Escola Naval - Curso de Engenharia Naval/Armas e Electrónica - 2025/2026 - 1º Semestre

Programação em Java Orientada aos Objetos

Especificações para projeto de gestão da manutenção

Objetivos

Esta seção apresenta os objetivos operacionais, de qualidade de dados, usabilidade, medição e engenharia de software do sistema.

1. **Operacional:** O sistema visa gerir ativos, incluindo equipamentos, locais e hierarquias organizacionais, para facilitar a gestão e localização dos recursos de manutenção. Suportar a criação e gestão de ordens de trabalho corretivas e preventivas, com abertura, priorização baseada em criticidade, acompanhamento de estados e manutenção de histórico de alterações. Incluir funcionalidades para planeamento de manutenção preventiva com planos, periodicidades e eventos baseados em horas de uso ou condições específicas. Gerir calendário para atribuição de tarefas, considerando equipas, turnos, resolução de conflitos e balanceamento de carga de trabalho. Registar detalhadamente a execução de trabalhos, incluindo tempos gastos, materiais consumidos, causas e ações tomadas, além de anexos como fotografias. Gestão de intervenções realizadas por terceiros, com definição de níveis de serviço (SLAs - Acordos de Nível de Serviço), ordens externas e controlo de custos associados. Controle de inventário e movimentação de peças, incluindo entradas, saídas, gestão de depósitos e realização de inventários cíclicos. Sistema de solicitação e workflow para pedidos de manutenção, com portal simples para submissão e processo de aprovação. Gestão de utilizadores e perfis, com diferentes roles como técnico, planeador, gestor e solicitante.

2. **Qualidade de dados:** Garantir integridade através de transações e consistência, permitindo desfazer operações em caso de falhas em ordens de trabalho ou inventário. Implementar validação de dados e regras de negócio, utilizando validações e exceções para garantir integridade. Incluir medidas de segurança, como autenticação e autorização baseada em perfis e regras. Auditoria completa de dados, rastreando quem e quando alterou informações, com trilha por ordem de trabalho ou ativo. Cumprir com privacidade e GDPR (Regulamento Geral de Proteção de Dados), minimizando dados pessoais e implementando retenção básica. O modelo de dados deve ser relacional e normalizado na terceira forma normal (3FN), cobrindo ativos, ordens de trabalho, peças, utilizadores e perfis. Integridade referencial deve ser mantida.

3. **Usabilidade:** Proporcionar uma interface de utilizador eficiente, começando com consola simples e evoluindo para JavaFX (no módulos mais avançados), com navegação por módulos. Tabelas devem suportar filtros, ordenação, paginação e edição inline segura para melhor interação com dados. Formulários devem incluir validação em tempo real e mensagens de ajuda para guiar o utilizador. Dashboard inicial deve ser personalizável por perfil de utilizador para atender necessidades específicas. Experiência de utilizador para fluxo de ordens de trabalho deve seguir um wizard:

detalhe, recursos, aprovação e conclusão. A usabilidade deve ser priorizada, com interface acessível, suporte a teclado, tratamento de nulos e feedback ao utilizador.

4. **Medição:** Disponibilizar funcionalidades avançadas de pesquisa e filtro, permitindo buscas por texto, atributos e datas, com armazenamento de vistas personalizadas. Geração de relatórios operacionais, incluindo métricas como MTTR (Tempo Médio de Reparação)/MTBF (Tempo Médio Entre Falhas), backlog (acumulação) de trabalho e cumprimento de planos de manutenção. Dashboard em tempo real com KPIs (Indicadores Chave de Performance), cartões informativos e gráficos, com atualização reativa aos dados. Sistema de notificações para alertas sobre estados de ordens, prazos e níveis de stock. Capacidades de exportação e importação de dados em formatos CSV/Excel para ativos, peças e ordens de trabalho. Integração de contadores de uso, através de importação CSV, entrada manual ou APIs (Interfaces de Programação de Aplicações) simuladas. Gestão de anexos e documentos, incluindo manuais e fotografias, com metadados associados. Devem ser criadas vistas e queries para calcular KPIs como MTTR/MTBF e backlog por criticidade.

5. **Engenharia de software:** Aplicar modelação orientada a objetos explícita, utilizando composição em vez de herança quando apropriado, interfaces e polimorfismo para flexibilidade e extensibilidade. Desenvolver com uma arquitetura em camadas, separando a interface de utilizador por consola, domínio, aplicação/serviço e infraestrutura/persistência. Suportar persistência de dados através de ficheiros de texto, binários e/ou bases de dados, garantindo armazenamento e recuperação eficientes. Implementar transações e consistência, testes unitários com cobertura mínima, qualidade de código através de documentação e convenções, logs e monitorização com diferentes níveis, resiliência a erros com mensagens claras e retry limitado, extensibilidade com pontos de extensão para novos KPIs/estados e configuração de regras, portabilidade utilizando Java e ficheiros/SQLite.

Âmbito (funcional)

Esta seção define o escopo funcional do sistema, incluindo os principais módulos e funcionalidades cobertas.

- **Ativos & Localizações:** Cadastro de ativos, incluindo equipamentos, locais e hierarquias organizacionais, para facilitar a gestão e localização dos recursos de manutenção. Gestão de localizações com hierarquias.

- **Ordens de Trabalho:** Suporte à criação e gestão de ordens de trabalho corretivas, incluindo abertura, priorização baseada em criticidade, acompanhamento de estados e manutenção de histórico de alterações. Funcionalidades para planeamento de manutenção preventiva, com definição de planos, periodicidades e eventos baseados em horas de uso ou condições específicas.

- **Planos Preventivos:** Definição de planos preventivos com periodicidade temporal, por contagem de uso, ou ambos, incluindo eventos e janelas flexíveis.

- **Inventário de Peças:** Manutenção de catálogo de peças e sobressalentes, com identificadores únicos (SKU - Unidade de Manutenção de Estoque), controle de unidades disponíveis e níveis de reposição automática. Controle de inventário e movimentação de peças, incluindo entradas, saídas, gestão de depósitos e realização de inventários cíclicos.

- **Utilizadores & Perfis:** Gestão de utilizadores e perfis, com diferentes roles (papéis) como técnico, planeador, gestor e solicitante. Sistema de solicitação e workflow para pedidos de manutenção, com portal simples para submissão e processo de aprovação.

- **Relatórios & KPIs:** Funcionalidades avançadas de pesquisa e filtro, armazenamento de vistas personalizadas. Geração de relatórios operacionais com métricas como MTTR/MTBF, backlog.

Dashboard em tempo real com KPIs, cartões e gráficos. Sistema de notificações para alertas sobre estados, prazos, níveis de stock.

Requisitos Funcionais (RF)

Esta seção lista os requisitos funcionais, ou seja, as funcionalidades que o sistema deve implementar para atender às necessidades dos usuários.

- RF01 -- Cadastro de Ativos (equipamentos, locais, hierarquias)

  O sistema deve permitir o cadastro de ativos, incluindo equipamentos, locais e hierarquias organizacionais, para facilitar a gestão e localização dos recursos de manutenção.

- RF02 -- Catálogo de Peças/Sobressalentes (identificador (SKU), unidades, nível de reposição)

  Deve ser possível manter um catálogo de peças e sobressalentes, com identificadores únicos (SKU), controle de unidades disponíveis e níveis de reposição automática.

- RF03 -- Ordens de Trabalho (OT) corretivas: abertura, priorização, estado, histórico

  O sistema deve suportar a criação e gestão de ordens de trabalho corretivas, incluindo abertura, priorização baseada em criticidade, acompanhamento de estados e manutenção de histórico de alterações.

- RF04 -- Planeamento de Manutenção Preventiva (planos, periodicidades, eventos por horas/uso)

  Deve incluir funcionalidades para planeamento de manutenção preventiva, com definição de planos, periodicidades e eventos baseados em horas de uso ou condições específicas.

- RF05 -- Gestão de Calendário & Atribuição (equipa, turnos, conflitos, carga de trabalho)

  Gestão de calendário para atribuição de tarefas, considerando equipas, turnos, resolução de conflitos e balanceamento de carga de trabalho.

- RF06 -- Registo de Execução (tempos, materiais consumidos, causa/ação, anexos/fotos)

  Registo detalhado da execução de trabalhos, incluindo tempos gastos, materiais consumidos, causas e ações tomadas, além de anexos como fotografias.

- RF07 -- Gestão de Intervenções de Terceiros (níveis de serviço (SLAs), ordens externas, custos)

  Gestão de intervenções realizadas por terceiros, com definição de níveis de serviço (SLAs), ordens externas e controlo de custos associados.

- RF08 -- Inventário & Movimentação de Peças (entradas/saídas, depósitos, inventário cíclico)

  Controle de inventário e movimentação de peças, incluindo entradas, saídas, gestão de depósitos e realização de inventários cíclicos.

- RF09 -- Solicitação/Workflow de Pedido de Manutenção (portal simples + aprovação)

  Sistema de solicitação e workflow para pedidos de manutenção, com portal simples para submissão e processo de aprovação.

- RF10 -- Tabelas de Apoio & Parametrização (prioridades, categorias de falha, centros de custo)

Tabelas de apoio e parametrização do sistema, incluindo prioridades, categorias de falha e centros de custo.

- RF11 -- Pesquisa e Filtro Avançados (texto, atributos, datas; armazenamento de vistas)

  Funcionalidades avançadas de pesquisa e filtro, permitindo buscas por texto, atributos e datas, com armazenamento de vistas personalizadas.

- RF12 -- Relatórios Operacionais (MTTR/MTBF, backlog, cumprimento de plano)

  Geração de relatórios operacionais, incluindo métricas como MTTR/MTBF, backlog de trabalho e cumprimento de planos de manutenção.

- RF13 -- Dashboard em Tempo Real (KPIs, cartões, gráficos; atualização reativa)

  Dashboard em tempo real com KPIs, cartões informativos e gráficos, com atualização reativa aos dados.

- RF14 -- Notificações por estados, prazos, níveis de stock

  Sistema de notificações para alertas sobre estados de ordens, prazos e níveis de stock.

- RF15 -- Integração de Contadores (CSV/manual; APIs simuladas)

  Integração de contadores de uso, através de importação CSV, entrada manual ou APIs simuladas.

- RF16 -- Exportação/Importação (CSV/Excel para ativos, peças, OTs)

  Capacidades de exportação e importação de dados em formatos CSV/Excel para ativos, peças e ordens de trabalho.

- RF17 -- Gestão de Utilizadores & Perfis (técnico, planeador, gestor, solicitante)

  Gestão de utilizadores e perfis, com diferentes roles como técnico, planeador, gestor e solicitante.

- RF18 -- Auditoria de Dados (quem/quando alterou; trilha por OT/ativo)

  Auditoria completa de dados, rastreando quem e quando alterou informações, com trilha por ordem de trabalho ou ativo.

- RF19 -- Anexos/Documentos (manuais, fotografias; metadados)

  Gestão de anexos e documentos, incluindo manuais e fotografias, com metadados associados.

Requisitos Não-Funcionais (RNF)

Esta seção descreve os requisitos não-funcionais, como performance, segurança, usabilidade e qualidade de código.

- RNF01 -- Arquitetura em Camadas (UI (Interface do Utilizador) por consola, domínio, aplicação/serviço, infra/persistência)

  O sistema deve ser desenvolvido com uma arquitetura em camadas, separando a interface de utilizador por consola, domínio, aplicação/serviço e infraestrutura/persistência, para promover modularidade e manutenção.

- RNF02 -- Modelação OO explícita (composição vs herança, interfaces, polimorfismo)

A modelação orientada a objetos deve ser explícita, utilizando composição em vez de herança quando apropriado, interfaces e polimorfismo para flexibilidade e extensibilidade.

- RNF03 -- Persistência (Ficheiros de texto/binários/Bases de Dados)

  O sistema deve suportar persistência de dados através de ficheiros de texto, binários e/ou bases de dados, garantindo armazenamento e recuperação eficientes.

- RNF04 -- Transações & Consistência (desfazer em caso de falhas de OT/inventário)

  Deve implementar transações e consistência, permitindo desfazer operações em caso de falhas em ordens de trabalho ou inventário.

- RNF05 -- Validação & Regras (validação; exceções)

  Validação de dados e regras de negócio devem ser implementadas, utilizando validações e exceções para garantir integridade.

- RNF06 -- Segurança (autenticação, autorização por perfil/regra)

  O sistema deve incluir medidas de segurança, como autenticação e autorização baseada em perfis e regras.

- RNF07 -- Usabilidade (UI por consola, acessibilidade básica, teclado, estados vazios, feedback)

  A usabilidade deve ser priorizada, com interface por consola acessível, suporte a teclado, tratamento de estados vazios e feedback ao utilizador.

- RNF08 -- Performance Base (consultas paginadas)

  Performance básica deve ser assegurada, com consultas paginadas para lidar com grandes volumes de dados.

- RNF09 -- Testes (unitários; cobertura mínima)

  O código deve ser testado com testes unitários, atingindo cobertura mínima para garantir qualidade.

- RNF10 -- Qualidade de Código (documentação; convenções)

  Qualidade de código deve ser mantida através de documentação e adesão a convenções de codificação.

- RNF11 -- Logs & Monitorização (níveis)

  Sistema de logs e monitorização deve ser implementado, com diferentes níveis de logging.

- RNF12 -- Resiliência & Erros (mensagens claras; retry limitado em I/O)

  O sistema deve ser resiliente a erros, com mensagens claras e retry limitado em operações de I/O.

- RNF13 -- Privacidade & GDPR (minimização de dados pessoais; retenção básica)

  Deve cumprir com privacidade e GDPR, minimizando dados pessoais e implementando retenção básica.

- RNF14 -- Extensibilidade (pontos de extensão para novos KPIs/estados; config de regras)

O sistema deve ser extensível, com pontos de extensão para novos KPIs, estados e configuração de regras.

- RNF15 -- Portabilidade (Java; bases de dados)

  Deve ser portável, utilizando Java 17+ e bases de dados para compatibilidade.

Requisitos de Interface com o Utilizador (GUI)

Esta seção especifica os requisitos para a interface gráfica do usuário, incluindo navegação, tabelas e formulários.

- GUI01 -- Consola simples (JavaFX em níveis mais avançados) com navegação por módulos

  A interface deve ser uma consola simples, com possibilidade de evolução para JavaFX em níveis mais avançados, incluindo navegação por módulos.

- GUI02 -- Tabelas com filtros, ordenação, paginação e edição inline segura

  Tabelas devem suportar filtros, ordenação, paginação e edição inline segura para melhor interação com dados.

- GUI03 -- Formulários com validação em tempo real e mensagens de ajuda

  Formulários devem incluir validação em tempo real e mensagens de ajuda para guiar o utilizador.

- GUI04 -- Dashboard inicial personalizável por perfil

  Dashboard inicial deve ser personalizável por perfil de utilizador para atender necessidades específicas.

- GUI05 -- UX de fluxo para OT (wizard: detalhe → recursos → aprovação → conclusão)

  Experiência de utilizador para fluxo de ordens de trabalho deve seguir um wizard: detalhe, recursos, aprovação e conclusão.

Requisitos de Dados / Esquema Relacional

Esta seção define o modelo de dados relacional e os requisitos para a persistência e integridade dos dados.

- D01 -- Modelo relacional normalizado (3FN) para ativos, OTs, peças, utilizadores, perfis

  O modelo de dados deve ser relacional e normalizado na terceira forma normal (3FN), cobrindo ativos, ordens de trabalho, peças, utilizadores e perfis.

- D02 -- Vistas/queries para KPIs (MTTR/MTBF, backlog por criticidade)

  Devem ser criadas vistas e queries para calcular KPIs como MTTR/MTBF e backlog por criticidade.

- D03 -- Integridade referencial, cascatas controladas, triggers leves (p.ex. atualização de stock)

  Integridade referencial deve ser mantida, com cascatas controladas e triggers leves para operações como atualização de stock.

Casos de Uso (núcleo)

Esta seção descreve os casos de uso principais do sistema, detalhando os atores, fluxos e regras de negócio.

- **UC01 -- Cadastrar Ativo:** criar/editar/arquivar; associar localização e hierarquia (pai/filho).
- **UC02 -- Criar OT Corretiva:** a partir de pedido (UC05) ou direta; priorizar; atribuir técnico; estados.
- **UC03 -- Executar OT:** registar tempos, peças consumidas, causa/ação, anexos; concluir.
- **UC04 -- Planeamento Preventivo:** definir plano (tempo/uso), gerar OTs preventivas, reagendar.
- **UC05 -- Pedido de Manutenção (Solicitante):** submeter pedido; aprovação do gestor; conversão em OT.
- **UC06 -- Inventário de Peças:** entradas/saídas, ponto de reposição, múltiplos depósitos.
- **UC07 -- Relatórios & KPIs:** gerar relatórios e ver dashboard.
- **UC08 -- Administração:** perfis, parametrizações, categorias, prioridades.

Para cada UC, descrever: Atores, Pré-condições, Fluxo principal, Fluxos alternativos, Pós-condições, Regras de negócio e Erros.

Modelo de Domínio (visão textual)

Esta seção apresenta o modelo de domínio textual, incluindo entidades, atributos e relacionamentos.

Entidades principais e relações (→ associação, ◆ composição, △ herança):

- **Ativo** (id, código, nome, estado, criticidade, idAtivoPai?)
  → ◆ **Contador** (id, tipo, leituraAtual, unidade)
  → ◆ **Documento** (id, tipo, path, meta)
- **Localização** (id, nome, pai?) --- Ativo ↔ Localização (N:1).
- **OT** (id, tipo {Corretiva, Preventiva}, prioridade, estado, descrição, datas, custoTotal, idAtivo)
  → ◆ **TarefaOT** (id, descrição, duraçãoPlaneada)
  → ◆ **ConsumoPeça** (id, idPeça, quantidade, custo)
  → ◆ **RegistoExecução** (id, horaInício, horaFim, causa, ação, técnico)
- **PlanoPreventivo** (id, idAtivo, política {tempo|uso|mista}, periodicidade, janela, últimoDisparo)
  → ◆ **RegraGatilho** (id, tipo {tempo|uso}, valor, unidade)
- **Peça** (id, sku, designação, unidade, pontoReposição)
  → ◆ **Stock** (id, depósito, quantidade)
  → ◆ **MovimentoStock** (id, tipo {entrada|saída|ajuste}, data, quantidade, idOT?)
- **Utilizador** △ **Técnico**, **Planeador**, **Gestor**, **Solicitante** (id, nome, email, perfil)
- **Auditoria** (id, entidade, idEntidade, campo, valorAntigo, valorNovo, timestamp, utilizador)

Roadmap por Nível (dificuldade crescente)

Esta seção propõe um plano de desenvolvimento incremental por níveis de dificuldade.

Nível 50 --- "Consola com ficheiros"

- **GUI01:** Consola simples (JavaFX em níveis mais avançados) com navegação por módulos
- **RNF03:** Persistência (Ficheiros de texto/binários/Bases de Dados)
- **Domínio mínimo:** Ativo, OT, Utilizador (Técnico), Peça.
- **Funcionalidades:**
    - RF01 -- Cadastro de Ativos (equipamentos, locais, hierarquias)
    - RF03 -- Ordens de Trabalho (OT) corretivas: abertura, priorização, estado, histórico
    - RF08 -- Inventário & Movimentação de Peças (entradas/saídas, depósitos, inventário cíclico)
    - RF11 -- Pesquisa e Filtro Avançados (texto, atributos, datas; armazenamento de vistas)
- **RNF05:** Validação & Regras (validação; exceções)
- **RNF09:** Testes (unitários; cobertura mínima)
- **RNF07:** Usabilidade (UI por consola, acessibilidade básica, teclado, estados vazios, feedback)

Nível 60 --- "Camadas + Ficheiros/SQLite + UI básica"

- **GUI02:** Tabelas com filtros, ordenação, paginação e edição inline segura
- **RNF03:** Persistência (Ficheiros de texto-binários/Bases de Dados)
- **RNF01:** Arquitetura em Camadas (UI por consola, domínio, aplicação/serviço, infra/persistência)
- **Funcionalidades novas:**
    - RF02 -- Catálogo de Peças/Sobressalentes (identificador (SKU), unidades, nível de reposição)
    - RF04 -- Planeamento de Manutenção Preventiva (planos, periodicidades, eventos por horas/uso)
    - RF05 -- Gestão de Calendário & Atribuição (equipa, turnos, conflitos, carga de trabalho)
    - RF14 -- Notificações por estados, prazos, níveis de stock
    - RF17 -- Gestão de Utilizadores & Perfis (técnico, planeador, gestor, solicitante)
- **RNF09:** Testes (unitários; cobertura mínima)
- **RNF07:** Usabilidade (UI por consola, acessibilidade básica, teclado, estados vazios, feedback)
- **D01:** Modelo relacional normalizado (3FN) para ativos, OTs, peças, utilizadores, perfis

- **D03:** Integridade referencial, cascatas controladas, triggers leves (p.ex. atualização de stock)

Nível 70 --- "Manutenção Preventiva por contadores + relatórios + auditoria"

- **GUI02:** Tabelas com filtros, ordenação, paginação e edição inline segura
- **Funcionalidades novas:**
    - RF04 -- Planeamento de Manutenção Preventiva (planos, periodicidades, eventos por horas/uso)
    - RF12 -- Relatórios Operacionais (MTTR/MTBF, backlog, cumprimento de plano)
    - RF18 -- Auditoria de Dados (quem/quando alterou; trilha por OT/ativo)
    - RF16 -- Exportação/Importação (CSV/Excel para ativos, peças, OTs)
- **RNF08:** Performance Base (consultas paginadas)
- **RNF09:** Testes (unitários; cobertura mínima)
- **RNF07:** Usabilidade (UI por consola, acessibilidade básica, teclado, estados vazios, feedback)

Nível 80 --- "Dashboard, KPIs, workflow de pedido"

- **GUI04:** Dashboard inicial personalizável por perfil
- **Funcionalidades novas:**
    - RF09 -- Solicitação/Workflow de Pedido de Manutenção (portal simples + aprovação)
    - RF06 -- Registo de Execução (tempos, materiais consumidos, causa/ação, anexos/fotos)
    - RF05 -- Gestão de Calendário & Atribuição (equipa, turnos, conflitos, carga de trabalho)
    - RF19 -- Anexos/Documentos (manuais, fotografias; metadados)
- **RNF06:** Segurança (autenticação, autorização por perfil/regra)
- **RNF09:** Testes (unitários; cobertura mínima)
- **RNF07:** Usabilidade (UI por consola, acessibilidade básica, teclado, estados vazios, feedback)

Nível 90 --- "Inventário robusto + KPIs avançados + qualidade"

- **Funcionalidades novas:**
    - RF08 -- Inventário & Movimentação de Peças (entradas/saídas, depósitos, inventário cíclico)
    - RF12 -- Relatórios Operacionais (MTTR/MTBF, backlog, cumprimento de plano)
    - RF18 -- Auditoria de Dados (quem/quando alterou; trilha por OT/ativo)
    - RF15 -- Integração de Contadores (CSV/manual; APIs simuladas)

- **RNF10:** Qualidade de Código (documentação; convenções)
- **RNF08:** Performance Base (consultas paginadas)
- **RNF09:** Testes (unitários; cobertura mínima)
- **RNF07:** Usabilidade (UI por consola, acessibilidade básica, teclado, estados vazios, feedback)

Nível 100 --- "Completo: regras avançadas, planeador preventivo, personalização"

- **Funcionalidades novas:**
    - RF04 -- Planeamento de Manutenção Preventiva (planos, periodicidades, eventos por horas/uso)
    - RF13 -- Dashboard em Tempo Real (KPIs, cartões, gráficos; atualização reativa)
    - RF10 -- Tabelas de Apoio & Parametrização (prioridades, categorias de falha, centros de custo)
    - RNF14 -- Extensibilidade (pontos de extensão para novos KPIs/estados; config de regras)
- **RNF10:** Qualidade de Código (documentação; convenções)
- **RNF07:** Usabilidade (UI por consola, acessibilidade básica, teclado, estados vazios, feedback)

Abordagem para Elaboração do Projeto

Esta seção sugere uma abordagem estruturada para o desenvolvimento do projeto.

Propõe-se uma abordagem estruturada para se desenvolver o projeto de gestão da manutenção, baseada no percurso por níveis de dificuldade crescente. O objetivo é garantir um desenvolvimento incremental, reduzindo riscos e permitindo foco progressivo na complexidade.

1. Planeamento Inicial

- Leia atentamente todo o enunciado, focando nos objetivos, requisitos funcionais (RF), não-funcionais (RNF), de interface (GUI) e de dados (D).
- Entenda a arquitetura em camadas (UI, domínio, aplicação/serviço, infraestrutura) e os princípios de orientação a objetos (composição, interfaces, polimorfismo).
- Identifique as dependências entre requisitos e planeie a ordem de implementação.

2. Abordagem por Níveis

- Siga o percurso incrementalmente, começando pelo **Nível 50** (MVP - Produto Viável Mínimo básico com consola e ficheiros).
- Implemente apenas os RF, RNF, GUI e D listados para cada nível, validando com testes unitários.
- Avance para o próximo nível apenas após validar o atual com os critérios de aceitação.

3. Estrutura do Projeto

- Use Java e uma ferramenta de build como Maven para gerir dependências e compilação.

- Organize o código em pacotes claros: `ui` (consola/JavaFX), `domínio` (entidades, regras de negócio), `aplicação` (serviços), `infraestrutura` (repositórios, persistência).
- Inicie com persistência em ficheiros de texto/binários (Nível 50), migrando para SQLite no Nível 60.
- Implemente validações, exceções e logs desde o início para garantir robustez.

4. Desenvolvimento Iterativo

- Para cada nível, implemente as funcionalidades novas, integrando com as existentes.
- Priorize a usabilidade: comece com interface consola simples, evoluindo para JavaFX nos níveis superiores.
- Garanta qualidade: documentação, convenções de código, testes e auditoria de dados.

5. Testes e Validação

- Execute testes unitários e de integração em cada nível.
- Valide com os critérios de aceitação (ex.: estados válidos, feedback ao utilizador, performance).
- Realize testes de carga e usabilidade nos níveis avançados.

6. Entrega e Revisão

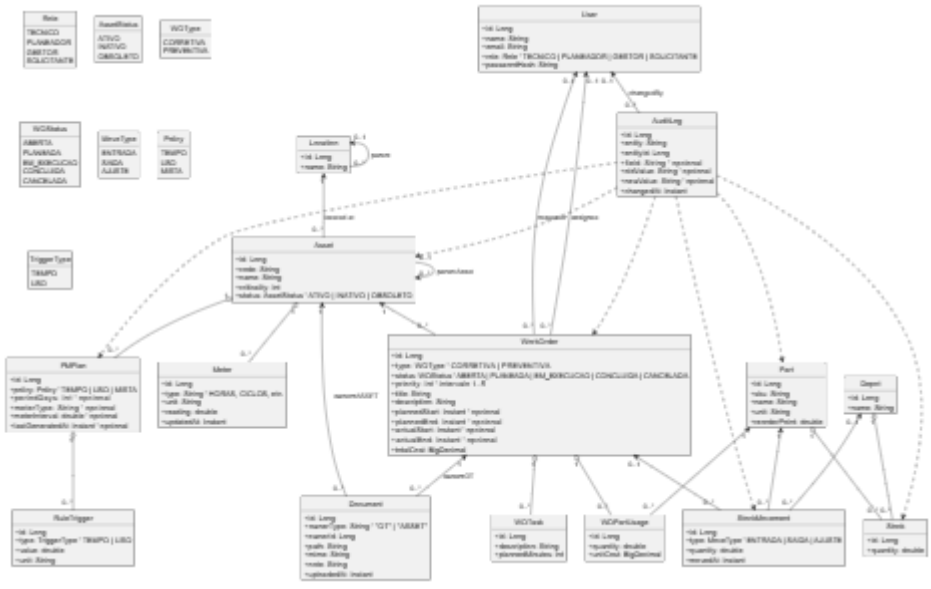
- No **Nível 100**, o sistema deve ser completo, extensível e portável, com dashboard, KPIs e todas as funcionalidades implementadas.
- Faça revisões regulares do código para refatoração e otimização.
- Documente decisões de desenho e lições aprendidas.

Essa abordagem incremental facilita o aprendizagem progressiva e a entrega de um produto funcional em cada etapa. Comece codificando as entidades do domínio e um repositório básico.

Bom trabalho!

Anexos

Diagrama de Classes (Modelo de Domínio)



```
// Esboço do Modelo de Domínio em Java 17+.
// Nota: em Java, cada "top-level type" (class/enum) deve estar no seu próprio ficheirc
```

```java
import java.math.BigDecimal;
import java.time.Instant;
import java.util.ArrayList;
import java.util.List;

// ------------------------
// Enums
// ------------------------

enum Role { TECNICO, PLANEADOR, GESTOR, SOLICITANTE }

enum AssetStatus { ATIVO, INATIVO, OBSOLETO }

enum WOType { CORRETIVA, PREVENTIVA }

enum WOStatus { ABERTA, PLANEADA, EM_EXECUCAO, CONCLUIDA, CANCELADA }

enum MoveType { ENTRADA, SAIDA, AJUSTE }

enum Policy { TEMPO, USO, MISTA }

enum TriggerType { TEMPO, USO }

enum DocumentOwnerType { OT, ASSET }

// ------------------------
// Entidades principais
// ------------------------

final class User {
    Long id;
    String name;
    String email;
    Role role;
    String passwordHash;
}

final class Location {
    Long id;
    String name;
    Location parent; // opcional
}

final class Asset {
    Long id;
    String code;
    String name;
    int criticality; // 0..5
    AssetStatus status;

    Location location; // opcional
    Asset parentAsset; // opcional

    List<Meter> meters = new ArrayList<>();
    List<PMPlan> preventivePlans = new ArrayList<>();
}

final class Meter {
    Long id;
    String type;     // "HORAS", "CICLOS", ...
    String unit;     // "h", "ciclos", ...
    double reading;
    Instant updatedAt;
}

final class WorkOrder {
```

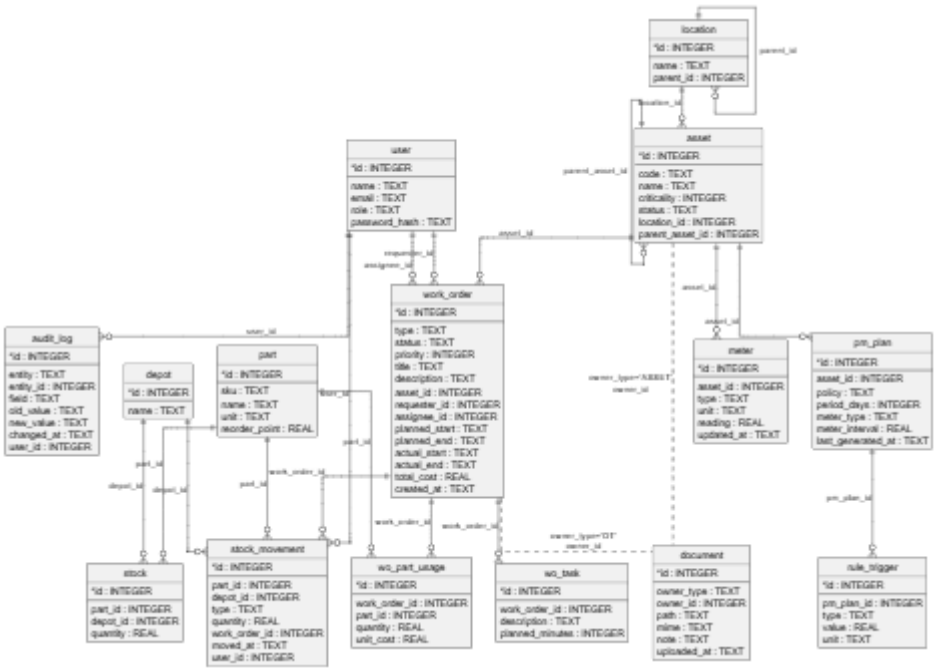
```
    Long id;
    WOType type;
    WOStatus status;
    int priority; // 1..5
    String title;
    String description;

    Asset asset;
    User requester; // opcional
    User assignee;  // opcional

    Instant plannedStart; // opcional
    Instant plannedEnd;   // opcional
    Instant actualStart;  // opcional
    Instant actualEnd;    // opcional

    BigDecimal totalCost = BigDecimal.ZERO;

    List<WOTask> tasks = new ArrayList<>();
    List<WOPartUsage> partUsages = new ArrayList<>();
    List<Document> documents = new ArrayList<>();
}

final class WOTask {
    Long id;
    String description;
    int plannedMinutes;
}

final class Part {
    Long id;
    String sku;
    String name;
    String unit;
    double reorderPoint;
}

final class Depot {
    Long id;
    String name;
}

final class Stock {
    Long id;
    Part part;
    Depot depot;
    double quantity;
}

final class StockMovement {
    Long id;
    Part part;
    Depot depot;       // opcional
    MoveType type;
    double quantity;
    WorkOrder workOrder; // opcional
    User user;           // opcional
    Instant movedAt;
}

final class WOPartUsage {
    Long id;
    Part part;
    double quantity;
    BigDecimal unitCost = BigDecimal.ZERO;
}
```

```
final class PMPlan {
    Long id;
    Asset asset;
    Policy policy;

    Integer periodDays;      // opcional
    String meterType;        // opcional
    Double meterInterval;    // opcional
    Instant lastGeneratedAt; // opcional

    List<RuleTrigger> ruleTriggers = new ArrayList<>();
}

final class RuleTrigger {
    Long id;
    TriggerType type;
    double value;
    String unit;
}

final class Document {
    Long id;
    DocumentOwnerType ownerType;
    Long ownerId;

    String path;
    String mime;
    String note;
    Instant uploadedAt;
}

final class AuditLog {
    Long id;
    String entity;
    Long entityId;
    String field;    // opcional
    String oldValue; // opcional
    String newValue; // opcional
    Instant changedAt;
    User changedBy;  // opcional (coerente com audit_log.user_id)
}
```

Estrutura de dados dos ficheiros/Base de dados

Esta seção descreve a estrutura de dados para persistência. Para o **Nível 50**, a persistência pode ser feita em ficheiros (texto/binário) seguindo o mesmo modelo lógico. A partir do **Nível 60**, recomenda-se SQLite com o esquema abaixo.

- Tabelas marcadas com **(≥70/≥80/≥90)** são incrementos previstos no roadmap.
- O esquema inclui chaves primárias/estrangeiras para refletir as relações do **Modelo de Domínio**.
- As **views/queries de KPIs (D02)** e **triggers leves (D03)** podem ser adicionadas depois do esquema base.



```sql
-- Utilizadores e Perfis
CREATE TABLE user (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  name TEXT NOT NULL,
  email TEXT UNIQUE NOT NULL,
  role TEXT NOT NULL CHECK (role IN ('TECNICO','PLANEADOR','GESTOR','SOLICITANTE')),
  password_hash TEXT NOT NULL
);

-- Localizações
CREATE TABLE location (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  name TEXT NOT NULL,
  parent_id INTEGER,
  FOREIGN KEY (parent_id) REFERENCES location(id) ON DELETE SET NULL
);

-- Ativos
CREATE TABLE asset (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  code TEXT NOT NULL UNIQUE,
  name TEXT NOT NULL,
  criticality INTEGER DEFAULT 0 CHECK (criticality BETWEEN 0 AND 5),
  status TEXT NOT NULL DEFAULT 'ATIVO' CHECK (status IN ('ATIVO','INATIVO','OBSOLETO'))
  location_id INTEGER,
  parent_asset_id INTEGER,
  FOREIGN KEY (location_id) REFERENCES location(id) ON DELETE SET NULL,
  FOREIGN KEY (parent_asset_id) REFERENCES asset(id) ON DELETE SET NULL
);
CREATE INDEX idx_asset_location ON asset(location_id);
CREATE INDEX idx_asset_parent ON asset(parent_asset_id);

-- Contadores (≥70)
CREATE TABLE meter (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  asset_id INTEGER NOT NULL,
  type TEXT NOT NULL,                -- 'HORAS', 'CICLOS', etc.
  unit TEXT NOT NULL,                -- 'h', 'ciclos'
  reading REAL NOT NULL DEFAULT 0,
  updated_at TEXT NOT NULL DEFAULT (datetime('now')),
  FOREIGN KEY (asset_id) REFERENCES asset(id) ON DELETE CASCADE
);
CREATE INDEX idx_meter_asset ON meter(asset_id);
```

```sql
-- Peças e Stocks
CREATE TABLE part (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  sku TEXT NOT NULL UNIQUE,
  name TEXT NOT NULL,
  unit TEXT NOT NULL,
  reorder_point REAL NOT NULL DEFAULT 0
);

-- Depósitos (≥90)
CREATE TABLE depot (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  name TEXT NOT NULL UNIQUE
);

-- Stock por depósito (≥90; para nível 60, usar coluna quantity em part_mov ou tabela s
CREATE TABLE stock (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  part_id INTEGER NOT NULL,
  depot_id INTEGER NOT NULL,
  quantity REAL NOT NULL DEFAULT 0,
  UNIQUE (part_id, depot_id),
  FOREIGN KEY (part_id) REFERENCES part(id) ON DELETE CASCADE,
  FOREIGN KEY (depot_id) REFERENCES depot(id) ON DELETE CASCADE
);

-- Ordens de Trabalho
CREATE TABLE work_order (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  type TEXT NOT NULL CHECK (type IN ('CORRETIVA','PREVENTIVA')),
  status TEXT NOT NULL CHECK (status IN ('ABERTA','PLANEADA','EM_EXECUCAO','CONCLUIDA',
  priority INTEGER NOT NULL CHECK (priority BETWEEN 1 AND 5),
  title TEXT NOT NULL,
  description TEXT,
  asset_id INTEGER NOT NULL,
  requester_id INTEGER,
  assignee_id INTEGER,
  planned_start TEXT,
  planned_end TEXT,
  actual_start TEXT,
  actual_end TEXT,
  total_cost REAL NOT NULL DEFAULT 0,
  created_at TEXT NOT NULL DEFAULT (datetime('now')),
  FOREIGN KEY (asset_id) REFERENCES asset(id),
  FOREIGN KEY (requester_id) REFERENCES user(id),
  FOREIGN KEY (assignee_id) REFERENCES user(id)
);
CREATE INDEX idx_wo_asset ON work_order(asset_id);
CREATE INDEX idx_wo_status ON work_order(status);

-- Tarefas da OT
CREATE TABLE wo_task (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  work_order_id INTEGER NOT NULL,
  description TEXT NOT NULL,
  planned_minutes INTEGER DEFAULT 0,
  FOREIGN KEY (work_order_id) REFERENCES work_order(id) ON DELETE CASCADE
);

-- Consumo de Peças
CREATE TABLE wo_part_usage (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  work_order_id INTEGER NOT NULL,
  part_id INTEGER NOT NULL,
  quantity REAL NOT NULL CHECK (quantity > 0),
  unit_cost REAL NOT NULL DEFAULT 0,
```

```
  FOREIGN KEY (work_order_id) REFERENCES work_order(id) ON DELETE CASCADE,
  FOREIGN KEY (part_id) REFERENCES part(id)
);
CREATE INDEX idx_usage_wo ON wo_part_usage(work_order_id);

-- Plano Preventivo
CREATE TABLE pm_plan (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  asset_id INTEGER NOT NULL,
  policy TEXT NOT NULL CHECK (policy IN ('TEMPO','USO','MISTA')), -- 'MISTA' (≥100)
  period_days INTEGER,           -- usado se TEMPO ou parte temporal
  meter_type TEXT,               -- usado se USO/MISTA
  meter_interval REAL,           -- incremento alvo
  last_generated_at TEXT,
  FOREIGN KEY (asset_id) REFERENCES asset(id) ON DELETE CASCADE
);

-- Regras/Gatilhos do Plano Preventivo (≥70)
-- Coerente com o Modelo de Domínio: PMPlan 1..* RuleTrigger
CREATE TABLE rule_trigger (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  pm_plan_id INTEGER NOT NULL,
  type TEXT NOT NULL CHECK (type IN ('TEMPO','USO')),
  value REAL NOT NULL,
  unit TEXT NOT NULL,
  FOREIGN KEY (pm_plan_id) REFERENCES pm_plan(id) ON DELETE CASCADE
);
CREATE INDEX idx_rule_trigger_plan ON rule_trigger(pm_plan_id);

-- Auditoria (≥70)
CREATE TABLE audit_log (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  entity TEXT NOT NULL,
  entity_id INTEGER NOT NULL,
  field TEXT,
  old_value TEXT,
  new_value TEXT,
  changed_at TEXT NOT NULL DEFAULT (datetime('now')),
  user_id INTEGER,
  FOREIGN KEY (user_id) REFERENCES user(id)
);

-- Documentos / Anexos (≥80)
CREATE TABLE document (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  owner_type TEXT NOT NULL CHECK (owner_type IN ('OT','ASSET')),
  owner_id INTEGER NOT NULL,
  path TEXT NOT NULL,
  mime TEXT,
  note TEXT,
  uploaded_at TEXT NOT NULL DEFAULT (datetime('now'))
);
CREATE INDEX idx_document_owner ON document(owner_type, owner_id);

-- Movimentos de Stock (≥90)
CREATE TABLE stock_movement (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  part_id INTEGER NOT NULL,
  depot_id INTEGER,
  type TEXT NOT NULL CHECK (type IN ('ENTRADA','SAIDA','AJUSTE')),
  quantity REAL NOT NULL,
  work_order_id INTEGER,
  moved_at TEXT NOT NULL DEFAULT (datetime('now')),
  user_id INTEGER,
  FOREIGN KEY (part_id) REFERENCES part(id),
  FOREIGN KEY (depot_id) REFERENCES depot(id),
  FOREIGN KEY (work_order_id) REFERENCES work_order(id),
```

```
  FOREIGN KEY (user_id) REFERENCES user(id)
);
```