicação depende apenas da camada de Domínio, conversando com o banco através das interfaces de repositório existentes na camada de domínio.

4.4. Camada de Domain

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



As **Entities** representam os objetos de negócio, como Usuário, Fornecedor, Empréstimo e etc.

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

Cada entidade é responsável pela sua criação, manipulação e validação.

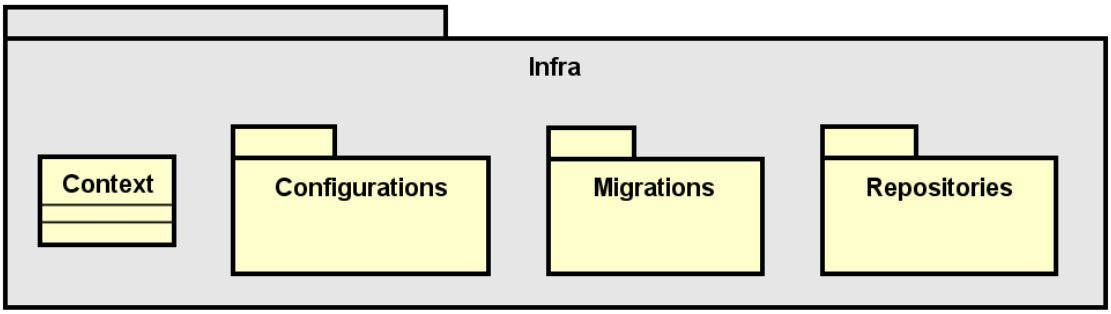
Para realizar a validação, a entidade deve fazer referência a uma classe de validação contida em **Validations,** as classes de validação utilizam do FluentValidation para definir suas regras.

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.

4.5. Camada de Infra

Na camada de infra temos o **Context, Configurations, Migrations e Repositories.**



A camada de infra faz uso do ORM Entity Framework para o acesso a banco, na sua estrutura temos a classe **Context,** que representa o contexto da aplicação com o banco, nele definimos quais as entidades são mapeadas, dentre outras configurações.

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.

5. **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.

6. **Qualidade**

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

7. **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**