# Serviço de Pedidos de Crédito

Esse serviço é uma aplicação web que permite gerenciar e persistir solicitações de crédito dos clientes, também é responsável por armazenar os log/eventos de alterações no pedido. O serviço utiliza o framework Spring Boot para desenvolver uma API REST que expõe os recursos e as operações relacionados aos pedidos de crédito. O serviço também utiliza o banco de dados NoSQL DynamoDB da AWS para armazenar e manipular os dados dos pedidos de crédito modelado sobre os pricípos de uma SingleTableModeling. Além disso, o serviço utiliza o mecanismo de busca distribuído ElasticSearch para indexar e consultar os dados dos pedidos de crédito de forma eficiente e flexível.

## Rotas

GET /pedidos?codigoCliente={}&nomeProduto={}&valorMinimo={}&valorMaximo={}&dataInicio={}&dataFim={}&status={}&sortBy={}&pageSize={}&pageNumber={}

Essa consulta permite consultar os pedidos de crédito de acordo com os critérios especificados nos parâmetros da URL. Os parâmetros são opcionais e podem ser combinados de diferentes formas. Os parâmetros são: - codigoCliente: o código do cliente que fez o pedido de crédito. - nomeProduto: o nome do produto solicitado pelo cliente. - valorMinimo: o valor mínimo do pedido de crédito. - valorMaximo: o valor máximo do pedido de crédito. - dataInicio: a data de início do período de consulta dos pedidos de crédito. - dataFim: a data de fim do período de consulta dos pedidos de crédito. - status: o status do pedido de crédito, que pode ser NOVO, EM\_ANALISE, APROVADO, REPROVADO ou CANCELADO, etc. - sortBy: o campo pelo qual os pedidos de crédito serão ordenados, que pode ser codigoPedidoCredito, codigoCliente, nomeProduto, valorPedido, dataPedido ou statusPedido. - pageSize: o número de pedidos de crédito que serão retornados por página. - pageNumber: o número da página que será retornada.

Retorna uma lista de pedidos de crédito que atendem aos critérios da consulta, no formato JSON, e retorna o código de status 200 (OK) se a consulta for bem-sucedida, ou o código de status 400 (Bad Request) se algum parâmetro for inválido ou faltar.

GET /pedidos/{CodigoPedidoCredito}

Essa Consulta permite consultar um pedido de crédito específico, de acordo com o código do pedido de crédito informado na URL. O parâmetro é obrigatório e deve ser um UUID válido. a consulta retorna o código de status 200 (OK) se a consulta for bem-sucedida, ou o código de status 404 (Not Found) se o pedido de crédito não existir.  
Exemplo de retorno:

{  
 "pedido": {  
 "codigoCliente": "23345677906912",  
 "agencia": <agencia>,  
 "conta": <conta>,  
 "regiao": "01",  
 "dicom": <dicom>,  
 "statusPedido": "NOVO",  
 "segmentoBancario": "K",  
 "codigoCanalSolicitacao": "01",  
 "descricaoCanalSolicitacao": null,  
 "valorPedido": <valorPedido>,  
 "unidadeMonetaria": "BRL",  
 "nomeGrupo": "<Nome Empresa>",  
 "dataPedido": "2020-01-01T11:00:00.00000",  
 "prazo": 360,  
 "unidadePrazo": null,  
 "codigoIdentificacaoOrigem": null,  
 "parecerOrigemPedido": {  
 "parecer": "S",  
 "justificativa": "estou de acordo com aprovação desse credito",  
 "dataParecer": null,  
 "usuarioParecer": "999999999"  
 },  
 "codigoProduto": "1234",  
 "descricaoProduto": "descricao\_produto",  
 "qualifier": "#PEDIDO",  
 "codigoPedidoCredito": "ccc9c52a-34cf-49da-ab31-bacef8f99234"  
 } ,  
 "eventos": [  
 {  
 "codigoEvento": "620097f1-c4c4-4866-b7d0-6a7ecb54c39c",  
 "dataHoraEvento": "2023-11-05T17:01:47.961+00:00",  
 "origemEvento": "Sistema Solicitar Crédito",  
 "dadosEventoAlteracao": null,  
 "detalhes": null,  
 "tipoEvento": "NOVO",  
 "qualifier": "#EVENTO#2023-11-05T15:01:47.954-02:00",  
 "codigoPedidoCredito": "0ac8b6c7-a570-4720-84b5-11cb81b8f46f"  
 }  
 ]  
}

DELETE /pedidos/{CodigoPedidoCredito}

Essa rota permite deletar um pedido de crédito específico, de acordo com o código do pedido de crédito informado na URL. O parâmetro é obrigatório e deve ser um UUID válido. Essa rota não retorna nenhum conteúdo, apenas o código de status 204 (No Content) se o pedido de crédito for deletado com sucesso, ou o código de status 404 (Not Found) se o pedido de crédito não existir.

PUT /pedidos/{CodigoPedidoCredito}

Essa rota permite atualizar um pedido de crédito específico, de acordo com o código do pedido de crédito informado na URL. O parâmetro é obrigatório e deve ser um UUID válido.

POST /pedidos

Essa rota permite criar um novo pedido de crédito, com os dados informados no corpo da requisição. O corpo da requisição deve conter um objeto JSON com os seguintes campos obrigatórios: exemplo de requisição:

{  
 "codigo\_cliente": "99945677906913",  
 "valor\_pedido": 55000000.00,  
 "codigo\_produto":"1234",  
 "segmento":"K",  
 "produto":"1234",  
 "path":"ltc",  
 "nome\_grupo": "Grupo Carrefur",  
 "descricao\_produto":"descricao\_produto",  
 "data\_pedido": "2020-01-01T11:00:00.00000",  
 "prazo": 360,  
 "parecer\_origem\_pedido": {  
 "funcional\_colaborador": "999999999",  
 "texto\_parecer": "estou de acordo com aprovação desse credito",  
 "indicador\_parecer": "S"  
 }  
}

## Arquitetura Hexagonal

O serviço segue a arquitetura hexagonal, que é um padrão arquitetural que visa isolar a lógica de domínio das dependências externas. A lógica de domínio é especificada em um núcleo de negócio, que chamamos de parte interna, com o resto sendo partes externas. O acesso à lógica de domínio a partir do exterior é feito através de portas e adaptadores. Esses componentes são modulares e intercambiáveis, o que aumenta a consistência do processamento e facilita a automação de testes.

O serviço é dividido em três camadas: aplicação (externa), domínio (interna) e infraestrutura (externa):

* Através da camada de aplicação, o usuário ou qualquer outro programa interage com a aplicação. Essa área deve conter coisas como interfaces de usuário, controladores RESTful e bibliotecas de serialização JSON. Ela inclui tudo que expõe a entrada para a nossa aplicação e orquestra a execução da lógica de domínio.
* Na camada de domínio, mantemos o código que toca e implementa a lógica de negócio. Esse é o núcleo da nossa aplicação. Essa camada deve ser isolada tanto da parte de aplicação quanto da parte de infraestrutura. Além disso, ela deve conter interfaces que definem a API para se comunicar com partes externas, como o banco de dados, com as quais o domínio interage.
* Finalmente, a camada de infraestrutura é a parte que contém tudo que a aplicação precisa para funcionar, como configuração do banco de dados ou configuração do Spring. Ela também implementa as interfaces dependentes da infraestrutura da camada de domínio.

## Estrutura do Projeto

O projeto é composto por quatro módulos: application, domain, infrastructure e ms-launcher. Cada módulo tem sua própria estrutura de arquivos e classes, conforme descrito abaixo:

### application

Esse módulo contém as classes relacionadas à camada de aplicação, como controladores, serviços e DTOs. A estrutura de arquivos desse módulo é:

application  
├── src  
│ └── main  
│ └── java  
│ └── com  
│ └── example   
│ └── application  
│ ├── controller  
│ │ └── PedidoCreditoController.java  
│ ├── dto  
│ │ ├── PedidoCreditoDTO.java  
│ │ └── PedidoCreditoMapper.java  
│ └── service  
│ └── PedidoCreditoService.java  
├── build.gradle  
└── settings.gradle

As principais classes desse módulo são:

* PedidoCreditoController: define os endpoints da API REST usando as anotações do Spring Boot. Essa classe tem métodos para criar, obter, atualizar e deletar um pedido de crédito no DynamoDB. Essa classe também tem um método para fazer uma consulta avançada no ElasticSearch usando vários critérios de filtragem e ordenação.
* PedidoCreditoDTO: representa um pedido de crédito com os seus atributos em formato DTO (Data Transfer Object). Essa classe é usada para transferir os dados dos pedidos entre as camadas de aplicação e infraestrutura.
* PedidoCreditoMapper: faz o mapeamento entre as classes PedidoCredito e PedidoCreditoDTO, usando a biblioteca MapStruct.
* PedidoCreditoService: implementa a lógica de negócio relacionada aos pedidos de crédito, usando as interfaces da camada de domínio e as classes da camada de infraestrutura.

### domain

Esse módulo contém as classes relacionadas à camada de domínio, como entidades, repositórios e portas. A estrutura de arquivos desse módulo é:

├── src   
│ └── main   
│ └── java   
│ └── com   
│ └── example  
│ └── domain  
│ ├── entity  
│ │ ├── Pedido.java  
│ │ └── PedidoItem.java  
│ ├── port  
│ │ ├── in  
│ │ │ ├── CreatePedidoUseCase.java  
│ │ │ ├── DeletePedidoUseCase.java  
│ │ │ ├── GetPedidoUseCase.java  
│ │ │ ├── ListPedidosUseCase.java  
│ │ │ └── UpdatePedidoUseCase.java  
│ │ └── out  
│ │ ├── LoadPedidoPort.java  
│ │ ├── PersistPedidoPort.java  
│ │ └── SearchPedidosPort.java  
│ └── repository  
│ └── PedidoRepository.java  
├── build.gradle  
└── settings.gradle

As principais classes desse módulo são:

* Pedido: representa um pedido de crédito com os seus atributos e métodos. Essa classe é a raiz do agregado do pedido, que contém uma lista de itens do pedido.
* PedidoItem: representa um item do pedido com os seus atributos e métodos. Essa classe é uma entidade filha do agregado do pedido.
* CreatePedidoUseCase: define a interface para o caso de uso de criar um novo pedido de crédito. Essa interface é implementada pela camada de aplicação e usa a porta PersistPedidoPort para persistir o pedido na camada de infraestrutura.
* DeletePedidoUseCase: define a interface para o caso de uso de deletar um pedido de crédito pelo seu código. Essa interface é implementada pela camada de aplicação e usa a porta LoadPedidoPort para carregar o pedido da camada de infraestrutura e a porta PersistPedidoPort para deletar o pedido na camada de infraestrutura.
* GetPedidoUseCase: define a interface para o caso de uso de obter um pedido de crédito pelo seu código. Essa interface é implementada pela camada de aplicação e usa a porta LoadPedidoPort para carregar o pedido da camada de infraestrutura.
* ListPedidosUseCase: define a interface para o caso de uso de listar todos os pedidos de crédito. Essa interface é implementada pela camada de aplicação e usa a porta LoadPedidoPort para carregar os pedidos da camada de infraestrutura.
* UpdatePedidoUseCase: define a interface para o caso de uso de atualizar um pedido de crédito pelo seu código. Essa interface é implementada pela camada de aplicação e usa a porta LoadPedidoPort para carregar o pedido da camada de infraestrutura e a porta PersistPedidoPort para persistir o pedido na camada de infraestrutura.
* LoadPedidoPort: define a interface para carregar um ou mais pedidos da fonte de dados. Essa interface é definida pela camada de domínio e implementada pela camada de infraestrutura, usando o repositório do DynamoDB ou o cliente do ElasticSearch.
* PersistPedidoPort: define a interface para persistir ou deletar um pedido na fonte de dados. Essa interface é definida pela camada de domínio e implementada pela camada de infraestrutura, usando o repositório do DynamoDB ou o cliente do ElasticSearch.
* SearchPedidosPort: define a interface para fazer uma consulta avançada nos pedidos usando vários critérios de filtragem e ordenação. Essa interface é definida pela camada de domínio e implementada pela camada de infraestrutura, usando o cliente do ElasticSearch.
* PedidoRepository: define a interface para o repositório do pedido, que estende a interface CrudRepository do Spring Data. Essa interface é usada pela camada de infraestrutura para interagir com o banco de dados DynamoDB.

### infrastructure

Esse módulo contém as classes relacionadas à camada de infraestrutura, como configurações, implementações das portas e adaptadores. A estrutura de arquivos desse módulo é:

infrastructure  
├── src  
 └── main  
 ├── java  
 │ └── com  
 │ └── example  
 │ └── mslibrary  
 │ └── infrastructure  
 │ ├── adapter  
 │ │ ├── DynamoDBPedidoAdapter.java  
 │ │ └── ElasticSearchPedidoAdapter.java  
 │ ├── config  
 │ │ ├── DynamoDBConfig.java  
 │ │ └── ElasticSearchConfig.java  
 │ └── repository  
 │ └── DynamoDBPedidoRepository.java  
 └── resources  
 └── application.properties

As principais classes desse módulo são:

DynamoDBPedidoAdapter: implementa as interfaces LoadPedidoPort e PersistPedidoPort usando o repositório do DynamoDB. Essa classe faz o mapeamento entre as entidades Pedido e PedidoItem e as tabelas do DynamoDB.  
ElasticSearchPedidoAdapter: implementa as interfaces LoadPedidoPort, PersistPedidoPort e SearchPedidosPort usando o cliente do ElasticSearch. Essa classe faz o mapeamento entre as entidades Pedido e PedidoItem e os documentos do ElasticSearch.  
DynamoDBConfig: define a configuração do DynamoDB, como o endpoint, a região, as credenciais e as tabelas. Essa classe usa as anotações do Spring Boot para criar os beans necessários para o DynamoDB.  
ElasticSearchConfig: define a configuração do ElasticSearch, como o endpoint, a porta, o esquema e o índice. Essa classe usa as anotações do Spring Boot para criar os beans necessários para o ElasticSearch.  
DynamoDBPedidoRepository: implementa a interface PedidoRepository usando o repositório do Spring Data DynamoDB. Essa classe usa as anotações do Spring Boot para definir os métodos CRUD para o pedido.  
application.properties: contém as propriedades da aplicação, como o nome, a versão, a porta e os perfis.

projeto  
├─ README.md  
├─ pom.xml  
└── src  
 └── main  
 ├── java  
 │ └── com  
 │ └── example  
 │ └── mslibrary  
 │ │── application  
 │ │ ├── controller  
 │ │ │ └── PedidoCreditoController.java  
 │ │ ├── dto  
 │ │ │ ├── PedidoCreditoDTO.java  
 │ │ │ └── PedidoCreditoMapper.java  
 │ │ └── service  
 │ │ └── PedidoCreditoService.java  
 │ │── domain  
 │ │ ├── entity  
 │ │ │ ├── Pedido.java  
 │ │ │ └── PedidoItem.java  
 │ │ ├── port  
 │ │ │ ├── in  
 │ │ │ │ ├── CreatePedidoUseCase.java  
 │ │ │ │ ├── DeletePedidoUseCase.java  
 │ │ │ │ ├── GetPedidoUseCase.java  
 │ │ │ │ ├── ListPedidosUseCase.java  
 │ │ │ │ └── UpdatePedidoUseCase.java  
 │ │ │ └── out  
 │ │ │ ├── LoadPedidoPort.java  
 │ │ │ ├── PersistPedidoPort.java  
 │ │ │ └── SearchPedidosPort.java  
 │ │ └── repository  
 │ │ └── PedidoRepository.java  
 │ └──── infrastructure   
 │ ├── adapter   
 │ | ├── DynamoDBPedidoAdapter.java   
 │ | └── ElasticSearchPedidoAdapter.java   
 │ ├── config   
 │ | ├── DynamoDBConfig.java   
 │ | └── ElasticSearchConfig.java   
 │ └── repository   
 │ └── DynamoDBPedidoRepository.java   
 └── resources   
 └── application.properties