Projeto Agendamento Visita Tecnica Jacto

Autor: Domingos Paulo Ferreira Neto

Esta documentação descreve a arquitetura da aplicação, destacando as principais tecnologias e componentes envolvidos.

Visão Geral:

A aplicação foi projetada para gerenciar o agendamento de visitas técnicas, desde o cadastro de clientes e fazendas até a notificação dos técnicos responsáveis. A arquitetura é composta por:

- **Camada de Persistência:** Banco de dados PostgreSQL para armazenamento de dados.
- **Camada de Mensageria:** Filas JMS/ActiveMQ para comunicação assíncrona e processamento de tarefas em segundo plano.
- **Serviço de Envio de Email:** JavaMail e MailHog para envio e teste de e-mails.
- **Serviço de Geolocalização:** API Nominatim para obter informações detalhadas sobre a localização das fazendas.
- **Agendamento de Tarefas:** Jobs automáticos para notificar os técnicos sobre as visitas próximas do horário.
- **Segurança:** Implementação de segurança com Spring Security e tokens JWT para autenticação e autorização.

Componentes Principais:

1. **Banco de Dados PostgreSQL:**
    - **Responsabilidade:** Armazenamento persistente dos dados da aplicação, incluindo informações sobre clientes, fazendas, funcionários, visitas técnicas, equipamentos, peças de reposição e outros dados relacionados.
    - **Tecnologia:** PostgreSQL (versão mais recente) com a extensão PostGIS (para suporte a dados geográficos).

2. **Filas JMS/ActiveMQ:**
    - **Responsabilidade:** Gerenciar tarefas assíncronas, como o envio de notificações de e-mail. Isso permite que a aplicação processe as solicitações de forma mais eficiente e evite bloqueios na interface do usuário.
    - **Tecnologia:** Apache ActiveMQ (versão mais recente) e Spring JMS.
    - **Configuração:**
        - A aplicação se conecta ao ActiveMQ através de JMS.
        - As configurações de conexão (URL, usuário, senha) são definidas no arquivo `application.properties` ou por meio de variáveis de ambiente.
        - As mensagens são enviadas para filas específicas no ActiveMQ.
3. **Serviço de Envio de Email (JavaMail e MailHog):**
    - **Responsabilidade:** Enviar e-mails de notificação e confirmação aos usuários.

- **Tecnologia:** JavaMailSender (Spring Framework) e MailHog (servidor SMTP de teste).
- **Configuração:**
    - O `JavaMailSender` é configurado com as informações do servidor SMTP (host, porta, usuário, senha).
    - Em ambientes de desenvolvimento, o MailHog é usado para interceptar e-mails e visualizá-los em uma interface web, sem enviá-los para contas de e-mail reais.
    - Em ambientes de produção, um serviço de e-mail transacional (como SendGrid, Mailjet ou Amazon SES) é usado para enviar e-mails de forma confiável.

4. **Serviço de Geolocalização (API Nominatim):**
    - **Responsabilidade:** Obter informações detalhadas sobre a localização das fazendas (cidade, estado, país, etc.) a partir de suas coordenadas (latitude e longitude).
    - **Tecnologia:** OpenStreetMap Nominatim API e RestTemplate (Spring Framework).
    - **Configuração:**
        - A URL da API Nominatim é definida como uma propriedade no arquivo `application.properties`.
        - O `RestTemplate` é usado para fazer requisições HTTP para a API Nominatim.
        - A resposta JSON da API é processada para extrair os dados relevantes da localização.
5. **Jobs Automáticos (Spring Scheduling):**
    - **Responsabilidade:** Executar tarefas periódicas, como buscar as visitas técnicas que serão realizadas dentro da próxima hora e enviar notificações aos técnicos responsáveis.
    - **Tecnologia:** Spring Scheduling e anotação `@Scheduled`.

6. **Segurança (Spring Security e JWT):**
    - **Responsabilidade:** Autenticar e autorizar os usuários da aplicação, protegendo os endpoints da API e garantindo que apenas usuários autorizados possam acessar determinadas funcionalidades.
    - **Tecnologia:** Spring Security e JSON Web Tokens (JWT).
    - **Configuração:**
        - O Spring Security é configurado para usar autenticação baseada em JWT.
        - Quando um usuário faz login com sucesso, um token JWT é gerado e retornado ao cliente.
        - O cliente deve incluir o token JWT no cabeçalho de autorização das requisições subsequentes.
        - O Spring Security valida o token JWT e autoriza o acesso ao endpoint se o token for válido e tiver as permissões necessárias.

Para executar a aplicacao:

Pré-requisitos:

- Docker instalado e em execução no seu sistema.
- Git instalado para clonar o repositório.

Passo a Passo:

- **Clonar o Repositório:**
  - Abra um terminal e execute o seguinte comando para clonar o repositório da aplicação:

    ```
    git clone https://github.com/domingospaulo/projetojacto
    ```

- **Navegar até o Diretório do Projeto:**
  - Navegue até o diretório onde o projeto foi clonado:

    ```
    cd projetojacto
    ```

- **Iniciar os Serviços Docker (PostgreSQL, MailHog e ActiveMQ):**
  - Execute os seguintes comandos para iniciar os serviços Docker:

    ```
    docker run -d \ --name postgres-agendamento \ -p 5433:5432 \ -e POSTGRES_DB=agendamento \ -e POSTGRES_USER=postgres \ -e POSTGRES_PASSWORD=Admin123# \ postgres:latest
    ```

    ```
    docker run -d -p 1027:1025 -p 8027:8025 mailhog/mailhog
    ```

    ```
    docker run -d --name=activemq -p 61616:61616 -p 8161:8161 rmohr/activemq
    ```

  - **Explicação dos Comandos:**
    - **PostgreSQL:** Inicia um container PostgreSQL com o nome `postgres-agendamento`, mapeia a porta 5433 da sua máquina host para a porta 5432 do container, e define as variáveis de ambiente para o banco de dados, usuário e senha.
    - **MailHog:** Inicia um container MailHog que intercepta os emails de sua aplicação e permite visualizá-los em uma interface web, sem enviar para contas de email reais.
    - **ActiveMQ:** Inicia um container ActiveMQ com o nome `activemq`, mapeia as portas 61616 (para o protocolo ActiveMQ) e 8161 (para a interface web de administração).
- **Aguardar a Inicialização dos Serviços:**
  - Aguarde alguns instantes para que os containers do PostgreSQL, MailHog e ActiveMQ sejam inicializados completamente. Você pode verificar o status dos containers usando o comando:

```
docker ps
```

Certifique-se de que todos os containers estejam listados e com o status `Up`.

- **Compilar e Executar a Aplicação Spring Boot:**
  - No diretório raiz do projeto, execute o seguinte comando para compilar e executar a aplicação Spring Boot:

    ```
    ./mvnw spring-boot:run
    ```
- **Acessar a Aplicação e as Interfaces Web:**
  - **Aplicação Spring Boot:** Acesse a sua aplicação no navegador usando a URL:

    `http://localhost:8080`
  - **Swagger UI:**

    `http://localhost:8080/swagger-ui/index.html`
  - **MailHog:** Acesse a interface web do MailHog no navegador usando a URL:

    `http://localhost:8027`
  - **ActiveMQ:** Acesse a interface web de administração do ActiveMQ no navegador usando a URL:

    `http://localhost:8161`

    Use o nome de usuário e senha padrão (`admin` e `admin`) para fazer login.

Observações:

- Certifique-se de que as portas 5433, 1027, 8027, 61616 e 8161 não estão sendo usadas por outras aplicações na sua máquina.

Componentes de Implementação:

- **`DataLoader`:** Inicializa o banco com dados básicos e valida regras de CRUD.
- **`NotificacaoJob`:** Agenda e envia para a fila IDs de visitas técnicas futuras (próxima meia hora).
- **JMSListeners:** Consome a fila, busca detalhes das visitas e envia e-mails aos técnicos.
- **`EmailServiceImpl`:** Abstrai o envio de e-mails (usa MailHog em desenvolvimento).
- **`LocalizacaoServiceImpl`:** Obtém detalhes de endereço (cidade, estado, país) a partir de coordenadas (API Nominatim).
- **`SwaggerConfig`:** Configura a documentação da API (Swagger UI).
- **`AgendamentoDSConfig`:** Configura o acesso ao banco PostgreSQL local.
- **`SecurityConfig`:** Configura a segurança da aplicação (autenticação, autorização, JWT).

Foco:

- Cada componente tem uma responsabilidade específica e bem definida.

- A arquitetura visa escalabilidade e manutenibilidade.

Pontos de Melhoria Futura:

I. Robustez e Tratamento de Exceções:

- **Problema:** O tratamento de exceções atual pode ser genérico e não fornece informações detalhadas sobre a causa raiz dos problemas, dificultando a depuração e a resolução.
- **Solução:**
    - **Exceções Especializadas:** Criar classes de exceção personalizadas (ex: `VisitaTecnicaNaoEncontradaException`, `FuncionarioInvalidoException`, `IntegracaoNominatimException`) para representar erros específicos que podem ocorrer na aplicação.
    - **`@ControllerAdvice`:** Implementar um `@ControllerAdvice` para centralizar o tratamento de exceções e converter as exceções personalizadas em respostas HTTP com códigos de status apropriados (ex: 404 Not Found, 400 Bad Request, 500 Internal Server Error).
    - **JSON de Resposta Padronizado:** Definir um formato JSON padrão para as respostas de erro da API, incluindo campos como `timestamp`, `status`, `error`, `message` e `path`. Isso facilita a interpretação dos erros pelos clientes da API.
    - **Logs Detalhados:** Registrar logs detalhados das exceções, incluindo o stack trace completo, para facilitar a depuração.

II. Testes Unitários e de Integração:

- **Problema:** A falta de testes automatizados torna difícil garantir a qualidade do código e a detecção de bugs.
- **Solução:**
    - **Testes Unitários:** Escrever testes unitários para os serviços, componentes e outras classes da aplicação para verificar se elas estão funcionando corretamente de forma isolada.
    - **Testes de Integração:** Escrever testes de integração para verificar se os diferentes componentes da aplicação estão integrados corretamente e se a aplicação está interagindo corretamente com o banco de dados, o ActiveMQ e a API Nominatim.
    - **Cobertura de Código:** Medir a cobertura de código dos testes para garantir que uma porcentagem razoável do código esteja sendo testada.
    - **Integração Contínua:** Integrar os testes ao seu pipeline de integração contínua para que eles sejam executados automaticamente sempre que o código for alterado.

III. Relatórios e Informações Gerenciais:

- **Problema:** A falta de relatórios dificulta o acompanhamento do desempenho do sistema e a identificação de áreas para melhoria.
- **Solução:**

- **Listagem de Visitas Técnicas Agendadas:** Implementar um endpoint da API que retorne uma lista paginada de visitas técnicas agendadas, com filtros para pesquisar por data, funcionário, cliente, fazenda, status, etc.
- **Avaliações dos Clientes:** Implementar um endpoint da API que retorne um relatório com as avaliações dos clientes, incluindo estatísticas como média, desvio padrão, etc.
- **Dashboard:** Criar um painel de controle (usando tecnologias como Thymeleaf, React ou Angular) que exiba os relatórios e as informações gerenciais de forma visualmente atraente.

IV. Aprimoramento da Segurança com UserDetails:

- **Problema:** O modelo de autenticação atual pode não ser flexível o suficiente para lidar com requisitos de autorização mais complexos.
- **Solução:**
  - **Implementar `UserDetailsService`:** Criar uma implementação de `UserDetailsService` que carregue as informações do usuário (incluindo as permissões) do banco de dados.
  - **Atribuir Permissões aos Perfis:** Associar as permissões diretamente aos perfis de usuário. Isso pode ser feito criando uma nova tabela `perfil_permissao` para representar o relacionamento muitos-para-muitos entre perfis e permissões.
  - **Usar `@PreAuthorize`:** Usar a anotação `@PreAuthorize` do Spring Security para proteger os endpoints da API com base nas permissões do usuário autenticado.
  - Ex: Utilizar a anotação `@PreAuthorize("hasAuthority('VISITA_TECNICA_CREATE')")`

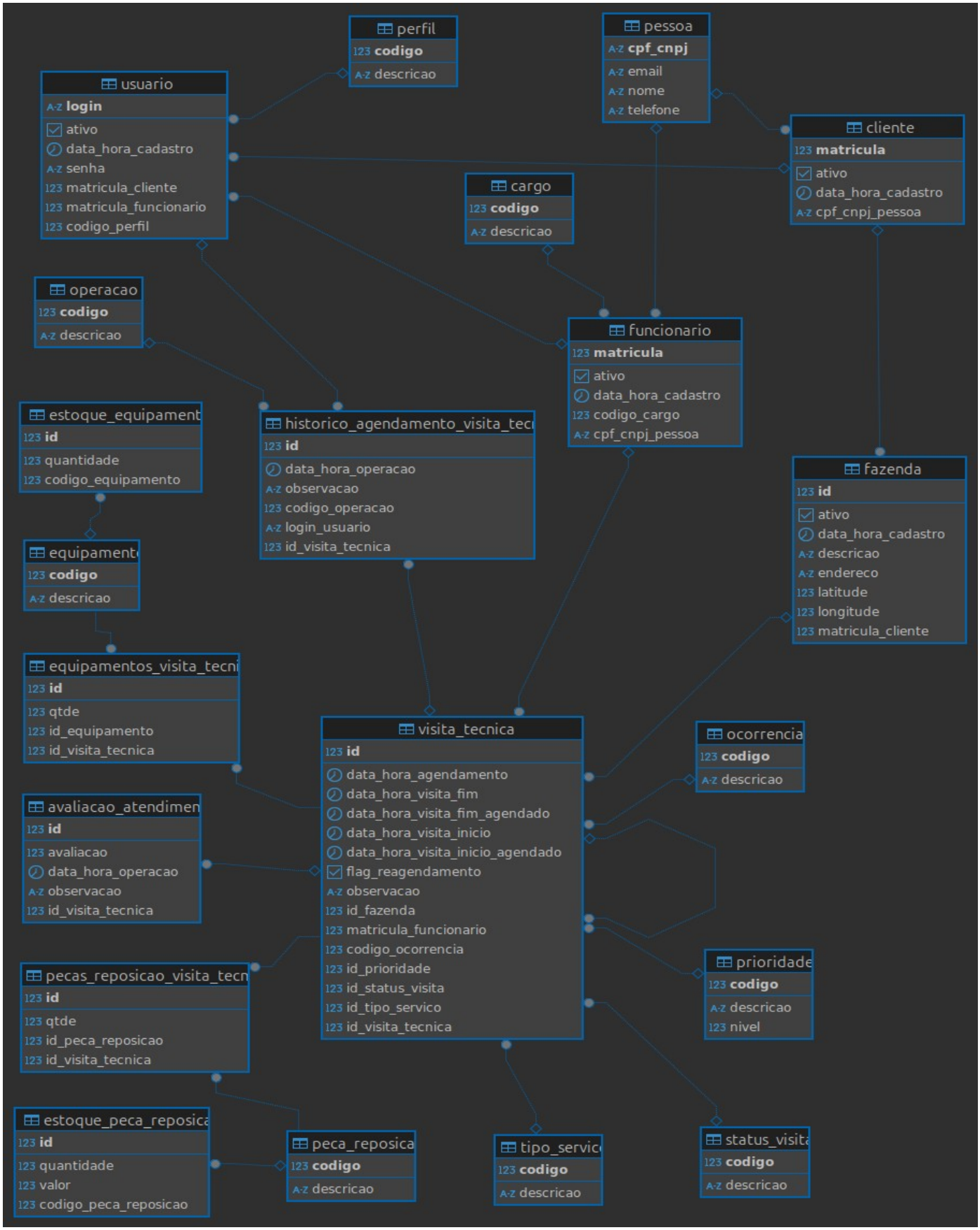
V. Implementação do Job de Priorização de Agendamentos:

- **Propósito:** Otimizar o agendamento das visitas técnicas, priorizando aquelas consideradas mais urgentes e garantindo o uso eficiente dos recursos disponíveis (técnicos e equipamentos).
- **Implementação:**
  1. **Buscar Visitas Reagendadas:** Selecionar todas as visitas técnicas com o campo `flag_reagendamento` definido como `true`.
  2. **Analisar Critérios de Priorização:**
     - **Tipo de Serviço:** Considerar a prioridade associada ao tipo de serviço (ex: "Reposição de Peça" > "Avaliação Técnica").
     - **Prioridade:** Usar o nível de prioridade definido na entidade `Prioridade`.
     - **Data de Agendamento:** Dar preferência para visitas agendadas para datas mais próximas.
  3. **Reagendamento:**
     - **Verificar Disponibilidade:** Para a visita de maior prioridade, verificar a disponibilidade do técnico responsável na data e hora agendadas.
     - **Alocação de Técnico Alternativo:** Se o técnico original não estiver disponível, verificar a disponibilidade de outros técnicos com as habilidades necessárias.
     - **Estoque de Equipamentos:** Verificar se o estoque de equipamentos necessários está disponível.

- **Ajustar Agendamento:** Se necessário, reagendar as visitas de menor prioridade para liberar o técnico e os equipamentos para a visita de maior prioridade.
- **Atualizar `flag_reagendamento`:** Definir o campo `flag_reagendamento` como `false` para as visitas que foram reagendadas.
- **Notificar Técnicos:** Enviar notificações aos técnicos sobre as mudanças no agendamento.

4. **Transacionalidade:** Executar toda a lógica de reagendamento dentro de uma transação para garantir a consistência dos dados.

Diagrama de Entidade-Relacionamento do Banco



O banco foi pensado de forma mais abrangente, considerando evolucoes futuras que não chegaram a ser totalmente utilizadas pela implementacao atual.

A Entidade **Perfil** foi pensada para servir de base na definicao dos perfis de acesso da aplicacao Informacoes como **Prioridade** e **Tipo de Servico** foram mapeadas para auxiliar na implementacao da regra de priorizacao dos agendamentos de visita tecnica.

Outras entidades como **Ocorrencia**, **Status Visita** e **Historico Agendamento** foram pensadas para fornecer mais informacoes sobre as operacoes realizadas pelo usuario da aplicacao, e como forma de auxiliar relatórios informativos para as areas de negocio interessadas.

Pontos de melhoria:

- Definir a nivel de banco os campos obrigatorios, como seguranca extra da garantia de integridade dos dados.
- Criacao de indices com base nos parametros das principais consultas das tabelas mais utilizadas pela aplicacao, como VisitaTecnica e HistoricoAgendamentoVisitaTecnica.
- No gerenciamento de perfis/acesso, adicionar entidades para armazenamento das funcionalidades habilitas para cada perfil.
- Considerar criacao da Entidade GrupoAcesso, para estender o potencial de configuracao de diversos perfis de acesso, levando em consideracao as funcionalidades liberadas e as operacoes liberadas para cada funcionalidade.
- Em EstoquePecaReposicao, considerar historico de alteracao dos valores das pecas, para garantir uma visao real dos valores das pecas no momento em que os pedidos de compra foram realizados.

Descricao Das Entidades e Relacionamentos

1. **usuario:**
    - **Descrição:** Representa os usuários do sistema, tanto clientes quanto funcionários.
    - **Atributos:**
        - `login` (PK): Login do usuário (String, chave primária).
        - `ativo`: Indica se o usuário está ativo (Boolean).
        - `data_hora_cadastro`: Data e hora do cadastro do usuário (DateTime).
        - `senha`: Senha do usuário (String).
        - `matricula_cliente` (FK): Matrícula do cliente (Integer, chave estrangeira para a entidade `cliente`).
        - `matricula_funcionario` (FK): Matrícula do funcionário (Integer, chave estrangeira para a entidade `funcionario`).
        - `codigo_perfil` (FK): Código do perfil (Integer, chave estrangeira para a entidade `perfil`).
    - **Relacionamentos:**
        - 1:1 com `cliente` (opcional): Um usuário pode ser um cliente.
        - 1:1 com `funcionario` (opcional): Um usuário pode ser um funcionário.
        - 1:1 com `perfil`: Um usuário tem um perfil.
2. **perfil:**

- **Descrição:** Define os perfis de acesso dos usuários.
- **Atributos:**
  - `codigo` (PK): Código do perfil (Integer, chave primária).
  - `descricao`: Descrição do perfil (String).

3. **pessoa:**
   - **Descrição:** Armazena informações básicas sobre pessoas (tanto clientes quanto funcionários).
   - **Atributos:**
     - `cpf_cnpj` (PK): CPF ou CNPJ da pessoa (String, chave primária).
     - `email`: Email da pessoa (String).
     - `nome`: Nome da pessoa (String).
     - `telefone`: Telefone da pessoa (String).
4. **cliente:**
   - **Descrição:** Representa os clientes.
   - **Atributos:**
     - `matricula` (PK): Matrícula do cliente (Integer, chave primária).
     - `ativo`: Indica se o cliente está ativo (Boolean).
     - `data_hora_cadastro`: Data e hora do cadastro do cliente (DateTime).
     - `cpf_cnpj_pessoa` (FK): CPF ou CNPJ da pessoa (String, chave estrangeira para a entidade `pessoa`).
   - **Relacionamentos:**
     - 1:1 com `pessoa`: Um cliente é uma pessoa.
5. **funcionario:**
   - **Descrição:** Representa os funcionários.
   - **Atributos:**
     - `matricula` (PK): Matrícula do funcionário (Integer, chave primária).
     - `ativo`: Indica se o funcionário está ativo (Boolean).
     - `data_hora_cadastro`: Data e hora do cadastro do funcionário (DateTime).
     - `codigo_cargo` (FK): Código do cargo (Integer, chave estrangeira para a entidade `cargo`).
     - `cpf_cnpj_pessoa` (FK): CPF ou CNPJ da pessoa (String, chave estrangeira para a entidade `pessoa`).
   - **Relacionamentos:**
     - 1:1 com `pessoa`: Um funcionário é uma pessoa.
     - 1:1 com `cargo`: Um funcionário tem um cargo.
6. **cargo:**
   - **Descrição:** Define os cargos dos funcionários.
   - **Atributos:**
     - `codigo` (PK): Código do cargo (Integer, chave primária).
     - `descricao`: Descrição do cargo (String).
7. **fazenda:**

- **Descrição:** Representa as fazendas dos clientes.
- **Atributos:**
  - `id` (PK): ID da fazenda (Integer, chave primária).
  - `ativo`: Indica se a fazenda está ativa (Boolean).
  - `data_hora_cadastro`: Data e hora do cadastro da fazenda (DateTime).
  - `descricao`: Descrição da fazenda (String).
  - `endereco`: Endereço da fazenda (String).
  - `latitude`: Latitude da fazenda (Double).
  - `longitude`: Longitude da fazenda (Double).
  - `matricula_cliente` (FK): Matrícula do cliente (Integer, chave estrangeira para a entidade `cliente`).
- **Relacionamentos:**
  - 1:N com `cliente`: Uma fazenda pertence a um cliente.

8. **visita_tecnica:**

- **Descrição:** Representa as visitas técnicas agendadas.
- **Atributos:**
  - `id` (PK): ID da visita técnica (Integer, chave primária).
  - `data_hora_agendamento`: Data e hora do agendamento (DateTime).
  - `data_hora_visita_inicio_agendado`: Data e hora de início agendada da visita (DateTime).
  - `data_hora_visita_fim_agendado`: Data e hora de fim agendada da visita (DateTime).
  - `data_hora_visita_inicio`: Data e hora de início real da visita (DateTime).
  - `data_hora_visita_fim`: Data e hora de fim real da visita (DateTime).
  - `observacao`: Observações sobre a visita (String).
  - `flag_reagendamento`: Indica se a visita foi reagendada (Boolean).
  - `id_fazenda` (FK): ID da fazenda (Integer, chave estrangeira para a entidade `fazenda`).
  - `matricula_funcionario` (FK): Matrícula do funcionário responsável (Integer, chave estrangeira para a entidade `funcionario`).
  - `codigo_ocorrencia` (FK): Código da ocorrência (Integer, chave estrangeira para a entidade `ocorrencia`).
  - `id_prioridade` (FK): ID da prioridade (Integer, chave estrangeira para a entidade `prioridade`).
  - `id_status_visita` (FK): ID do status da visita (Integer, chave estrangeira para a entidade `status_visita`).
  - `id_tipo_servico` (FK): ID do tipo de serviço (Integer, chave estrangeira para a entidade `tipo_servico`).
  - `id_visita_tecnica` (FK): ID da visita técnica de referência (Integer, chave estrangeira para a própria entidade, para representar relacionamentos de auto-referência).
- **Relacionamentos:**

- 1:N com `fazenda`: Uma visita técnica é realizada em uma fazenda.
- 1:N com `funcionario`: Uma visita técnica é realizada por um funcionário.
- 1:N com `ocorrencia`: Uma visita técnica pode ter uma ocorrência.
- 1:N com `prioridade`: Uma visita técnica tem uma prioridade.
- 1:N com `status_visita`: Uma visita técnica tem um status.
- 1:N com `tipo_servico`: Uma visita técnica é de um determinado tipo de serviço.
- 1:1 com `visita_tecnica` (auto-relacionamento): Uma visita técnica pode ser uma referência para outra visita técnica (por exemplo, para representar reagendamentos).

9. **tipo_servico:**
    - **Descrição:** Define os tipos de serviço oferecidos.
    - **Atributos:**
        - `codigo` (PK): Código do tipo de serviço (Integer, chave primária).
        - `descricao`: Descrição do tipo de serviço (String).
10. **prioridade:**
    - **Descrição:** Define as prioridades das visitas técnicas.
    - **Atributos:**
        - `codigo` (PK): Código da prioridade (Integer, chave primária).
        - `descricao`: Descrição da prioridade (String).
        - `nivel`: Nível da prioridade (Integer).
11. **status_visita:**
    - **Descrição:** Define os status das visitas técnicas.
    - **Atributos:**
        - `codigo` (PK): Código do status da visita (Integer, chave primária).
        - `descricao`: Descrição do status da visita (String).
12. **ocorrencia:**
    - **Descrição:** Define as ocorrências que podem ocorrer durante as visitas técnicas.
    - **Atributos:**
        - `codigo` (PK): Código da ocorrência (Integer, chave primária).
        - `descricao`: Descrição da ocorrência (String).
13. **equipamentos_visita_tecnica:**
    - **Descrição:** Tabela de junção que relaciona visitas técnicas a equipamentos.
    - **Atributos:**
        - `id` (PK): ID do registro (Integer, chave primária).
        - `qtde`: Quantidade do equipamento utilizado (Integer).
        - `id_equipamento` (FK): ID do equipamento (Integer, chave estrangeira para a entidade `equipamento`).
        - `id_visita_tecnica` (FK): ID da visita técnica (Integer, chave estrangeira para a entidade `visita_tecnica`).
    - **Relacionamentos:**

        - N:1 com `visita_tecnica`: Uma visita técnica pode ter vários equipamentos.
        - N:1 com `equipamento`: Um equipamento pode ser usado em várias visitas técnicas.

14.**avaliacao_atendimento:**

- **Descrição:** Representa a avaliação do atendimento da visita técnica.
- **Atributos:**
    - `id` (PK): ID da avaliação (Integer, chave primária).
    - `avaliacao`: Avaliação do atendimento (Integer).
    - `data_hora_operacao`: Data e hora da avaliação (DateTime).
    - `observacao`: Observações sobre a avaliação (String).
    - `id_visita_tecnica` (FK): ID da visita técnica (Integer, chave estrangeira para a entidade `visita_tecnica`).
- **Relacionamentos:**
    - N:1 com `visita_tecnica`: Uma avaliação de atendimento é para uma visita técnica.

15.**pecas_reposicao_visita_tecnica:**

- **Descrição:** Tabela de junção que relaciona visitas técnicas a peças de reposição.
- **Atributos:**
    - `id` (PK): ID do registro (Integer, chave primária).
    - `qtde`: Quantidade da peça de reposição utilizada (Integer).
    - `id_peca_reposicao` (FK): ID da peça de reposição (Integer, chave estrangeira para a entidade `peca_reposicao`).
    - `id_visita_tecnica` (FK): ID da visita técnica (Integer, chave estrangeira para a entidade `visita_tecnica`).
- **Relacionamentos:**
    - N:1 com `visita_tecnica`: Uma visita técnica pode ter várias peças de reposição.
    - N:1 com `peca_reposicao`: Uma peça de reposição pode ser usada em várias visitas técnicas.

16.**estoque_peca_reposica:**

- **Descrição:** Controla o estoque de peças de reposição.
- **Atributos:**
    - `id` (PK): ID do registro (Integer, chave primária).
    - `quantidade`: Quantidade em estoque (Integer).
    - `valor`: Valor da peça de reposição (Decimal).
    - `codigo_peca_reposicao` (FK): Código da peça de reposição (Integer, chave estrangeira para a entidade `peca_reposicao`).
- **Relacionamentos:**
    - N:1 com `peca_reposicao`: Um registro de estoque de peça de reposição é para uma peça de reposição.

17.**peca_reposica:**

- **Descrição:** Define as peças de reposição.
- **Atributos:**
    - `codigo` (PK): Código da peça de reposição (Integer, chave primária).
    - `descricao`: Descrição da peça de reposição (String).

18.**tipo_servic:**

- **Descrição:** Define os tipos de serviço oferecidos.
- **Atributos:**
    - `codigo` (PK): Código do tipo de serviço (Integer, chave primária).
    - `descricao`: Descrição do tipo de serviço (String).

19.**prioridade:**

- **Descrição:** Define as prioridades das visitas técnicas.
- **Atributos:**
    - `codigo` (PK): Código da prioridade (Integer, chave primária).
    - `descricao`: Descrição da prioridade (String).
    - `nivel`: Nível da prioridade (Integer).

20.**status_visita:**

- **Descrição:** Define os status das visitas técnicas.
- **Atributos:**
    - `codigo` (PK): Código do status da visita (Integer, chave primária).
    - `descricao`: Descrição do status da visita (String).

21.**ocorrencia:**

- **Descrição:** Define as ocorrências que podem ocorrer durante as visitas técnicas.
- **Atributos:**
    - `codigo` (PK): Código da ocorrência (Integer, chave primária).
    - `descricao`: Descrição da ocorrência (String).

22.**equipamento:**

- **Descrição:** Define os tipos de equipamentos utilizados. * `codigo` (PK): Código do equipamento (Integer, chave primária). * `descricao`: Descrição do equipamento (String).

23.**estoque_equipamento:**

- **Descrição:** Controla o estoque de equipamentos.
    - `id` (PK): ID do registro (Integer, chave primária).
    - `quantidade`: Quantidade em estoque (Integer).
- `codigo_equipamento` (FK): Código do equipamento (Integer, chave estrangeira para a entidade `equipamento`).
    - **Relacionamentos:**
    - N:1 com `equipamento`: Um registro de estoque de equipamento é para um equipamento.

24.**historico_agendamento_visita_tec:** * **Descrição:** Registra o histórico de agendamentos das visitas técnicas. * **Atributos:**

- `id` (PK): ID do registro (Integer, chave primária).

- `data_hora_operacao`: Data e hora da operação (DateTime).
- `observacao`: Observações sobre a operação (String).
- `codigo_operacao` (FK): Código da operação (Integer, chave estrangeira para a entidade `operacao`).
- `login_usuario` (FK): Login do usuário que realizou a operação (String, chave estrangeira para a entidade `usuario`).
- `id_visita_tecnica` (FK): ID da visita técnica (Integer, chave estrangeira para a entidade `visita_tecnica`). * **Relacionamentos:**
- N:1 com `operacao`: Um registro de histórico é para uma operação.
- N:1 com `usuario`: Um registro de histórico é realizado por um usuário.
- N:1 com `visita_tecnica`: Um registro de histórico é para uma visita técnica.

25.**operacao:**

- **Descrição:** Define as operações que podem ser realizadas no sistema.
    - `codigo` (PK): Código da operação (Integer, chave primária).
    - `descricao`: Descrição da operação (String).