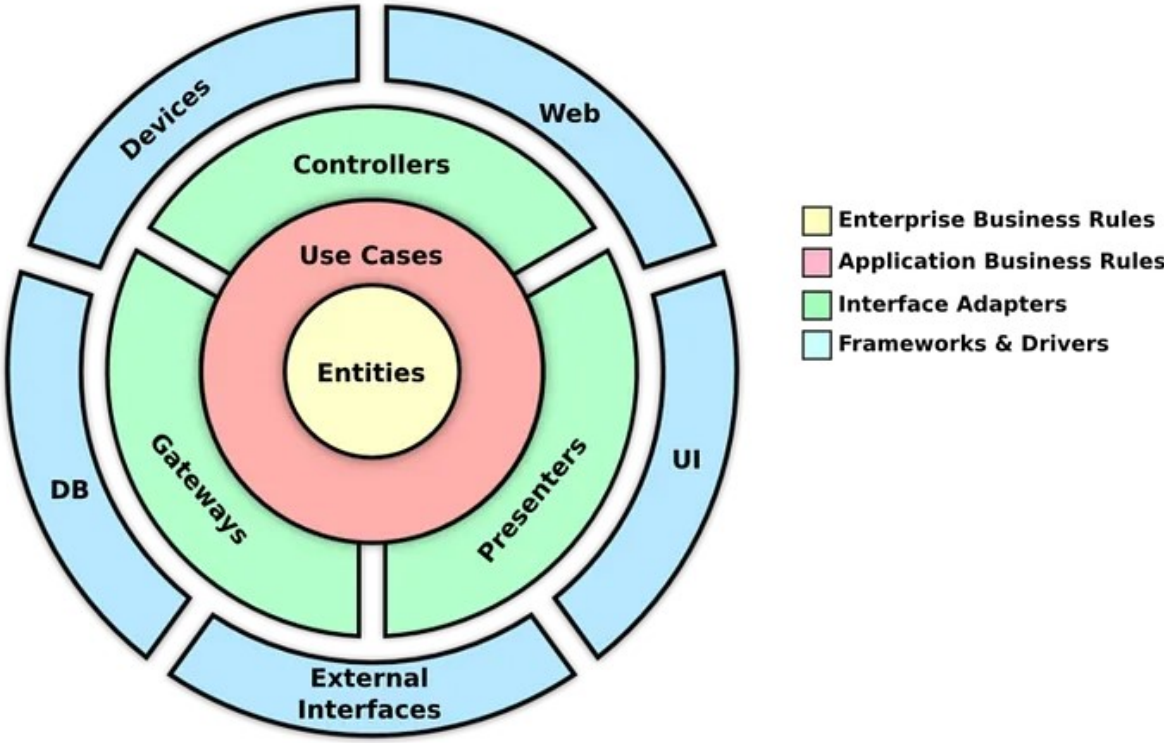
Projeto Hackaton

STATUS EM DESENVOLVIMENTO

A Health&Med, é uma Operadora de Saúde que tem como objetivo digitalizar seus processos e operação. O principal gargalo da empresa é o Agendamento de Consultas Médicas, que atualmente ocorre exclusivamente através de ligações para a central de atendimento da empresa.
Recentemente, a empresa recebeu um aporte e decidiu investir no desenvolvimento de um sistema proprietário, visando proporcionar um processo de Agendamentos de Consultas Médicas 100% digital e mais ágil.
Para viabilizar o desenvolvimento de um sistema que esteja em conformidade com as melhores práticas de desenvolvimento, a Health&Med contratou os alunos e alunas da turma de .NET da FIAP para realizar a análise do projeto e desenvolver o MVP da solução.
Assim, o objetivo do Hackathon é a entrega de um produto de MVP desenvolvido e que cumpra os requisitos funcionais e não funcionais.

Arquitetura

O projeto foi desenvolvido baseado na Clean Architecture.



O principal objetivo da Arquitetura Limpa é a Regra de Dependência,
essa regra tem tudo a ver com a direção que nossas dependências devem apontar, ou seja,
sempre para as políticas de alto nível.

Um objetivo importante da Clean Architecture é fornecer aos desenvolvedores uma maneira de organizar o código de forma que encapsule a lógica de negócios, mas mantenha-o separado do mecanismo de entrega.

Vantagens:

As vantagens de utilizar uma arquitetura em camadas são muitas, porém podemos pontuar algumas:

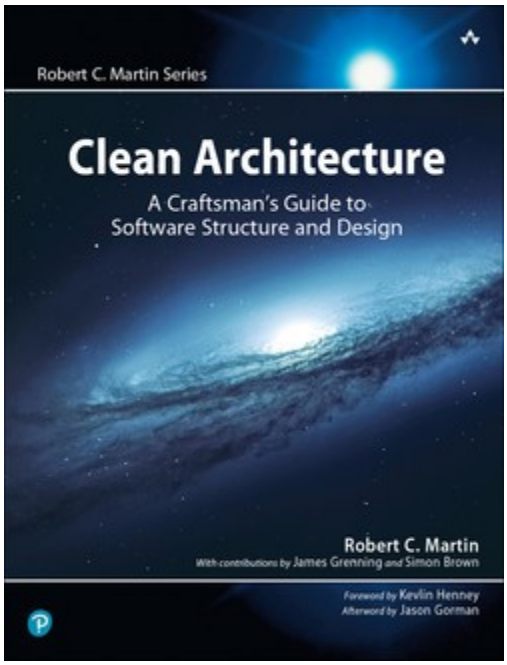
- Testável: As regras de negócios podem ser testadas sem a interface do usuário, banco de dados, servidor ou qualquer outro elemento externo.
- Independente da interface do usuário: A interface do usuário pode mudar facilmente, sem alterar o restante do sistema. Uma UI da Web pode ser substituída

por uma UI do console, por exemplo, sem alterar as regras de negócios.
- Independente de banco de dados: Você pode trocar o Postgres ou SQL Server, por Mongo, CosmosDb ou qualquer outro. Suas regras de negócios não estão vinculadas ao banco de dados.
- Independente de qualquer agente externo: Na verdade, suas regras de negócios simplesmente não sabem nada sobre o mundo exterior, não estão ligadas a nenhum Framework.

Referências

Blog do Uncle Bob: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

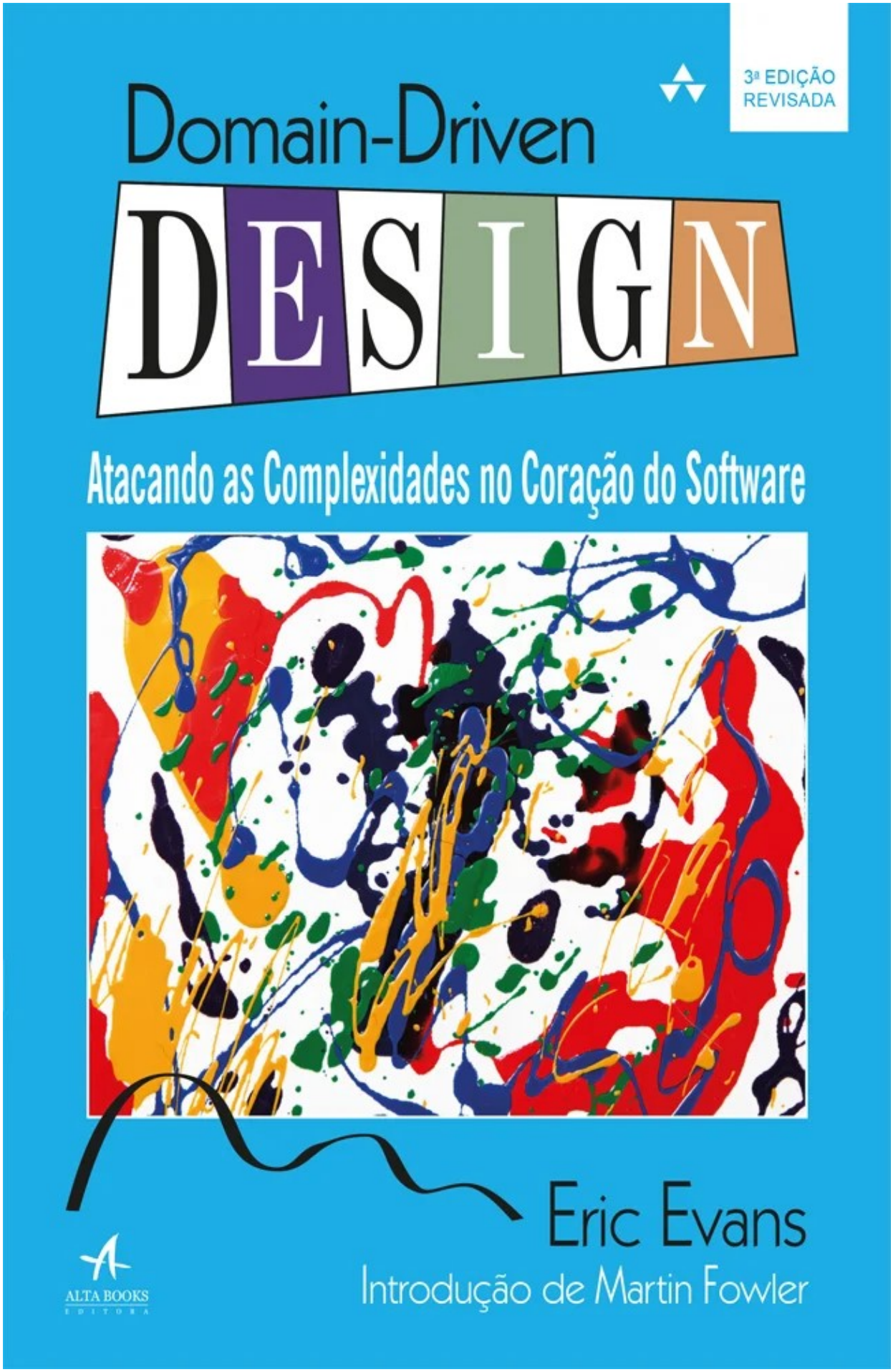
Livro:

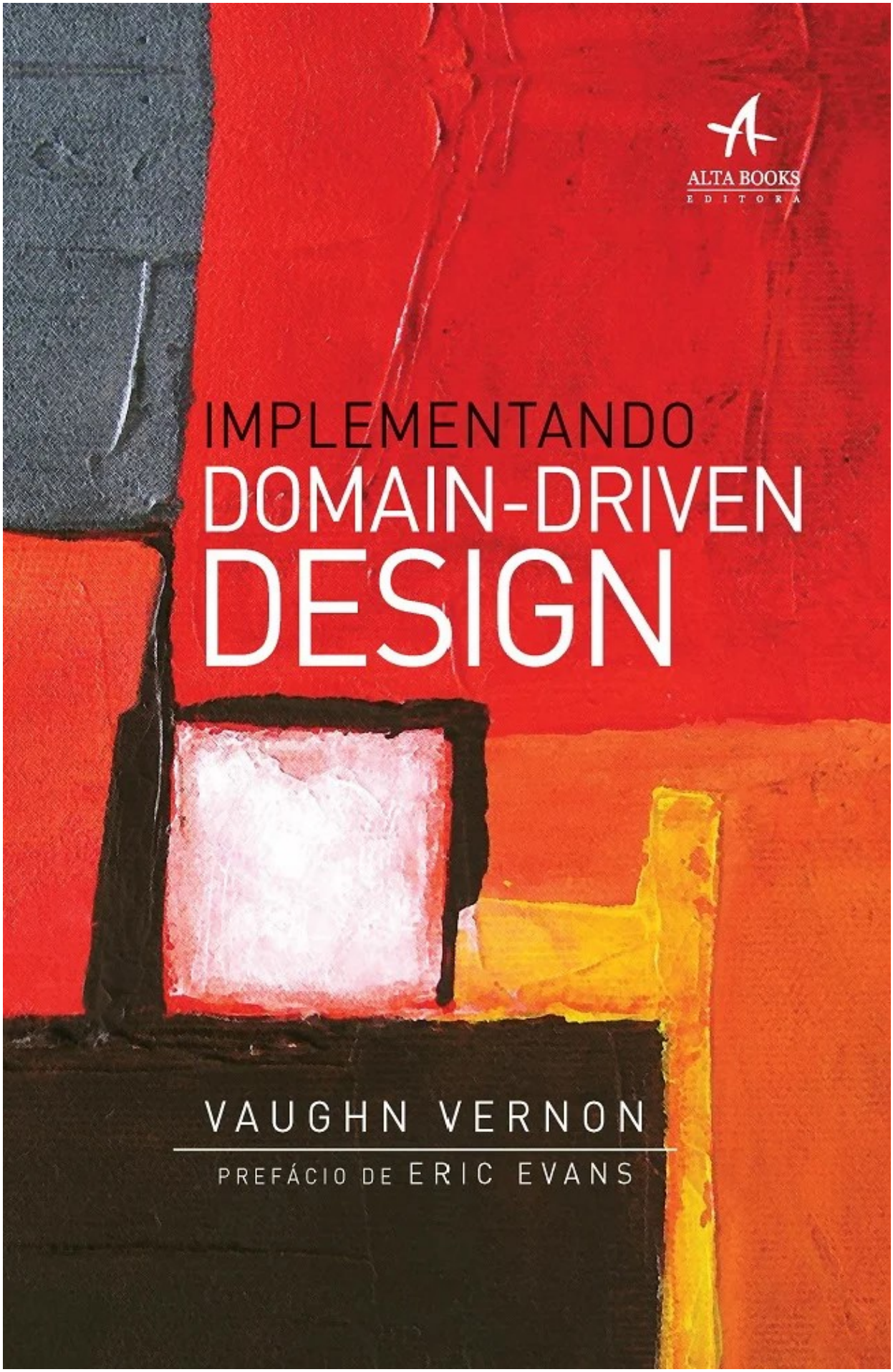


Padrões de Projeto, Princípios e Abordagens

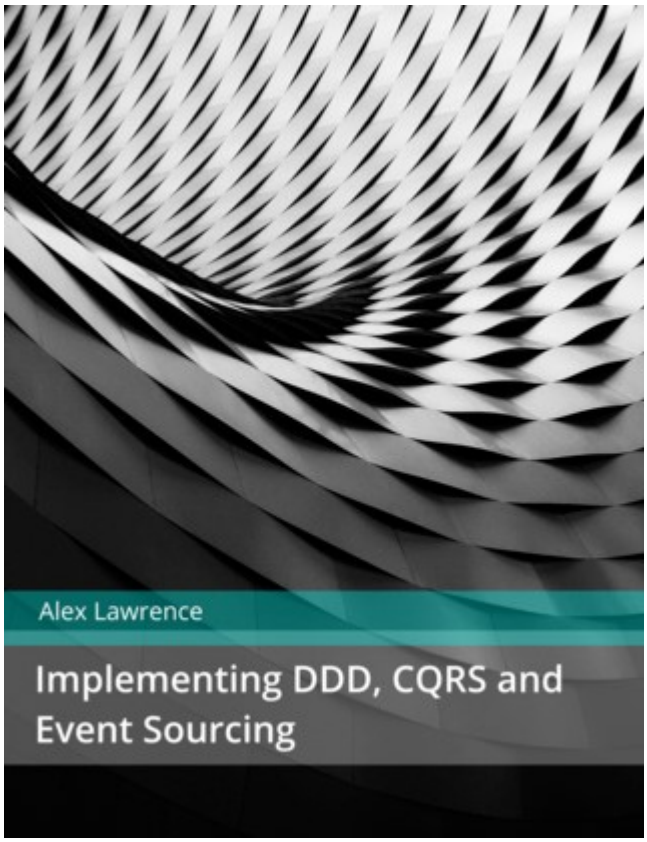
- DDD: A principal ideia do DDD é a de que o mais importante em um software não é o seu código,
  nem sua arquitetura, nem a tecnologia sobre a qual foi desenvolvido, mas sim o

problema que o mesmo se propõe
a resolver, ou em outras palavras, a regra de negócio. Ela é a razão do software existir, por isso deve receber
o máximo de tempo e atenção possíveis. Em praticamente todos os projetos de software,
a complexidade não está localizada nos aspectos técnicos, mas sim no negócio, na atividade que é exercida pelo
cliente ou problema que o mesmo possui. Como já diz o título do livro de Eric Evans, esse é o coração,
o ponto central de qualquer aplicação, portanto todo o resto deve ser trabalhado de forma que este coração
seja entendido e concebido da melhor forma possível.


3ª EDIÇÃO
REVISADA
Domain-Driven
DESIGN
Atacando as Complexidades no Coração do Software
Eric Evans
Introdução de Martin Fowler
ALTA BOOKS
EDITORA


ALTA BOOKS
EDITORA
IMPLEMENTANDO
DOMAIN-DRIVEN
DESIGN
VAUGHN VERNON
PREFÁCIO DE ERIC EVANS

- CQRS (Command Query Responsibility Segregation): Trata-se de um padrão arquitetural escalável, propondo a separação
  das responsabilidades em canais de comunicação distintos, descritos como modelo de Escrita (command) e leitura
  (query). Em sua essência, o CQRS visa segregar as atividades exercidas pelos componentes do software,
  entre Comandos e Consultas.



Estrutura da Aplicação

A aplicação está divida em camadas, em que cada componente tem sua responsabilidade separada.

Na **camada de negócios**, temos o projeto de dominio, responsável pelas regras de negócio individuais de cada entidade
e o projeto de aplicação responsável pelos casos de uso, e utilização das portas, através de comunicação com o
mundo externo.

Na **camada de infraestrutura**, estão os projetos que implementam as portas de comunicação com o mundo externo a aplicação.
Comunicação com banco de dados, sistema de arquivos, mensageria, envio de e-mails, sms, etc, ficam nesta camada.

Na **camada de serviços**, fica(m) o(s) serviço(s) própriamente dito(s), que tem a função de orquestrar e controlar todo o fluxo de requisições
a nossa aplicação.

Por fim a **camada de interfaces**, fornece aos diversos tipos de usuários, interfaces de comunicação com a aplicação.

Além dos projetos já citados, na pasta tests encontram-se os projetos para testes de unidade e integração.

Frameworks e Tecnologias

Frameworks

- `.NET 8`
- `ASP.NET Web API`

Frameworks de Teste

- `XUnit`

Bibliotecas

- `Fluent Validation`
- `MediatR`
- `Swagger`
- `Moq`

ORMS

- Dapper (SQL Server Legado)
- Entity Framework (Postgres)