Arquitetura Hexagonal

▼ **Significado**

Organiza o codigo de uma forma onde cada um tem sua responsabilidade, tendo como objetivo isolar a logica da aplicação do mundo externo;

▼ **Vantagens**

Toda logica separada, garante um menor acoplamento, facilita a testabilidade e facilita a trocar de componentes externos

Objetivo principal

- Isolar o nucleo da aplicação
- Ter um menor acoplamento possivel com as demais partes do sistemas (as regras de negocio as entidades)

Oque o Core da aplicação precisa ter ?

- O core precisa ter o que ele é e oque ele faz

Como esse núcleo se comunica com as demais partes do sistema

1. Se comunica atraves de ports que nada mais é do que abstrações (interfaces)
    - as interfaces sao usadas para encapsular toda regra de negocio da aplicação
    - assim a parte externa não sabe como o nucleo foi implementado assim como o nucleo também não precisa saber.
2. Junto com as ports também temos os adpters que são implementaçãos que vão permitir essa comunicação Existem 2 tipos de adpters
    - Quando o core é invocado pela parte Externa chamamos de inbound Adpters ex: navegador envia uma requisição http
    - Quando o core invoca a parte externa são chamados de outBounde Adpters ex: uma persistencia no banco de dados

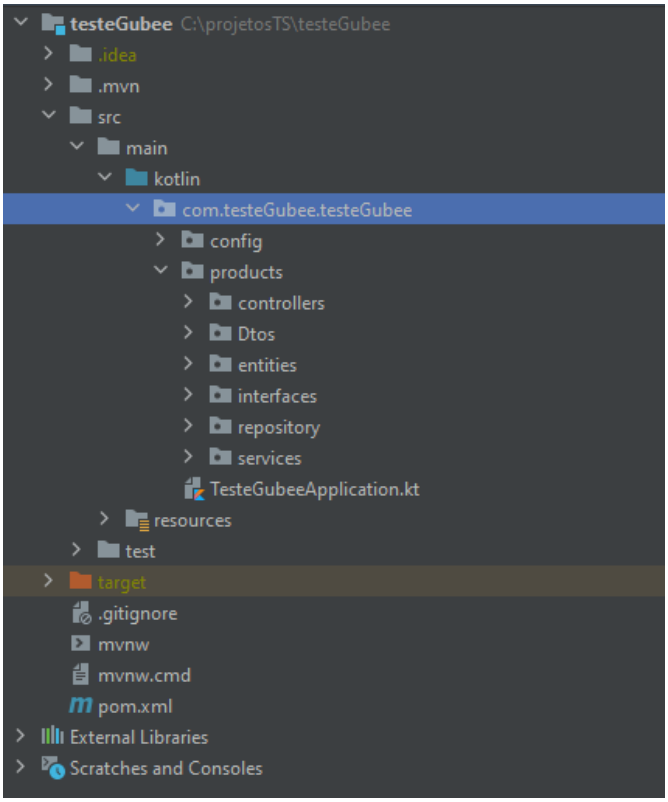
Implementação

▼ **Formato de pastas**

Padrão de um projeto dividido em Camadas

perceba que usando este padrão o nucleo da aplicação é fortemente acoplado com

com as demeais parte do codigo

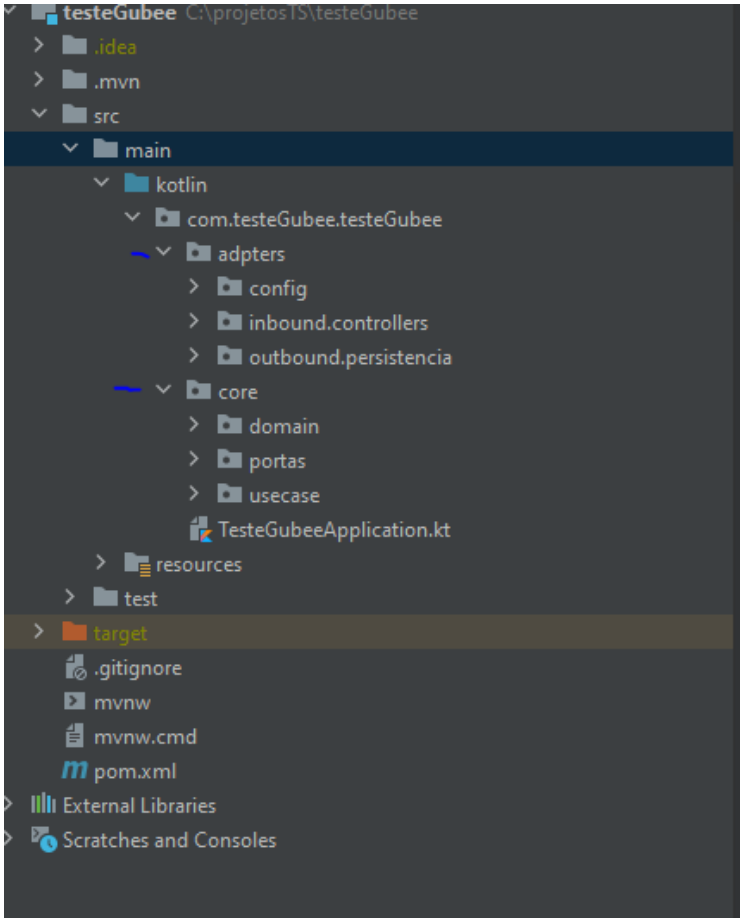


Padrão de um projeto com arquitetura Hexagonal

O núcleo da aplicação é desacoplado das dependencias da aplicação

onde o nucleo é o package (core) e as depencias estão no package (adapters)

onde esses dois package são ligados por abstração no java são as interfaces que estão no packge ports



▼ **implementação ports (abstração)**

Perceba que na implementação desse servico seguindo uma arquitetura padrão de camadas

a class FindByTargetMarket é dependente direto da class ProductRepository é não de abstrações

```
package com.testeGubee.testeGubee.products.services

import com.testeGubee.testeGubee.products.Dtos.ParametersFinds
import com.testeGubee.testeGubee.products.entities.Product
import com.testeGubee.testeGubee.products.interfaces.Finds
import com.testeGubee.testeGubee.products.repository.ProductRepository
import org.springframework.stereotype.Service

@Service
class FindByTargetMarket(val repository: ProductRepository):Finds {
    override fun find(parametersFinds: ParametersFinds): List<Product> {
        val listProducts:List<Product> = this.repository.findAll();
        return listProducts.filter { Contains.contains(parametersFinds.targetMarket, it.targetMarket)};
    }
}
```

Perceba agora que usando a arquitetura hexagonal usamos de abstrações (interfaces) Perceba abaixo que o servico FindByTargetMarket não depende mais de ProductRepository mas sim da abstração dele além

```
"""
interface IProductRepository {
    fun findAll(): List<ProductEntity>
    fun save(product: ProductEntity): ProductEntity;
}
"""


package com.testeGubee.testeGubee.core.usecase

import com.testeGubee.testeGubee.core.domain.Product
import com.testeGubee.testeGubee.core.portas.IFindByTargetMarket
import com.testeGubee.testeGubee.core.portas.IProductRepository
import com.testeGubee.testeGubee.adpters.inbound.controllers.dtos.ParametersFinds
import org.springframework.stereotype.Component

@Component
class FindByTargetMarket(private val servico: IProductRepository): IFindByTargetMarket {
    override fun find(parametersFinds: ParametersFinds): List<Product> {
        val listProducts:List<Product> = servico.findAll().map { it.toDomain() }
        return listProducts.filter { Contains.contains(parametersFinds.targetMarket, it.targetMarket)};
    }
}
```