E-Commerce

E-Commerces
Documento de Arquitetura de Software
Versão 1.0

Revision History

Date	Version	Description	Author
23/10/2020	1.0	Primeira versão da documentação	Joseph Weber
30/11/2020	1.1	Segunda versão da documentação	Joseph Weber

Sumário

Documento de Arquitetura de Software

1. Introdução

1.1 Objetivo

Esse documento de arquiteta de software tem como objetivo apresentar e documentar a arquiteta de um Sistema de E-commerces, o primeiro trabalho avaliativo de Projeto e Arquitetura de Software do curso de Engenharia de Software da PUCRS. Para esta solução, foi utilizado uma arquitetura cliente-servidor, conforme falaremos abaixo.

1.2 Escopo

O Sistema tem como objetivo centralizar em uma única plataforma todos as compras feitas por um cliente, realizando o cadastro de pedidos dentro desta aplicação, facilitando assim o acompanhamento de suas compras online.

1.3 Definições, Acrônimos e Abreviações

E-Commerce: Sistema de comercio online.

API: Application Programming Interface

CRUD: Create, Read, Update e Delete

PUCRS: Pontifícia Universidade Católica do Rio Grande do Sul

1.4 Visão Geral

Este documento apresenta toda a descrição detalhada do projeto arquitetônico realizado pelos autores para a solução do Sistema, mostrando os diagramas realizados para representar a arquitetura, codificação, base dados, interação do Sistema e requisitos.

2. Visão de Caso de Uso e Requisitos

Neste projeto utilizamos os casos de uso para representar o usuário e as ações possíveis dentro do Sistema em dois modos: logado e não logado. Também existe uma representativade do Administrador.

Referência	Descrição
US01	O usuário precisa logar para ter acesso as funcionalidades do sistema
US02	O usuário deve estar autenticado através de um token
US03	O usuário deve poder se cadastrar no sistema
US04	O banco de dados deve ser H2
US05	O administrador deve ter permissão para cadastrar um e-commerce
US06	O sistemas deve simular importação de pedidos para usuário

US07	Deve ser possível consultar todos os e-commerces que o sistema da suporte
US08	Deve ser possível filtrar os pedidos do cliente por e-commerce e opcionalmente, pelo status do pedido
US09	Deve ser possível consultar detalhadamente as informações do produtos comprados do pedido
US10	O sistema deve consultar os pedidos das compras realizadas, por ecommerce, por data, por agilidade e por cumprimento de entrega
US11	O sistema deve se baseado em um arquitetura cliente-servidor
US12	O frontend da aplicação deve ser em React Native
US13	O backend da aplicação deve ser em Spring

3. Requisitos e Restrições arquiteturais

Nesta sessão serão apresentados os requisitos de software utilizado na arquitetura:

Requisito	Solução
Linguagem	Spring boot 2 (API Backend), React Native (Frontend)
Plataforma	Mobile, IOS ou Android
Segurança	Spring Secruity e JWT Token
Persistência	H2 (Banco de dados embutido)

Requisitos Funcionais:

- Cadastrar um e-commerce;
- Cadastrar os pedidos dos clientes (deverá ser “simulado” um mecanismo de importação de pedidos em aberto do cliente a partir de um e-commerce pré-cadastrado);
- Consultar pedidos do cliente, por e-commerce (estes pedidos deveriam ser atualizados, “buscando” de cada ecommerce a informação do status do seu pedido – simular este comportamento.)
- Gerar relatórios de compras do cliente, por e-commerce, por data, por agilidade de entrega, por cumprimento de prazo de entrega.

Requisitos Não-Funcionais:

- Autenticação com criptografia;

4. Visão Lógica

A Arquitetura cliente-servidor é uma abordagem que separa os processos em módulos independentes que interagem entre si, o cliente e o servidor, permitindo o compartilhamentos e troca de mensagens afim de ter um maior benefício de cada parte.

O servidor fica responsável por realizar disponibilizar recursos e informações necessárias, faz o processamento das solicitações realizadas por máquinas clientes e responde conforme suas necessidades.

O Cliente é o responsável por iniciar e terminar a conversação com os servidores, realizando solicitações de entrada e obtendo como retorno as saídas.

Segue abaixo imagem de demonstração:

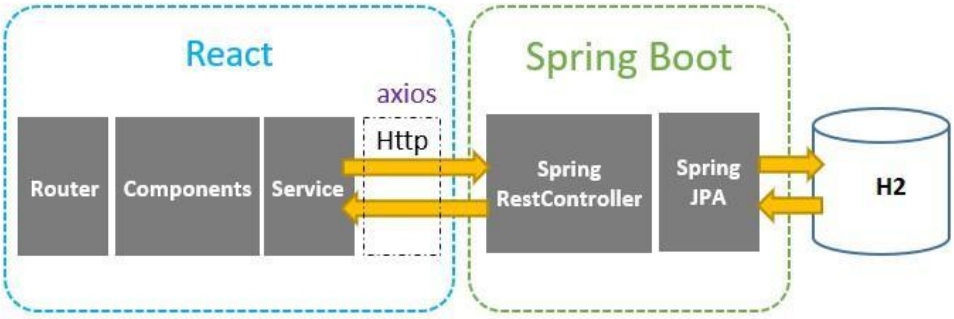


Figura 1 Arquitetura Cliente-servidor

4.1 Visão Geral

No sistema e-commerces o servidor e o cliente possuem os seguintes papéis:

Servidor:

- Processa solicitações de cadastro de usuários e as responde se foi realizado;
- Processa solicitação de cadastro de e-commerce e as responde se foi realizado;
- Processa solicitação de cadastro de pedidos e as responde se foi realizado;
- Processa solicitação de consulta de e-commerce e responde com as consultas realizadas;
- Processa solicitação de filtros e responde com os filtros realizados;

Cliente:

- Envia solicitação de cadastramento de usuário e mostra o retorno do servidor;
- Envia solicitação de cadastro de e-commerce e mostra o retorno do servidor;
- Envia solicitação de cadastro de pedidos e mostra o retorno do servidor;
- Envia solicitação de consultas e mostra o retorno do servidor;
- Envia solicitação de filtro e mostra o retorno do servidor;

4.2 Pacotes de projeto com significado arquitetônico

Nesta sessão será apresentada brevemente como foi modelada os pacotes do projeto:

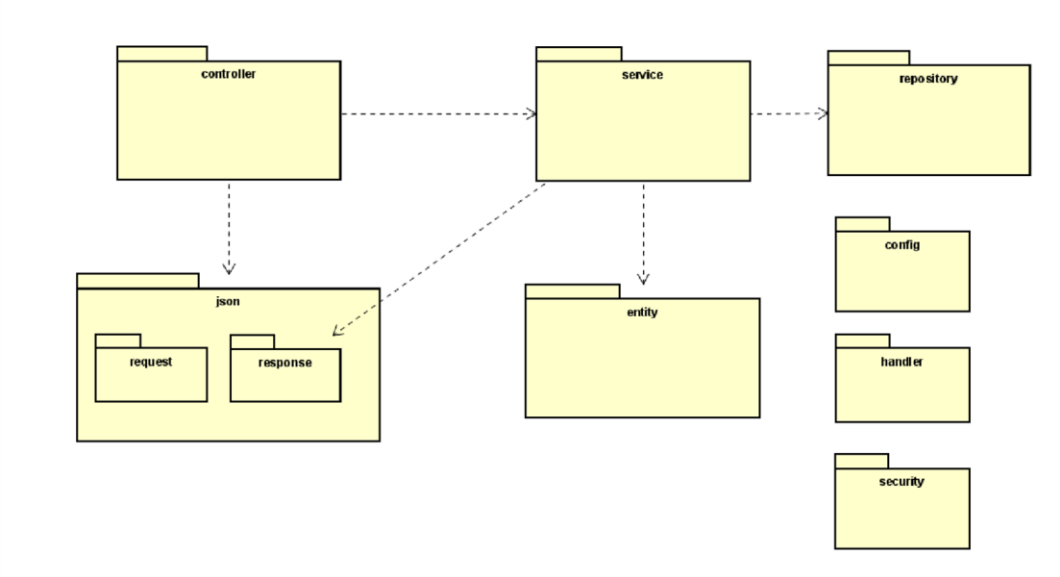


Figura 2 Diagrama de Pacotes

4.3 Realizações de Caso de Uso

Nesta sessão será apresentada o diagrama de casos de uso, representando a interação do usuário com a aplicação, mostrando as interações de caso de uso que o usuário é envolvido durante o fluxo da aplicação:

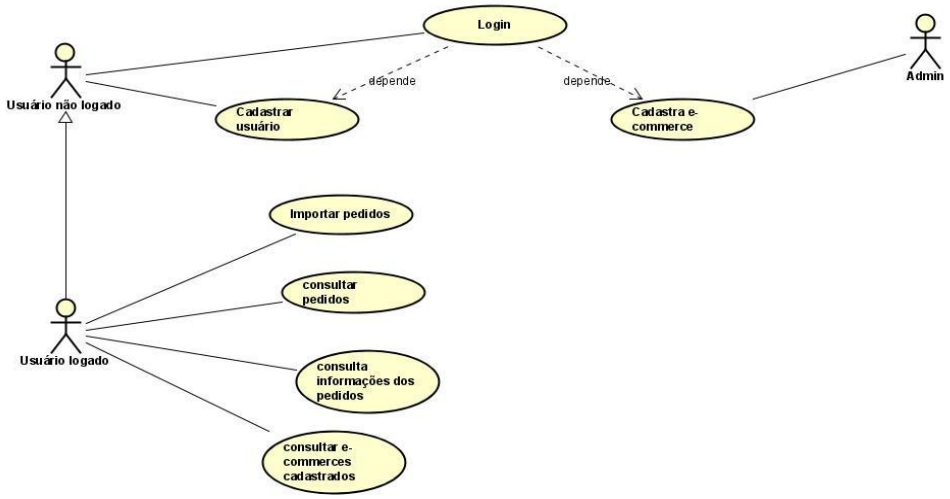


Figura 3 Diagrama de Casos de Uso

4.4 Design Patterns

Design Patterns são metodologias utilizadas no contexto de desenvolvimento de software para solucionar problemas generalistas recorrentes. Não se trata de nenhum framework ou qualquer tipo de biblioteca e sim de uma definição em alto nível de como um problema pode ser solucionado.

Ele é utilizado em muitos projetos atuais, pois gera um ganho de produtividade e um alto desempenho em sistemas, contribui para a organização e manutenção de códigos e facilita o engajamento do time, pois são simples de ser compreendidos.

Existem vários Design Patterns, sendo o mais conhecido o GoF (Gang of Four), que desenvolveram alguns padrões que se adaptam bem as linguagens atuais.

Neste trabalho, foram utilizados três padrões do GoF, são eles: Singleton, Factory e Strategy.

- **Singleton:** O padrão Singleton trata-se de uma forma de compartilhamento de dados dentro de um projeto. Normalmente ele é utilizado na camada de apresentação, mas pode ser modelado para camadas de negócio dependendo do contexto e de suas necessidades.

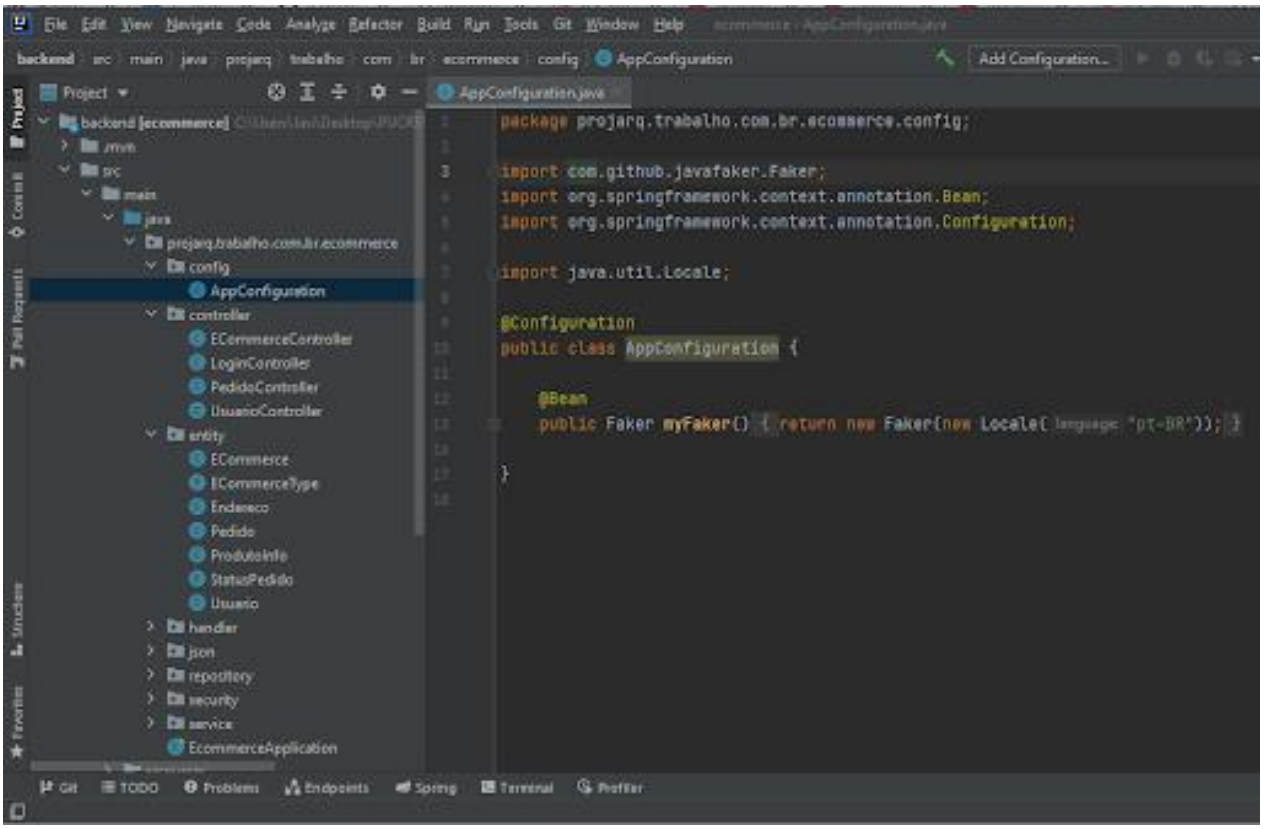
  Esse padrão sugere que seja criada uma única instancia de uma classe afim de compartilhar todas as informações contidas nela para as demais classes, sendo assim, não perderá a referência de seus dados dentro da execução.

  Isso é necessário por vários motivos, mas principalmente em programação orientada a objetos onde existem classes que podem instanciada várias vezes, tendo em sua referência os dados locais de cada objeto, com informações diferentes e apagadas quando os objetos são descartados ou destruídos.

  Neste contexto, ocorre inúmeras vezes de criar-se um Singleton, inicializando a instancia deste objeto único dentro do projeto e compartilhando informações para o demais, sendo estes podendo excluir, visualizar, criar e editar dentro deste objeto, dependendo das permissões e necessidades.

  Utilizamos este padrão dentro do nosso projeto da seguinte forma:

  Criamos temos um serviço externo denominado com o @Bean chamado de Faker. Esse serviço é possui a simulação de produtos do tipo livro que utilizamos no projeto, ele possui uma única instância desse serviço através do @Bean, um jeito um pouco diferente realizado pelo springboot para conseguirmos obter um singleton dentro do projeto no backend.

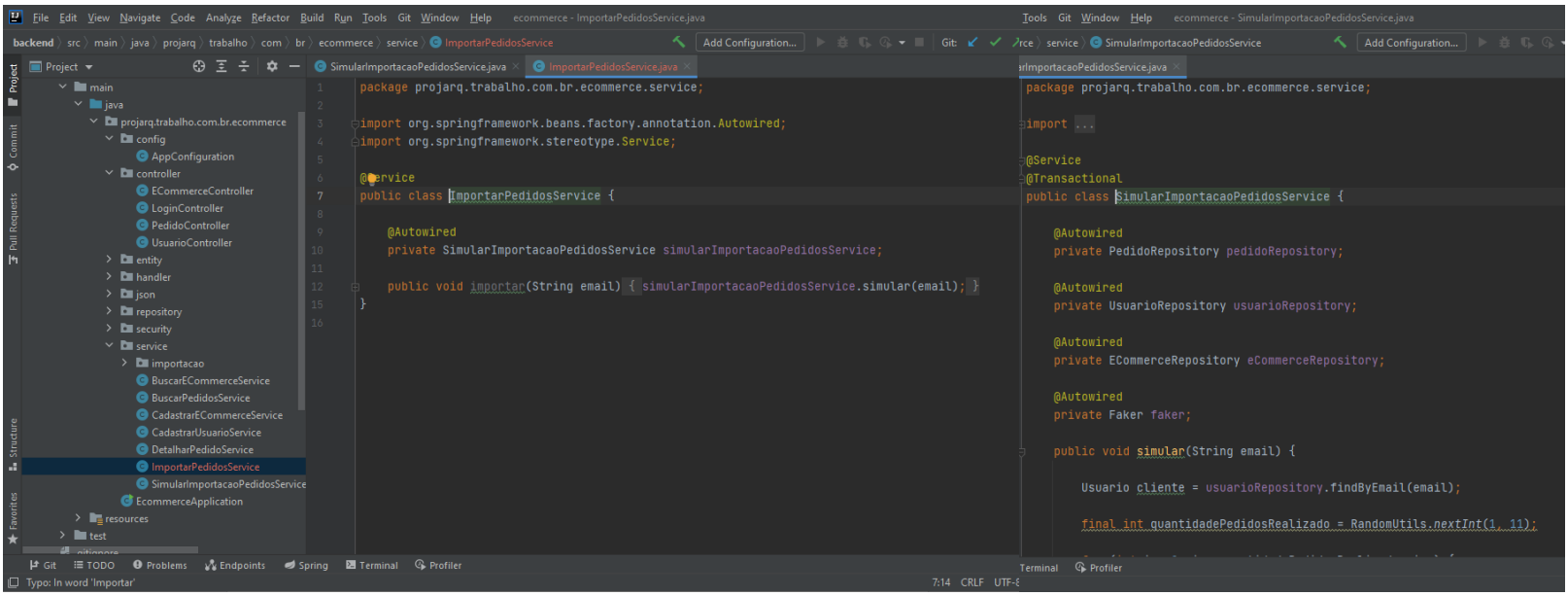


- **Factory:** O Padrão Factory trata-se de uma forma de organizar a responsabilidade de classes dentro do projeto. Ele permite que uma classe seja criada para padronizar uma assinatura de uma estrutura de classes, e as classes derivadas dela, utilizam essa assinatura a seu favor e conforme seu contexto.

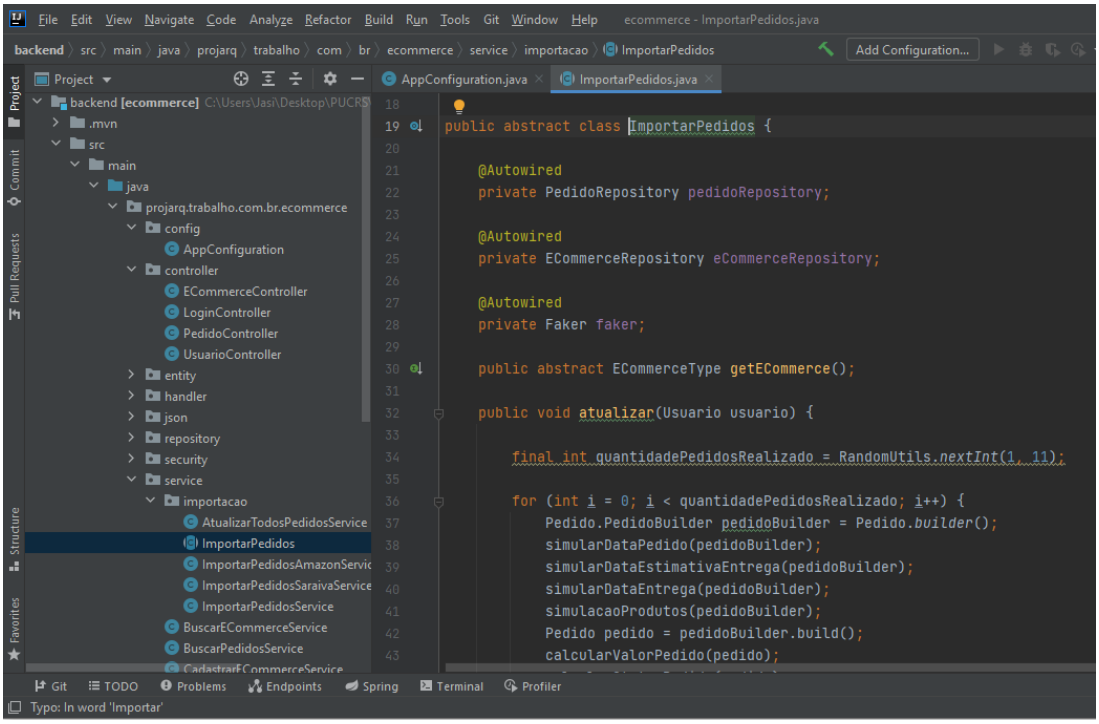
  Esse padrão sugere que seja criada uma classe para que classes que possuem métodos e atributos comuns possam estendê-la e implementar esses métodos e atributos como quiser, ou seja, basicamente ela delega as suas classes estendidas o que deve ser feito. Isso facilita muito na organização e padronização do código, na divisão de contexto, responsabilidades de cada classe.

  No contexto do nosso projeto o padrão Factory foi utilizado da seguinte forma:

  Criamos uma classe chamada, ImportarPedidosService que basicamente importa o pedido com base no tipo do prodido, realizando um filtro pelo tipo do ecommerce e criando apenas a instanciação deste pedido, assim garantimos a responsabilidade da importação para a classe adjacente. Isso explora a divisão de responsabilidade e performance da aplicação, uma vez que apenas a importação necessária está sendo instanciada e sendo delegada a manipulação para sua classe filha.



- **Strategy:** O Padrão Strategy trata-se de uma forma centralizar todos os tipos de comportamentos de um contexto em uma única classe responsável por gerar a instância do comportamento esperado pelo usuário e após este realizar sua tarefa, padronizando, organizando e otimizando todo o contexto, permitindo assim que seja instancia somente a classe do comportamento desejada e não instanciar todas as classes e esperar para que o usuário interaja com elas.
  Esse padrão sugere que você pegue classes que realizam algo específico em diversas maneiras diferentes e centraliza todos esses algoritmos para uma única classe chamada Strategy. Essa será responsável por delegar a tarefa a ser feita instanciando a classe da tarefa deseja, ao invés de executa-la. Isso gera ponto de acesso centralizado de todas as classes que realizam essa tarefa de maneiras diferentes e não cria instancias que não serão usadas.
  No contexto do nosso projeto o padrão Strategy foi utilizado da seguinte forma:
  Criamos uma classe chamada “ImportarPedidos”, que centraliza todas as importações de pedidos por ecommerce, criando uma única instancia do tipo de importação. Ela é uma classe abstrada, no qual delega as funções de simulação de compa para cada tipo de ecommerce apresentado.



5. Visão do Processo

A arquitetura é baseada em padrão API-REST, ou seja, um conjunto de definições a serem utilizadas para a criação de web services.

Utilizando Como base o a arquitetura de Cliente-Servidor e Rest-API, criamos uma API em backend utilizando Springboot 2, que permite a comunicação com uma camada de persistência h2, (banco de dados embutido no springboot) e realiza o response no formato JSON para ser consumido no frontend codificado em React Native.

O Cliente (frontend), também envia request no formato JSON para ser processados pelo Servidor (backend) e assim é realizado a comunicação do cliente com o server.

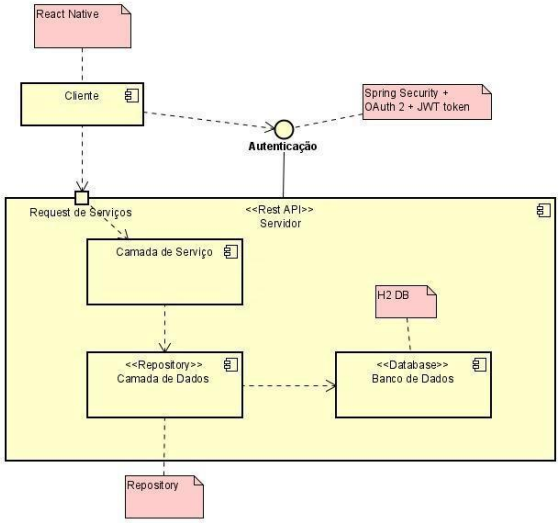


Figura 4 Diagrama de Componentes

6. Visão de Implantação

Nesta sessão será apresentada como foi pensada a modelagem da Implantação do sistema, conforme imagem abaixo:

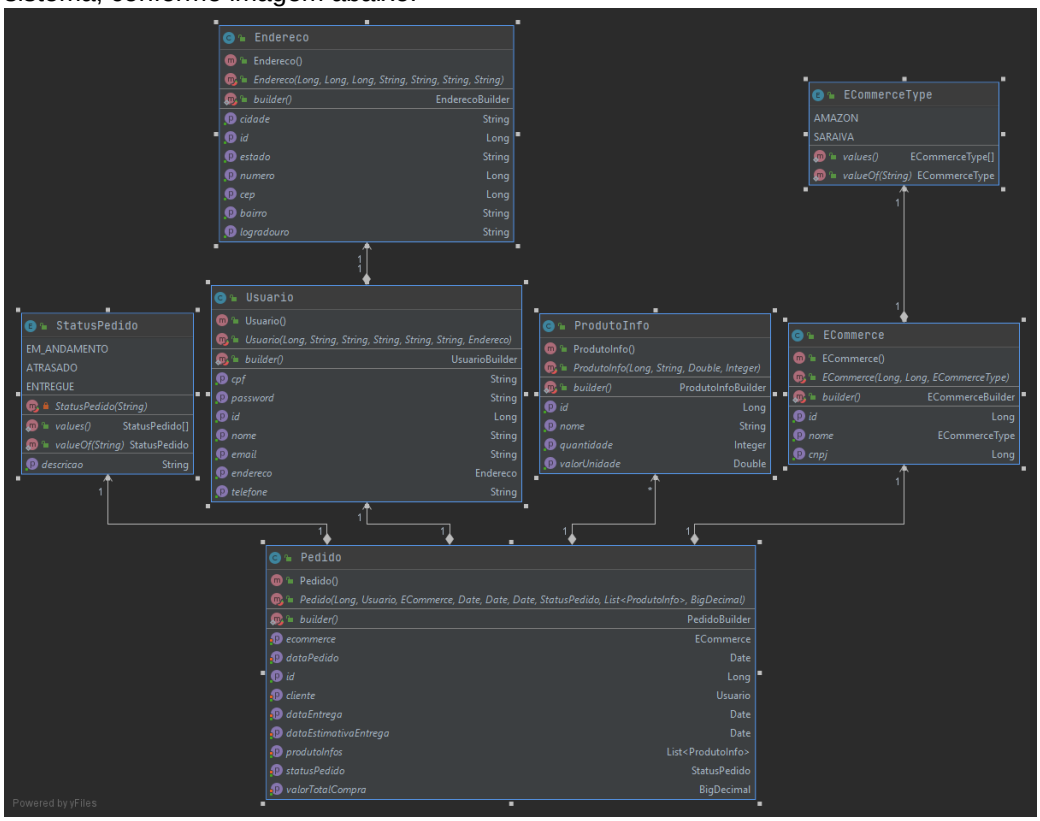


Figura 5 Diagrama de Classes

Classe Pedido:

- Responsável por pelas informações do pedido, possuindo o e-commerce que foi realizado a compra, as informações do produto que foi comprado e o status do pedido e suas principais funções;

Classe ProdutoInfo:

- Armazena todas as informações do Produto, como a quantidade, valor e o nome e suas principais funções; **Classe E-commerce:**
- Armazena todas as informações do e-commerce, como nome e cnpj e suas principais funções;

Classe Cliente:

- Armazena as informações dos clientes, como nome, cpf, endereço e também suas principais funções

Classe Endereço:

- Representa o endereço do Sistema.

Projeto foi dividido em Front-end e Back-End

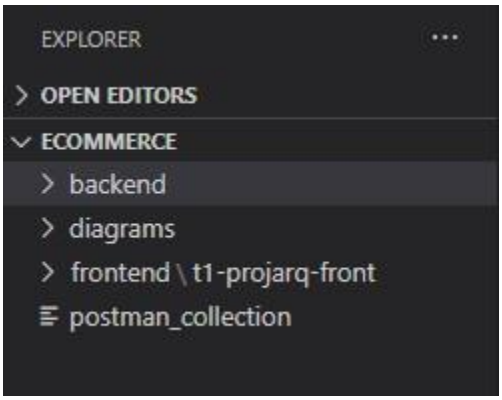
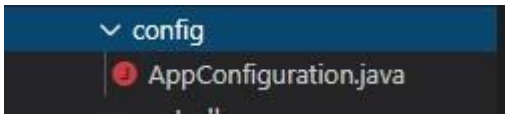


Figura 6 VSCode - Backend Frontend

No Back-End possuímos as seguintes divisões:

Config:

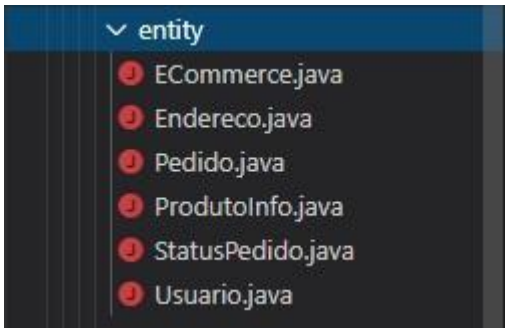


- Classes de configurações iniciais do sistema, como localidade;

Controller:

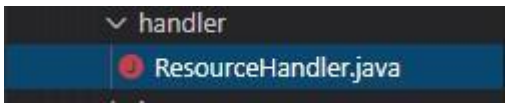
- Possui as classes que permitem as requisições da API

Entity:



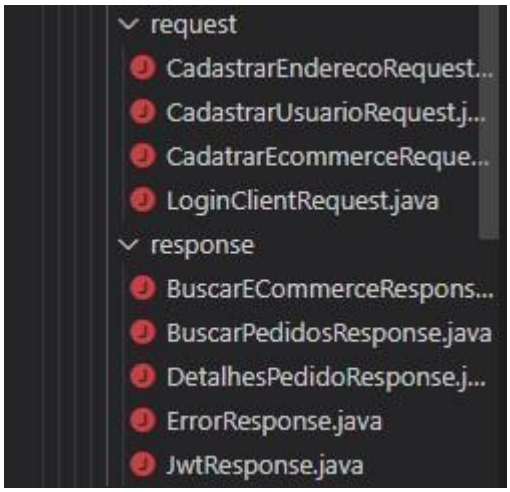
- Armazenas as informações dos objetos principais;

Handler:



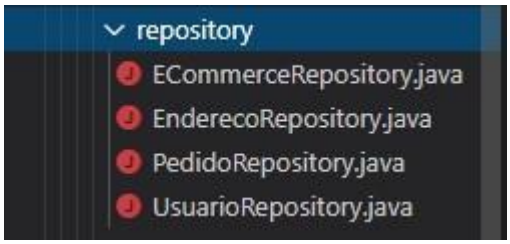
- Possui a classe de manipulação.

Json:



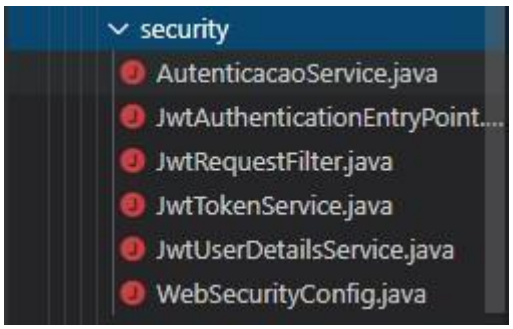
- Possui as classes para request e reponse da aplicação:

Repository:



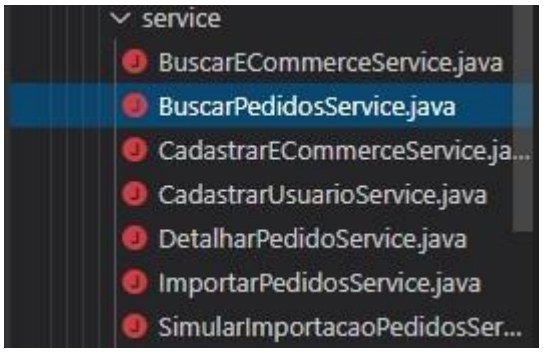
- Possui as classes para os CRUD do sistema.

Security:



- Possui as classes de autenticação (segurança) do usuário;

Service:



- Manter os serviços do sistema;

Bando de Dados H2:

H2 é um banco de dados embutido, portando o mecanismo de acesso ao banco de dados roda junto com a aplicação, ao em invés de ter uma outra aplicação que deva se comunicar para acessar um banco de dados.

Script do banco gerado pelo H2 dentro do contexto hibernate na aplicação Spring

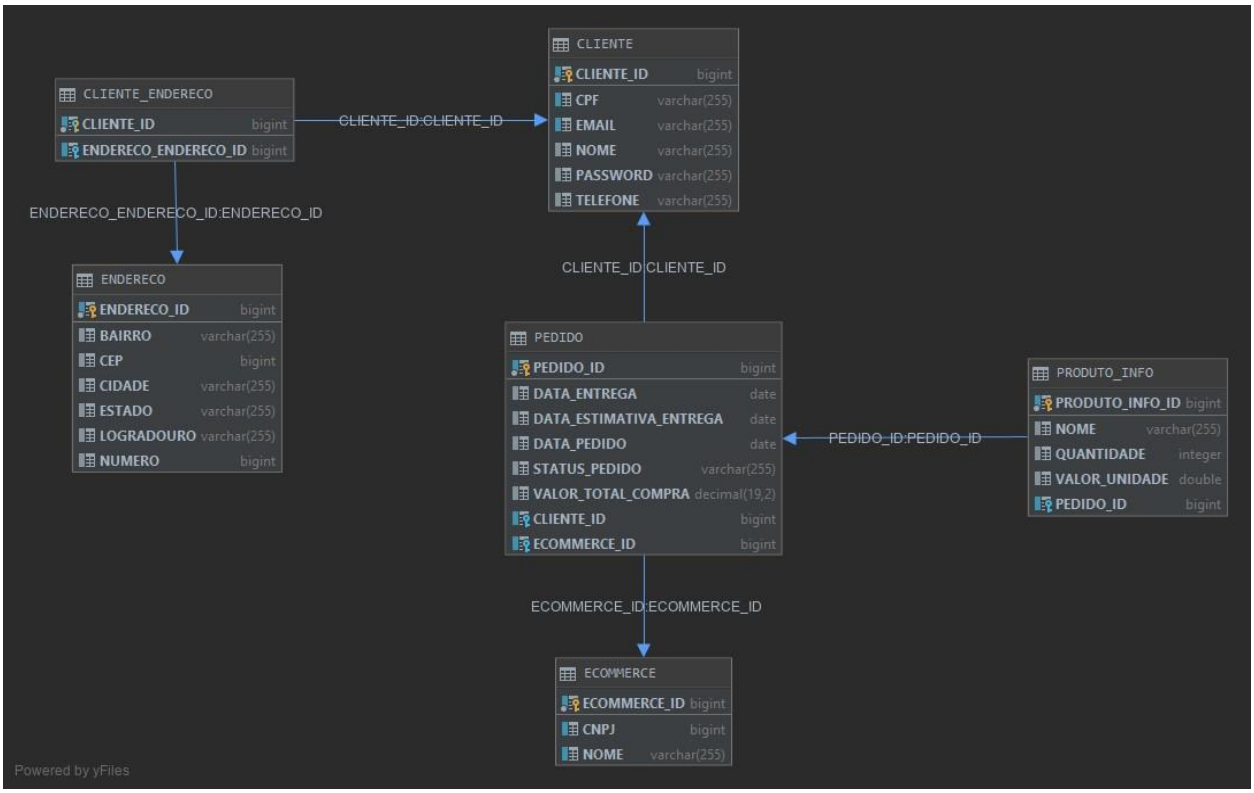


Figura 7 Banco de Dados

7. Qualidade

Nesta sessão apresentaremos alguns atributos de qualidade:

Item	Descrição	Solução
Segurança	Sistema possui autenticação de usuário criptografada.	Spring Security e JWT Token
Interoperabilidade	Sistema possui comunicação entre Front e Back-end	Padrão Rest-API, Spring boot 2, React Native e Axios