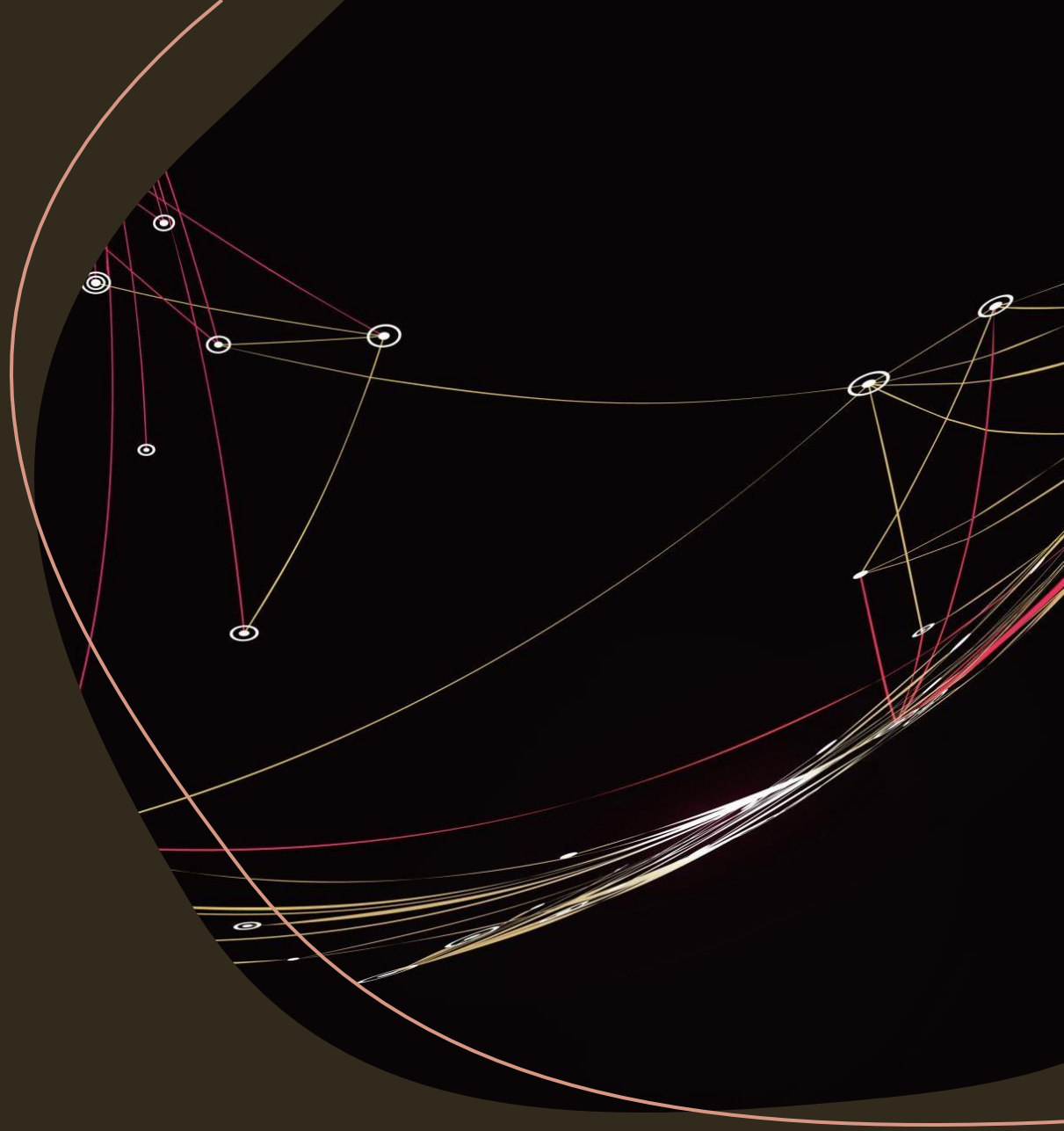
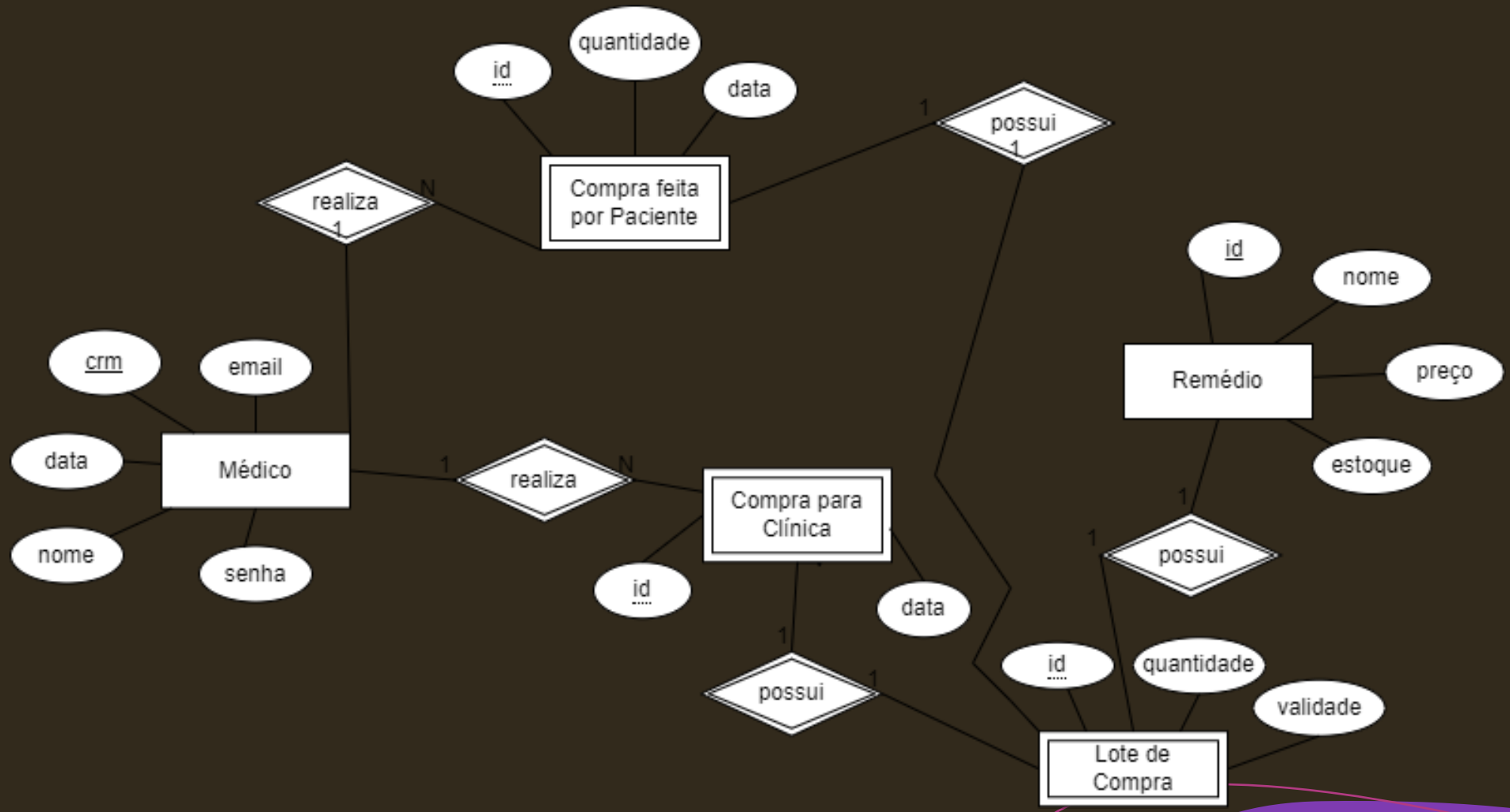
# Medstock

Integrantes:

- Felipe Rivetti Mizher
- Pedro Hosken Fernandes Guimarães
- Rafael Rehfeld Martins de Oliveira
- Vitor Augusto Viana Azevedo



# Diagrama


quantidade
id
data
possui
1
1
realiza
1
N
Compra feita
por Paciente
id
nome
crm
email
Remédio
preço
data
Médico
1
realiza
N
Compra para
Clínica
estoque
nome
senha
id
data
possui
1
1
possui
1
id
quantidade
validade
Lote de
Compra

# Conexão back-end ao Banco de dados

```java
public class Aplicacao {
    Run | Debug
    public static void main(String[] args) {
        // Configuração de porta
        port(4567);

        // Configuração do banco de dados
        String jdbcUrl = "jdbc:postgresql://medstocker.postgres.database.azure.com:5432/postgres?sslmode=require";
        String username = "vitor";
        String password = "Medstocker@";

        try {
            Connection connection = DriverManager.getConnection(jdbcUrl, username, password);

            // Criando instâncias dos DAOs
            MedicoDAO medicoDAO = new MedicoDAO(connection);
            RemedioDAO remedioDAO = new RemedioDAO(connection);
            LoteDAO loteDAO = new LoteDAO(connection);
            CompraMedicoDAO compraMedicoDAO = new CompraMedicoDAO(connection);
            CompraPacienteDAO compraPacienteDAO = new CompraPacienteDAO(connection);

            // Criando instâncias dos Services
            MedicoService medicoService = new MedicoService(medicoDAO);
            RemedioService remedioService = new RemedioService(remedioDAO);
            LoteService loteService = new LoteService(loteDAO, remedioDAO);
            CompraMedicoService compraMedicoService = new CompraMedicoService(compraMedicoDAO, medicoDAO, loteDAO);
            CompraPacienteService compraPacienteService = new CompraPacienteService(compraPacienteDAO, remedioDAO);
```

# Conexão back-end ao Banco de dados

INSERT INTO Remedio (nome, preco) Untitled-1

```
1  INSERT INTO Remedio (nome, preco)
2  VALUES ('Rivotril', 22.28);
3  SELECT * FROM "remedio";
```

untitled:untitled-1

4 rows returned

| | id integer | nome character varying | preco numeric | estoque integer |
|---|---|---|---|---|
| 1 | 4 | Naproxeno | 100.50 | 0 |
| 2 | 3 | Tylenol | 12.50 | 10 |
| 3 | 6 | Foxis | 10.50 | 0 |
| 4 | 1 | Paracetamol | 10.50 | 80 |

localhost:4567/remedios

Estilos de formatação

[{"id":4,"nome":"Naproxeno","preco":100.5,"estoque":0},{"id":3,"nome":"Tylenol","preco":12.5,"estoque":10},{"id":6,"nome":"Foxis","preco":10.5,"estoque":0},{"id":1,"nome":"Paracetamol","preco":10.5,"estoque":80}]

# Conexão back-end ao Banco de dados

INSERT INTO Remedio (nome, preco) Untitled-1

```
1  INSERT INTO Remedio (nome, preco)
2  VALUES ('Rivotril', 22.28);
3  SELECT * FROM "remedio";
```

untitled:untitled-1

1 row inserted

5 rows returned

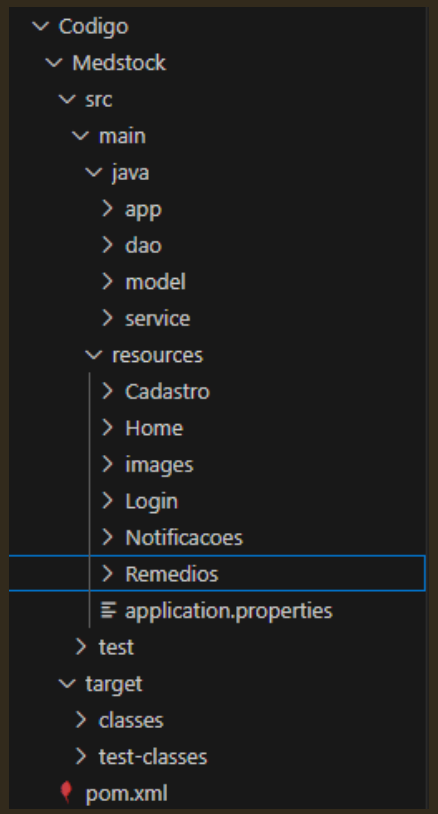
| | id integer | nome character varying | preco numeric | estoque integer |
|---|---|---|---|---|
| 1 | 4 | Naproxeno | 100.50 | 0 |
| 2 | 3 | Tylenol | 12.50 | 10 |
| 3 | 6 | Foxis | 10.50 | 0 |
| 4 | 1 | Paracetamol | 10.50 | 80 |
| 5 | 9 | Rivotril | 22.28 | 0 |

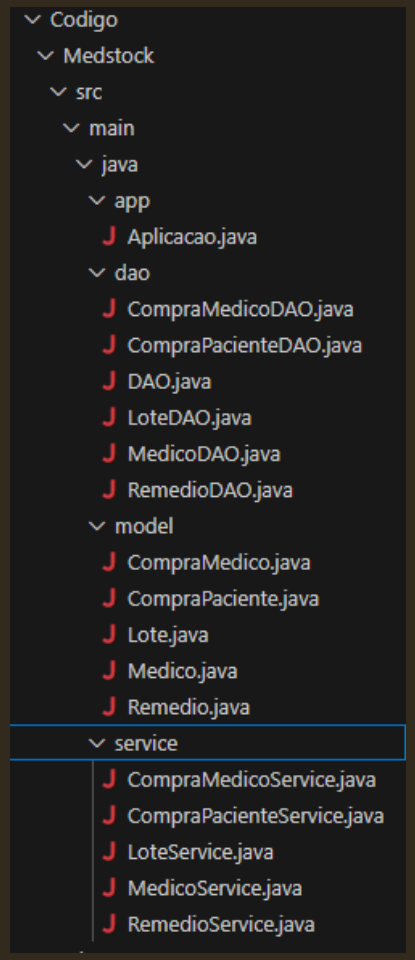
localhost:4567/remedios

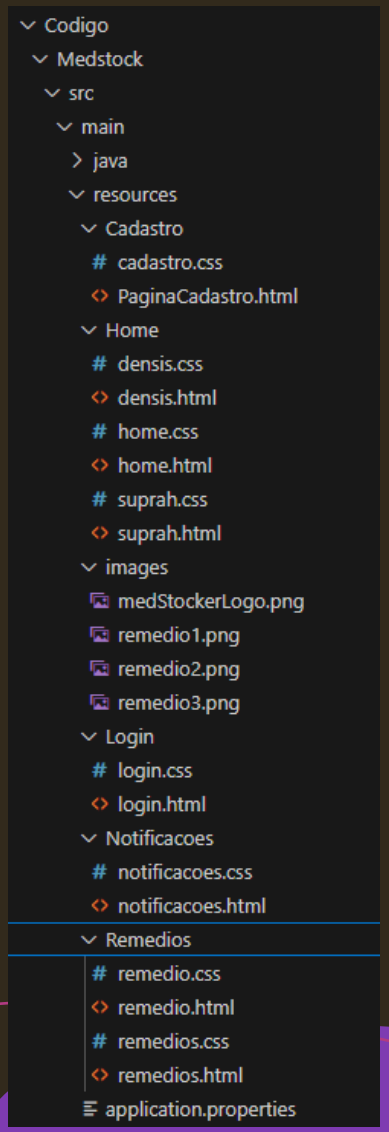
Estilos de formatação

[{"id":4,"nome":"Naproxeno","preco":100.5,"estoque":0},{"id":3,"nome":"Tylenol","preco":12.5,"estoque":10},{"id":6,"nome":"Foxis","preco":10.5,"estoque":0},{"id":1,"nome":"Paracetamol","preco":10.5,"estoque":80},{"id":9,"nome":"Rivotril","preco":22.28,"estoque":0}]

# Pastas do projeto



# Principios de desenvolvimento java (get/set)

```java
package model;

public class Medico {
    private int crm;
    private String nome;
    private String sobrenome;
    private String email;
    private String dataNascimento;

    // Construtor sem parâmetros
    public Medico() {
    }

    // Construtor completo
    public Medico(int crm, String nome, String sobrenome, String email, String dataNascimento) {
        this.crm = crm;
        this.nome = nome;
        this.sobrenome = sobrenome;
        this.email = email;
        this.dataNascimento = dataNascimento;
    }

    // Getters e Setters
    public int getCrm() {
        return crm;
    }

    public void setCrm(int crm) {
        this.crm = crm;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public String getSobrenome() {
        return sobrenome;
    }

    public void setSobrenome(String sobrenome) {
        this.sobrenome = sobrenome;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getDataNascimento() {
        return dataNascimento;
    }

    public void setDataNascimento(String dataNascimento) {
        this.dataNascimento = dataNascimento;
    }
}
```

# Código da Aplicação.java

```java
package app;

import static spark.Spark.*;
import com.google.gson.Gson;
import service.MedicoService;
import service.RemedioService;
import service.LoteService;
import service.CompraMedicoService;
import service.CompraPacienteService;
import model.Medico;
import model.Remedio;
import model.Lote;
import model.CompraMedico;
import model.CompraPaciente;
import dao.MedicoDAO;
import dao.RemedioDAO;
import dao.LoteDAO;
import dao.CompraMedicoDAO;
import dao.CompraPacienteDAO;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
```

```java
public class Aplicacao {
    public static void main(String[] args) {
        // Configuração de porta
        port(4567);

        // Configuração do banco de dados
        String jdbcUrl = "jdbc:postgresql://medstocker.postgres.database.azure.com:5432/postgres?sslmode=require";
        String username = "vitor";
        String password = "Medstocker@";

        try {
            Connection connection = DriverManager.getConnection(jdbcUrl, username, password);

            // Criando instâncias dos DAOs
            MedicoDAO medicoDAO = new MedicoDAO(connection);
            RemedioDAO remedioDAO = new RemedioDAO(connection);
            LoteDAO loteDAO = new LoteDAO(connection);
            CompraMedicoDAO compraMedicoDAO = new CompraMedicoDAO(connection);
            CompraPacienteDAO compraPacienteDAO = new CompraPacienteDAO(connection);

            // Criando instâncias dos Services
            MedicoService medicoService = new MedicoService(medicoDAO);
            RemedioService remedioService = new RemedioService(remedioDAO);
            LoteService loteService = new LoteService(loteDAO, remedioDAO);
```

```java
            CompraMedicoService compraMedicoService = new CompraMedicoService(compraMedicoDAO, medicoDAO, loteDAO);
            CompraPacienteService compraPacienteService = new CompraPacienteService(compraPacienteDAO, remedioDAO);

            // Usando Gson para conversão de JSON
            Gson gson = new Gson();

            // ------------------ Rotas para Médico ------------------

            // Rota para listar todos os médicos (GET)
            get("/medicos", (req, res) -> {
                res.type("application/json");
                return gson.toJson(medicoService.listarTodosMedicos());
            });

            // Rota para buscar um médico pelo CRM (GET)
            get("/medicos/:crm", (req, res) -> {
                res.type("application/json");
                int crm = Integer.parseInt(req.params("crm"));
                Medico medico = medicoService.buscarMedicoPorCRM(crm);
                if (medico != null) {
                    return gson.toJson(medico);
                } else {
                    res.status(404);
```

```
            return "Médico não encontrado";
        }
    });

    // Rota para cadastrar um médico (POST)
    post("/medicos", (req, res) -> {
        Medico medico = gson.fromJson(req.body(), Medico.class);
        medicoService.cadastrarMedico(medico);
        res.status(201); // Created
        return "Médico cadastrado com sucesso!";
    });

    // Rota para atualizar um médico (PUT)
    put("/medicos/:crm", (req, res) -> {
        Medico medico = gson.fromJson(req.body(), Medico.class);
        medicoService.atualizarMedico(medico);
        return "Médico atualizado com sucesso!";
    });

    // Rota para deletar um médico (DELETE)
    delete("/medicos/:crm", (req, res) -> {
        int crm = Integer.parseInt(req.params("crm"));
        medicoService.deletarMedico(crm);
```

```
        return "Médico deletado com sucesso!";
    });

    // ------------------- Rotas para Remédios -------------------

    // Rota para listar todos os remédios (GET)
    get("/remedios", (req, res) -> {
        res.type("application/json");
        return gson.toJson(remedioService.listarTodosRemedios());
    });

    // Rota para buscar um remédio por ID (GET)
    get("/remedios/:id", (req, res) -> {
        res.type("application/json");
        int id = Integer.parseInt(req.params("id"));
        Remedio remedio = remedioService.buscarRemedioPorId(id);
        if (remedio != null) {
            return gson.toJson(remedio);
        } else {
            res.status(404);
            return "Remédio não encontrado";
        }
    });
```

```

    // Rota para cadastrar um remédio (POST)
    post("/remedios", (req, res) -> {
        Remedio remedio = gson.fromJson(req.body(), Remedio.class);
        remedioService.cadastrarRemedio(remedio);
        res.status(201); // Created
        return "Remédio cadastrado com sucesso!";
    });

    // Rota para atualizar um remédio (PUT)
    put("/remedios/:id", (req, res) -> {
        Remedio remedio = gson.fromJson(req.body(), Remedio.class);
        remedioService.atualizarRemedio(remedio);
        return "Remédio atualizado com sucesso!";
    });

    // Rota para deletar um remédio (DELETE)
    delete("/remedios/:id", (req, res) -> {
        int id = Integer.parseInt(req.params("id"));
        remedioService.deletarRemedio(id);
        return "Remédio deletado com sucesso!";
    });

```

```
141             // ------------------ Rotas para Lotes ------------------
142
143             // Rota para listar todos os lotes (GET)
144             get("/lotes", (req, res) -> {
145                 res.type("application/json");
146                 return gson.toJson(loteService.listarTodosLotes());
147             });
148
149             // Rota para cadastrar um novo lote (POST)
150             post("/lotes", (req, res) -> {
151                 Lote lote = gson.fromJson(req.body(), Lote.class);
152                 loteService.cadastrarLote(lote);
153                 res.status(201); // Created
154                 return "Lote cadastrado com sucesso!";
155             });
156
157             // ------------------ Rotas para CompraMedico ------------------
158
159             // Rota para cadastrar uma nova compra de médico (POST)
160             post("/compramedico", (req, res) -> {
161                 CompraMedico compraMedico = gson.fromJson(req.body(), CompraMedico.class);
162                 compraMedicoService.cadastrarCompraMedico(compraMedico);
163                 res.status(201); // Created
```

```
164                 return "Compra do médico registrada com sucesso!";
165             });
166
167             // ------------------ Rotas para CompraPaciente ------------------
168
169             // Rota para cadastrar uma nova compra de paciente (POST)
170             post("/comprapaciente", (req, res) -> {
171                 CompraPaciente compraPaciente = gson.fromJson(req.body(), CompraPaciente.class);
172                 compraPacienteService.cadastrarCompraPaciente(compraPaciente);
173                 res.status(201); // Created
174                 return "Compra do paciente registrada com sucesso!";
175             });
176
177         } catch (SQLException e) {
178             e.printStackTrace();
179         }
180     }
181 }
```

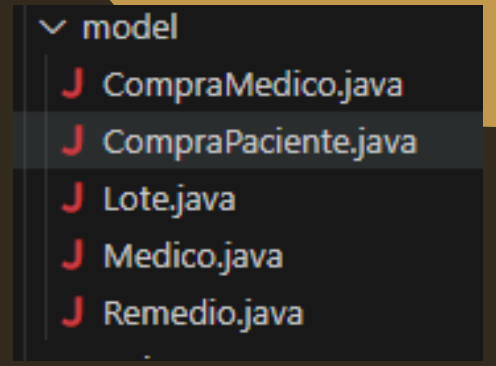
# Pacote DAO (insert/update/remove/get/


```java
public class MedicoDAO {
    private Connection connection;

    public MedicoDAO(Connection connection) {
        this.connection = connection;
    }

    // Método para inserir um médico no banco
    public void inserir(Medico medico) throws SQLException {
        String sql = "INSERT INTO Medico (crm, nome, sobrenome, email, data_nascimento) VALUES (?, ?, ?, ?, ?)";
        PreparedStatement statement = connection.prepareStatement(sql);
        statement.setInt(1, medico.getCrm());
        statement.setString(2, medico.getNome());
        statement.setString(3, medico.getSobrenome());
        statement.setString(4, medico.getEmail());
        statement.setString(5, medico.getDataNascimento());
        statement.executeUpdate();
    }

    // Método para buscar um médico por CRM
    public Medico buscarPorCRM(int crm) throws SQLException {
        String sql = "SELECT * FROM Medico WHERE crm = ?";
        PreparedStatement statement = connection.prepareStatement(sql);
        statement.setInt(1, crm);
        ResultSet resultSet = statement.executeQuery();

        if (resultSet.next()) {
            return new Medico(
                    resultSet.getInt("crm"),
                    resultSet.getString("nome"),
                    resultSet.getString("sobrenome"),
                    resultSet.getString("email"),
                    resultSet.getString("data_nascimento"));
        }
        return null; // Retorna null se o médico não for encontrado
    }

    // Método para atualizar um médico
    public void atualizar(Medico medico) throws SQLException {
        String sql = "UPDATE Medico SET nome = ?, sobrenome = ?, email = ?, data_nascimento = ? WHERE crm = ?";
        PreparedStatement statement = connection.prepareStatement(sql);
        statement.setString(1, medico.getNome());
        statement.setString(2, medico.getSobrenome());
        statement.setString(3, medico.getEmail());
        statement.setString(4, medico.getDataNascimento());
        statement.setInt(5, medico.getCrm());
        statement.executeUpdate();
    }

    // Método para deletar um médico
    public void deletar(int crm) throws SQLException {
        String sql = "DELETE FROM Medico WHERE crm = ?";
        PreparedStatement statement = connection.prepareStatement(sql);
        statement.setInt(1, crm);
        statement.executeUpdate();
    }

    // Método para listar todos os médicos
    public List<Medico> listarTodos() throws SQLException {
        String sql = "SELECT * FROM Medico";
        PreparedStatement statement = connection.prepareStatement(sql);
        ResultSet resultSet = statement.executeQuery();

        List<Medico> medicos = new ArrayList<>();
        while (resultSet.next()) {
            Medico medico = new Medico(
                    resultSet.getInt("crm"),
                    resultSet.getString("nome"),
                    resultSet.getString("sobrenome"),
                    resultSet.getString("email"),
                    resultSet.getString("data_nascimento"));
            medicos.add(medico);
        }
        return medicos;
    }
}
```

# Pacote DAO (método para abrir e fechar a conexão com o banco de dados)

```java
package dao;
import java.sql.*;

public class DAO {
    private String url = System.getenv("medstocker.postgres.database.azure.com");
    private String usuario = System.getenv("vitor");
    private String senha = System.getenv("MedStocker!");
    protected Connection conexao;

    public DAO() {
        try {
            conexao = DriverManager.getConnection(url, usuario, senha);
        } catch (SQLException e) {
            throw new RuntimeException("Erro ao conectar ao banco de dados: ", e);
        }
    }

    public ResultSet executeQuery(String sql) {
        try {
            Statement stmt = conexao.createStatement();
            return stmt.executeQuery(sql);
        } catch (SQLException e) {
            throw new RuntimeException("Erro ao executar consulta: ", e);
        }
    }

    public int executeUpdate(String sql) {
        try {
            Statement stmt = conexao.createStatement();
            return stmt.executeUpdate(sql);
        } catch (SQLException e) {
            throw new RuntimeException("Erro ao executar atualização: ", e);
        }
    }

    public PreparedStatement prepareStatement(String sql) throws SQLException {
        return conexao.prepareStatement(sql);
    }

    public void close() {
        try {
            if (conexao != null && !conexao.isClosed()) {
                conexao.close();
            }
        } catch (SQLException e) {
            throw new RuntimeException("Erro ao fechar conexão com o banco de dados: ", e);
        }
    }
}
```

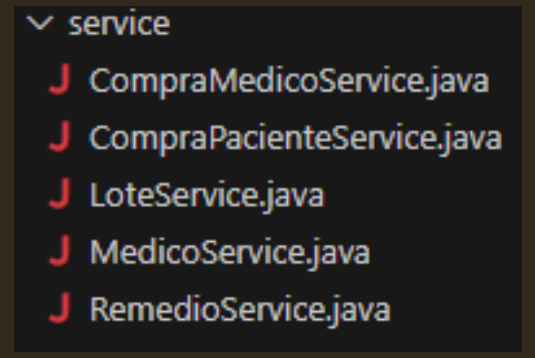
# Pacote Model




```java
package model;

public class Medico {
    private int crm;
    private String nome;
    private String sobrenome;
    private String email;
    private String dataNascimento;

    // Construtor sem parâmetros
    public Medico() {
    }

    // Construtor completo
    public Medico(int crm, String nome, String sobrenome, String email, String dataNascimento) {
        this.crm = crm;
        this.nome = nome;
        this.sobrenome = sobrenome;
        this.email = email;
        this.dataNascimento = dataNascimento;
    }

    // Getters e Setters
    public int getCrm() {
        return crm;
    }

    public void setCrm(int crm) {
        this.crm = crm;
    }

    public String getNome() {
        return nome;
    }
```

```java

    public String getSobrenome() {
        return sobrenome;
    }

    public void setSobrenome(String sobrenome) {
        this.sobrenome = sobrenome;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getDataNascimento() {
        return dataNascimento;
    }

    public void setDataNascimento(String dataNascimento) {
        this.dataNascimento = dataNascimento;
    }
```

# Pacote SERVICE




```java
public class MedicoService {
    private MedicoDAO medicoDAO;

    public MedicoService(MedicoDAO medicoDAO) {
        this.medicoDAO = medicoDAO;
    }

    // Método para cadastrar um novo médico
    public void cadastrarMedico(Medico medico) throws SQLException {
        medicoDAO.inserir(medico);
    }

    // Método para buscar médico pelo CRM
    public Medico buscarMedicoPorCRM(int crm) throws SQLException {
        return medicoDAO.buscarPorCRM(crm);
    }

    // Método para atualizar informações do médico
    public void atualizarMedico(Medico medico) throws SQLException {
        medicoDAO.atualizar(medico);
    }

    // Método para deletar um médico
    public void deletarMedico(int crm) throws SQLException {
        medicoDAO.deletar(crm);
    }

    // Método para listar todos os médicos
    public List<Medico> listarTodosMedicos() throws SQLException {
        return medicoDAO.listarTodos();
    }
}
```

# Inteligência do Sistema

- A inteligência do sistema reside na capacidade de capturar a imagem do rótulo do remédio, processá-la para extrair informações, e verificar automaticamente se o remédio está dentro da data de validade.

# Recursos e técnicas do sistema Inteligente

O sistema captura uma foto do rótulo do medicamento.

Processa a imagem e extrai informações textuais (como data de validade).

Verifica automaticamente se o medicamento está dentro da validade.

# Proposições de valores

- **Eficiência**: Automatização do processo de verificação da validade, eliminando a necessidade de entrada manual de dados.
- **Precisão**: Redução de erros humanos na identificação de datas de validade.

# Aprendizado da inteligência

O sistema usará um sistema de aprendizagem offline

Foco na natureza estável dos rótulos e datas, sem necessidade de constante adaptação.

Menor necessidade de recursos computacionais em tempo real.

# Entradas e saídas

**Entrada:** Imagem do rótulo do remédio.

**Saída:** Texto extraído (incluindo a data de validade) e status de validade (dentro ou fora da validade).

# Fornecedor de tecnologia

- Fornecedor: **Google Cloud** ou **Microsoft Azure**.
- Ambos possuem robustas soluções de IA como serviços.

# Serviços desse fornecedor

- **Google Cloud Vision API** ou **Azure Computer Vision API**.
- Reconhecimento de texto (OCR) para processamento das imagens capturadas.