

# Sistema de Estoque

20/07/20

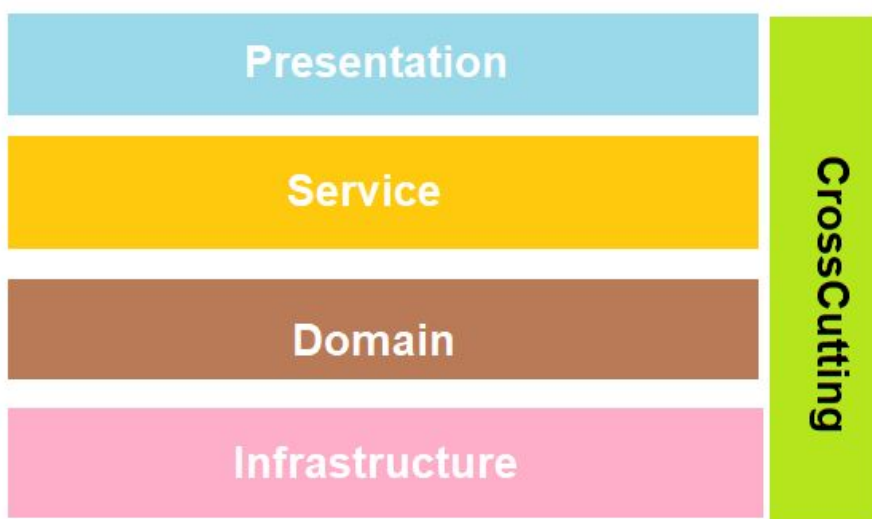
Isaque Seabra

## Visão geral

Para contemplar a historia do usuário “Eu como estoquista desejo cadastrar/alterar e excluir produtos no meu sistema de estoque” foi desenvolvido uma API .NET Core 3.1 e um front-end com React.js

## Arquitetura API Rest

Para a api, eu pensei em uma arquitetura não muito complexa, com 5 camadas, tentando deixar o mais próximo do que seria uma arquitetura para se encaixar no padrão DDD.



### I. Presentation

Essa camada é responsável pela API, é nela onde a requisição de entrada de dados chega, ou seja é nossa porta de comunicação, foi instalado no projeto o **Swashbuckle.AspNetCore** que é uma biblioteca para auxiliar no uso do swagger.

## II. Service

É a camada responsável pela regra de negócio propriamente dita, ela tem acesso a camada de domínio e de infra para assim conseguir orquestrar o funcionamento da aplicação.

## III. Domain

Essa camada ficou responsável por possuir todas **Entidades, Interfaces e Validações** que no caso não foram inseridas por conta de tempo ... Mas eu iria utilizar o fluent-validation pra ajudar nessa tarefa de validação.

## IV. Infrastructure

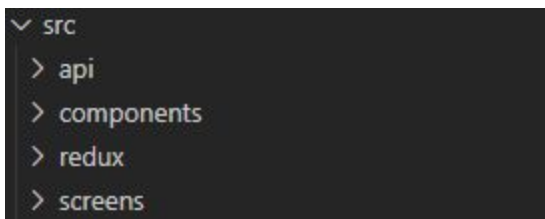
Nessa camada ficou o acesso a dados, nesta camada foi implementado o padrão **Repository** que também auxilia na redução de acoplamento do projeto.

## V. CrossCutting

Uma camada que auxilia o projeto, podendo passar por todas as camadas, foi inserida nessa camada a **injeção de dependência**, para desacoplar o projeto e facilitar na camada de Teste.


## React.js Front-end

A organização do projeto ficou dessa maneira:



**api** > Onde fica a base para realizar as requisições para a nossa API, foi utilizada a biblioteca **axios** para facilitar essa tarefa.

**componentes** > São todos os componentes que foram escritos para facilitar o uso na aplicação, e para isso foi utilizado o **material-ui**, onde boa parte dos componentes estão prontos e eu acho bem bonitos para serem utilizados em uma aplicação.



**redux** > Na gestão do contexto da aplicação foi utilizado o redux, dentro dessa pasta estão a Store, Reducer e as Actions da aplicação, e também foi utilizado o **redux-thunk**, para auxiliar nas chamadas das apis e esperar o retorno para executar o diptach.

**screen** > São onde as telas do projeto estão, no caso todas relacionadas com o estoque, e também foi utilizado o **material-ui** para auxiliar no desenho das telas.

***OBS: Infelizmente as telas ficaram sem resposta visual as ações do usuário .... Mas todas elas estão ocorrendo(Editar,Salvar,Excluir).***