Fiap - Faculdade de Informática e Administração Paulista São Paulo - SP, 2024

Challenge Plusoft Java Advanced

Integrantes:

RM552496 - Caio Boris Andrulis Correa

RM551938 - Denner de Oliveira Duarte

RM97861 - Lucas Petroni

RM552189 - Matheus Paulo de Lima Delgado

RM552538 - Nathaly Oliveira

Sumário:

Introdução	página 2
Objetivo do projeto	
Público-Alvo	página 3
Prazo e atividades	página 3
Diagrama de Classes	página 4
Diagrama de Relacionamento e Entidades	página 5
Endpoints e Evidências	página 6

Explicação sobre a Ideia

Nosso projeto visa combinar inteligência artificial (IA) e análise de dados para fornecer insights valiosos aos vendedores de fertilizantes parceiros da Plusoft.

O objetivo é ajudar esses profissionais a rastrearem e identificarem áreas e períodos específicos onde a alocação estratégica de recursos pode potencializar as vendas. Através da coleta e análise de dados relevantes, como padrões climáticos, características do solo, histórico de vendas e dados demográficos, nossa plataforma será capaz de gerar recomendações precisas sobre quais produtos e esforços de vendas devem ser concentrados. Isso inclui insights sobre sazonalidade, demanda regional e tendências de mercado.

Além disso, a IA será capaz de prever demandas futuras com base em dados históricos e em tempo real, permitindo que os vendedores antecipem as necessidades dos agricultores e ofereçam soluções sob medida, e preventivas.

Com nossa solução de IA e Analytics, os vendedores de fertilizantes poderão tomar decisões mais informadas e estratégicas, aumentando assim a eficiência das operações de vendas e maximizando o potencial de lucro em suas regiões de atuação

Objetivo do Projeto:

O nosso projeto tem como objetivo utilizar IA e Analytics para solucionar e melhorar o fator de vendas de fornecedores de materiais para a indústria do agronegócio.

Nossa API, tem como princípio, reunir informações e armazená-las em um banco de dados para que seja feita a exploração desses dados de forma que alavanque e traga mais precisão e ajude a tomada de decisões de fornecedores de materiais.

Público Alvo:

Nosso projeto tem como objetivos atingir fornecedores e busca atingir indiretamente produtores do agronegócio, fazendo com que potencialize as vendas de materiais considerados fundamentais para a prática da atividade.

Prazos e Atividades:

Dia 5/04/24 - Iniciar a documentação trazendo os objetivos, público-alvo e explicação da ideia do projeto.

Dia 12/04/24 - Finalizar o desenvolvimento do projeto.

Dia 13/04/24 - Realizar teste nos endpoints da API.

Dia 14/04/24 - Finalizar a documentação, adicionando as informações finais sobre o projeto.

Dia 15/04/24 - Realizar a entrega da 1º sprint do projeto.

Diagrama de Classes:

Class Vendedor

- id: Long
- nome: String
- contato: String
- vendas List<vendas>()
- + getId(): Long
- + setId(id: Long): void
- + getNome(): String
- + setNome(nome: String): void
- + getContato(): String
- + setContato(contato: String): void
- + getVendas(): List<HistoricoVendas>
- + setVendas(vendas: List<HistoricoVendas>): void

Interface VendedorRepository

- + save(vendedor: Vendedor): Vendedor
- + deleteByld(id: Long): void + findByld(id: Long): Optional<Vendedor>
- + findAll(): Iterable<Vendedor>

Class VendedorService

- vendedorRepository: VendedorRepository
- factory: VendedorFactory
- + getAll(): List<VendedorDTO>
- + getById(id: Long): VendedorDTO
- + criarVendedor(vendedor: VendedorDTO): VendedorDTO
- + updateVendedor(id: Long, vendedor: VendedorDTO): VendedorDTO
- + deleteVendedor(id: Long): boolean

Class VendedorFactory

- historicoVendasFactory: HistoricoVendasFactory
- + toDto(vendedores: List<Vendedor>): List<VendedorDTO>
- + toDto(vendedor: Vendedor): VendedorDTO
- + toEntity(vendedores: List<VendedorDTO>): List<Vendedor>
- toEntity(vendedor: VendedorDTO): Vendedor

Class VendedorDTO

- id: Long nome: String
- contato: String
- vendas: List<HistoricoVendasDTO>
- + getId(): Long + setId(id: Long): void
- + getNome(): String + setNome(nome: String): void

- + getContato(): String + setContato(contato: String): void
- + getVendas(): List<HistoricoVendasDTO> + setVendas(vendas: List<HistoricoVendasDTO>): void

Class VendedorController

- vendedorService: VendedorService
- + findAll(): List<VendedorDTO>
- + findByld(id: long): VendedorDTO
- + criarVendedor(vendedor: VendedorDTO): ResponseEntity<VendedorDTO>
- + excluirVendedor(id: Long): ResponseEntity<Void> + atualizarVendedor(id: Long, vendedor: VendedorDTO):

ResponseEntity<VendedorDTO>

Class Cliente

id: Long nome: String

getId(): Long
setId(id: Long): void
getNome(): String
setNome(nome: String): void

Class HistoricoVendas

id: Long
idVendedor: Long
idCliente: Long
dataVenda: LocalDate
quantidade: Integer
valorTotal: Double

getId(): Long
setId(id: Long): void
getId(id: Long): void
getIdVendedor(): Long
setIdVendedor(): Long
setIdVendedor(): Long
setIdCliente(): Long
setIdCliente(idCliente: Long): void
getDataVenda(): LocalDate
setDataVenda(): LocalDate
setDataVenda(dataVenda: LocalDate): void
getQuantidade(): Integer
setQuantidade(): Integer
setQuantidade(): Jountidade: Integer): void
getValorTotal(): Double
setValorTotal(): Double
setValorTotal() void

Class InfoRegiao

id: Long
 temperatura: Double
 umidade: Double
 precipitacao: Double
 dataInfo: LocalDate

getId(): Long
setId(id: Long): void
getTemperatura(): Double
setTemperatura(temperatura: Double): void
getUmidade(): Double
setUmidade(umidade: Double): void
getPrecipitacao(): Double
setPrecipitacao(precipitacao: Double): void
getDataInfo(): LocalDate
setDataInfo(dataInfo: LocalDate): void

Regiao

- id: Long - nome: String

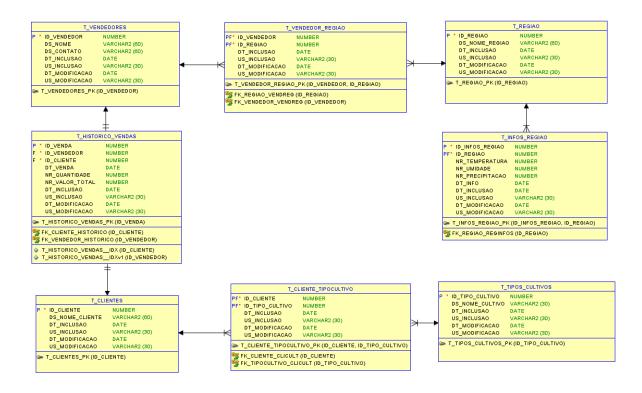
getId(): Long
setId(id: Long): void
getNome(): String
setNome(nome: String): void

TipoCultivo

id: Longdescricao: String

Métodos: + getId(): Long + setId(id: Long): void + getDescricao(): String + setDescricao(descricao: String): void

Diagrama de Entidades e Relacionamento (DER):



EndPoint e evidências:

GET: http://localhost:8080/vendedor

```
GET ▼ localhost:8080/vendedor Send ▼
```

```
{
    "id": 1,
    "nome": "João Silva",
    "contato": "joao@email.com",
    sdas": [
      "id": 1,
   "idvendedor": 1,
   "idCliente": 1,
   "dataVenda": "2024-04-13",
   "quantidade": 100,
   "coloratal": 100,
             "valorTotal": 5000.0
"id": 2,
    "idVendedor": 2,
    "idCliente": 2,
    "iataVenda": "20
    'a": 15
          "dataVenda": "2024-04-13",
"quantidade": 150,
             "valorTotal": 7500.0
 },

{
    "id": 3,
    "nome": "Pedro Oliveira",
    "contato": "pedro@email.co
      "contato": "pedro@email.com",
      {
    "id": 3,
    "idVended
             "idVendedor": 3,
           "dataVenda": "2024-04-13",
"quantidade": 200,
             "valorTotal": 10000.0
```

GET com id como parâmetro: http://localhost:8080/vendedor/1

```
GET ▼ http://localhost:8080/vendedor/1
```

POST:

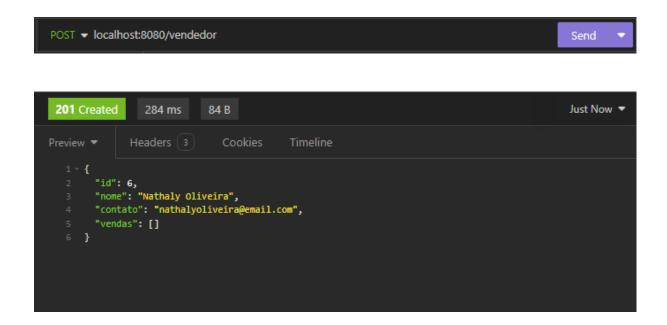
http://localhost:8080/vendedor

```
Request:

{
    "id": 6,
    "nome": "Nathaly",
    "contato": "nathaly@email.com"
}
```

Observação: No banco de dados foram criadas SEQUENCES e TRIGGERS para inserir o ID, porém, o Spring não permite que seja passado um objeto sem ID, dessa forma, até o momento é necessário passar o identificador da tabela na requisição POST.

Além disso, devido o uso do BeanValidation, se um objeto inválido for enviado, ocorrerá um erro na aplicação, porém posteriormente o projeto será ajustado para retornar a mensagem de erro de forma adequada.



PUT

http://localhost:8080/vendedor/6

```
Request:

{
     "id": 6,
     "nome": "Nathaly",
     "contato": "nathaly@tranquilo.com"
}
```

DELETE:

http://localhost:8080/vendedor/6

