RDTrackr: Sistema de Gerenciamento de Estoque para Empresas de Usinagem

João Antonio David

Resumo

O **RDTrackr** é um sistema web de gerenciamento de estoque desenvolvido para empresas de usinagem que enfrentam dificuldades no controle de insumos e ferramentas. O projeto oferece atualização em tempo real, rastreabilidade de movimentações e alertas automáticos. Sua arquitetura é baseada em Django, com Celery, RabbitMQ e Redis, promovendo escalabilidade, desempenho e modularidade.

Sumário

1 Introdução

1.1 Contexto

Empresas do setor de usinagem operam com altos níveis de complexidade no controle de materiais. A falta de visibilidade em tempo real sobre movimentações e saldos compromete a eficiência produtiva. O uso de planilhas ou sistemas genéricos mostra-se limitado diante da especificidade dessas operações.

1.2 Justificativa

Para evitar rupturas na produção, atrasos e desperdícios, é fundamental contar com um sistema que ofereça registro e acompanhamento contínuo do estoque. O RDTrackr surge como solução sob medida para o setor, garantindo controle total, integração e automação.

1.3 Objetivos

Objetivo Geral: Desenvolver um sistema web modular para gerenciamento de estoque, com foco em atualização em tempo real, rastreabilidade e automação de alertas.

Objetivos Específicos:

- Proporcionar uma interface web intuitiva e responsiva;
- Facilitar o acompanhamento em tempo real dos saldos e movimentações;
- Gerar alertas automáticos para reposição de itens críticos;
- Permitir emissão de relatórios por setor, período e movimentação;
- Incorporar dashboards interativos para análise estratégica do estoque.

2 Descrição do Projeto

2.1 Tema

Sistema web de gerenciamento de estoque para empresas de usinagem, com foco em rastreabilidade, automação e escalabilidade.

2.2 Problemas a Resolver

- Falta de controle de estoque em tempo real;
- Ausência de alertas automáticos;
- Dificuldade em rastrear movimentações e responsáveis;
- Falta de interface especializada para o setor de usinagem.

2.3 Limitações

- Integrações com sistemas externos (ERP, financeiro) não fazem parte do MVP;
- Módulo de controle de produção não incluso nesta versão.

3 Especificação Técnica

3.1 Requisitos Funcionais (RF)

Código	Descrição
RF01	Permitir cadastro e edição de itens no estoque.
RF02	Registrar entradas e saídas com origem e destino.
RF03	Consultar saldo atualizado por item/setor.
RF04	Emitir alertas automáticos conforme regras.
RF05	Manter histórico completo de movimentações.
RF06	Interface responsiva para diferentes dispositivos.
RF07	Configurar permissões por tipo de usuário.
RF08	Permitir integração via API REST.

3.2 Requisitos Não Funcionais (RNF)

Código	Descrição
RNF01	Garantir tempo de resposta inferior a 500ms.
RNF02	Utilizar Celery e Redis para atualização assíncrona.
RNF03	Usar Redis para cache de dados críticos.
RNF04	Garantir autenticação segura via JWT.

3.3 Considerações de Design

Padrões de Arquitetura

- **Monólito Modularizado:** o backend será desenvolvido em Django REST Framework, com divisão lógica em módulos de domínio (ex.: estoque, movimentação, alertas).
- **MVC:** o Django adota o padrão Model-View-Controller, organizando a lógica de dados, regras de negócio e endpoints REST de forma separada e coesa.
- **Event-Driven:** o sistema utilizará Celery para processar tarefas assíncronas (como geração de relatórios e envio de notificações), com Redis atuando como broker e backend das filas.

Stack Tecnológica e Justificativas

Backend: Django + Django REST Framework Escolhido pela robustez, maturidade e vasto ecossistema. O DRF facilita a criação de APIs RESTful seguras e escaláveis, suportando autenticação JWT e filtros dinâmicos.

Frontend: React Biblioteca amplamente adotada para Single Page Applications, oferece reatividade e construção de interfaces responsivas, essenciais para dashboards e relatórios dinâmicos.

Banco de Dados: PostgreSQL SGBD relacional open-source, reconhecido pela confiabilidade e por oferecer recursos avançados como transações ACID e extensões para dados geoespaciais (útil para futura rastreabilidade logística).

Cache e Broker: Redis Redis será utilizado tanto para caching de consultas e dados críticos quanto como backend (broker) do Celery, acelerando operações assíncronas e reduzindo latências.

Tarefas Assíncronas: Celery Permite processamento desacoplado e escalável de operações demoradas, como envio de alertas automáticos e cálculos agregados, utilizando Redis como sistema de filas.

Monitoramento: Prometheus e Grafana Para coleta de métricas (CPU, memória, throughput das APIs) e construção de dashboards que forneçam visibilidade em tempo real do ambiente.

Containerização e CI/CD: Docker e GitHub Actions Docker garante consistência entre ambientes locais, homologação e produção. O pipeline CI/CD será implementado com GitHub Actions, permitindo testes automatizados, lint e build dos containers em cada pull request.

Estilo Visual: Tailwind CSS Facilita a criação rápida de interfaces modernas e responsivas, mantendo consistência visual sem necessidade de CSS manual extenso.

4 Diagramas de Casos de Uso

Nesta seção são apresentados os diagramas de casos de uso que representam as principais interações dos atores com o sistema RDTrackr, divididos por áreas funcionais.

4.1 Caso de Uso 1: Processo de Compra

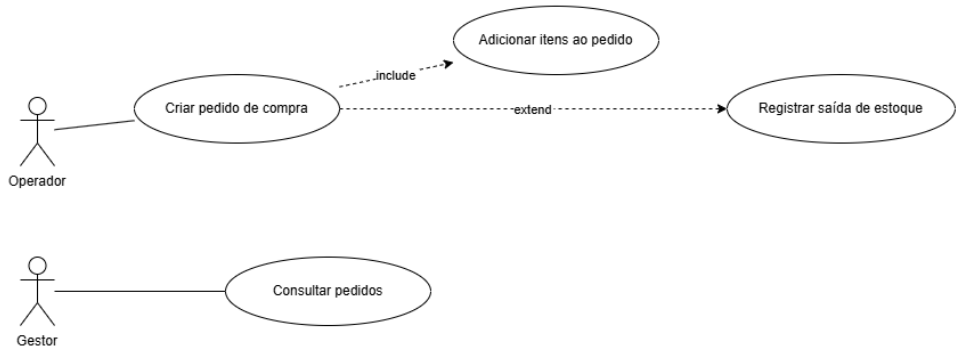


Figure 1: Diagrama de Caso de Uso 1 – Processo de Compra

4.2 Caso de Uso 2: Movimentação e Cadastro de Produtos

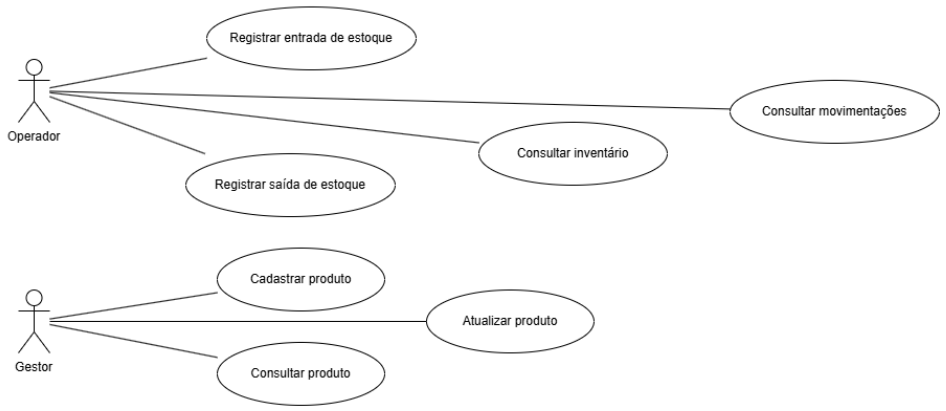


Figure 2: Diagrama de Caso de Uso 2 – Movimentação e Cadastro de Produtos

4.3 Caso de Uso 3: Gestão de Estoque e Alertas

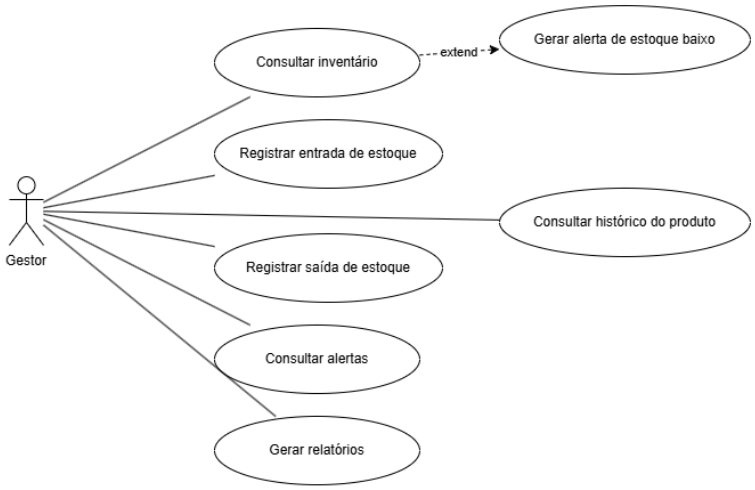
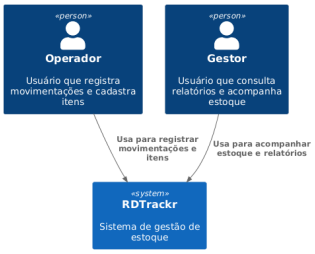
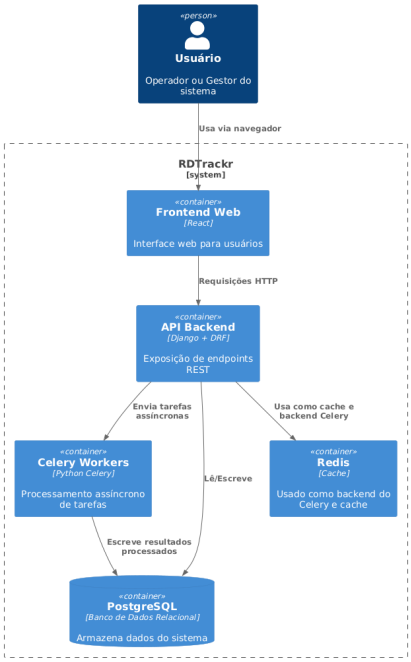
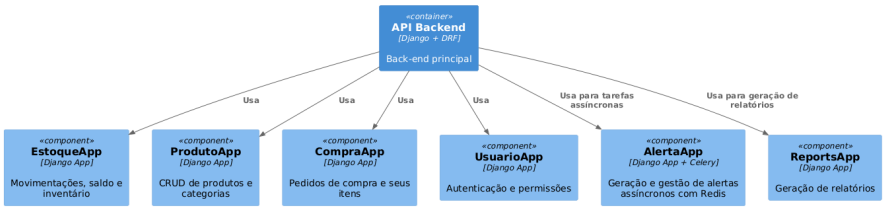


Figure 3: Diagrama de Caso de Uso 3 – Gestão de Estoque e Alertas

Modelos C4


«person»
Operador
Usuário que registra movimentações e cadastra itens
«person»
Gestor
Usuário que consulta relatórios e acompanha estoque
Usa para registrar movimentações e itens
Usa para acompanhar estoque e relatórios
«system»
RDTrackr
Sistema de gestão de estoque


«person»
Usuário
Operador ou Gestor do sistema
Usa via navegador
RDTrackr
[system]
«container»
Frontend Web
[React]
Interface web para usuários
Requisições HTTP
«container»
API Backend
[Django + DRF]
Exposição de endpoints REST
Envia tarefas assíncronas
Usa como cache e backend Celery
«container»
Celery Workers
[Python Celery]
Processamento assíncrono de tarefas
Lê/Escreve
«container»
Redis
[Cache]
Usado como backend do Celery e cache
Escreve resultados processados
«container»
PostgreSQL
[Banco de Dados Relacional]
Armazena dados do sistema


«container»
API Backend
[Django + DRF]
Back-end principal
Usa
Usa
Usa
Usa
Usa para tarefas assíncronas
Usa para geração de relatórios
«component»
EstoqueApp
[Django App]
Movimentações, saldo e inventário
«component»
ProdutoApp
[Django App]
CRUD de produtos e categorias
«component»
CompraApp
[Django App]
Pedidos de compra e seus itens
«component»
UsuarioApp
[Django App]
Autenticação e permissões
«component»
AlertaApp
[Django App + Celery]
Geração e gestão de alertas assíncronos com Redis
«component»
ReportsApp
[Django App]
Geração de relatórios

5 Stack Tecnológica

- **Linguagens:** Python, JavaScript
- **Backend:** Django, DRF
- **Frontend:** React, Tailwind CSS
- **Tarefas:** Celery
- **Cache:** Redis
- **Banco:** PostgreSQL
- **Monitoramento:** Prometheus, Grafana, Loguru
- **CI/CD:** GitHub Actions, Docker

6 Considerações de Segurança

Para garantir a integridade, confidencialidade e disponibilidade dos dados manipulados pelo sistema RDTrackr, foram adotadas práticas consolidadas de segurança em diferentes camadas do projeto.

- **Comunicação segura via HTTPS**: Todo o tráfego entre cliente e servidor será criptografado utilizando o protocolo HTTPS (TLS/SSL), prevenindo ataques do tipo man-in-the-middle e protegendo credenciais e dados sensíveis transmitidos.
- **Autenticação JWT com RBAC**: A autenticação será baseada em JSON Web Tokens (JWT), permitindo um fluxo stateless, escalável e seguro. O controle de acesso utilizará RBAC (Role-Based Access Control), garantindo que cada usuário tenha permissões estritamente alinhadas ao seu papel na organização.
- **Logs auditáveis e estruturados**: O sistema manterá registros detalhados de operações críticas, incluindo login, alterações de estoque e geração de relatórios. Os logs seguirão um formato estruturado (ex: JSON), facilitando análises e auditorias de conformidade.
- **Validação e sanitização de dados**: Todas as entradas recebidas pelas APIs passarão por validações rigorosas e sanitização, prevenindo injeções de SQL ou scripts maliciosos (XSS). Isso protege tanto a aplicação quanto o banco de dados contra ataques comuns.

7 Próximos Passos

Validação da Proposta

Apresentar a documentação técnica, arquitetura e requisitos do RDTrackr ao orientador e banca avaliadora para garantir que a proposta esteja em conformidade com os objetivos acadêmicos e alinhada às necessidades reais do mercado de usinagem.

Revisões e Refinamentos

Realizar revisões contínuas do documento e da modelagem do sistema com base no feedback recebido, ajustando requisitos, fluxos e diagramas. Aperfeiçoar a clareza e a objetividade das especificações técnicas para consolidar uma base sólida para a fase de desenvolvimento.

Aprovação Formal

Submeter a documentação final revisada ao orientador e demais professores para obtenção da aprovação formal, validando que todos os critérios acadêmicos do Portfólio I foram devidamente cumpridos.

Planejamento para Implementação (Portfólio II)

Elaborar o cronograma detalhado para o desenvolvimento do sistema na etapa do Portfólio II, definindo sprints, entregas intermediárias e milestones de acompanhamento. Incluir o planejamento para testes automatizados, integração contínua e homologação em ambiente controlado.

Preparação do Ambiente Técnico

Configurar o pipeline de CI/CD, os ambientes de desenvolvimento (local e homologação) e garantir a containerização via Docker, visando padronização do deploy e mitigação de problemas de ambiente.

Engajamento com Stakeholders

Promover reuniões com stakeholders (gestores da empresa de usinagem) para alinhar expectativas quanto a funcionalidades prioritárias, customizações e indicadores de sucesso do sistema.

Desenvolvimento

- Implementar funcionalidades principais do MVP
- Realizar testes automatizados e de integração.
- Implantar ambiente de homologação.
- Obter feedback de usuários reais.
- Ajustar funcionalidades conforme retorno obtido.

8 Referências

Frameworks e Bibliotecas

- Django: Framework web Python para aplicações rápidas e seguras.
- Django REST Framework: Extensão para construção de APIs REST robustas.
- React.js: Biblioteca para interfaces de usuário dinâmicas e reativas.
- Tailwind CSS: Framework CSS utilitário para design responsivo.
- Celery: Framework para processamento assíncrono e filas de tarefas distribuídas.
- Redis: Armazenamento em memória para cache e broker de filas.
- JWT: Autenticação e autorização baseadas em tokens JSON Web Token.

Ferramentas de Desenvolvimento e Gestão

- GitHub Actions: CI/CD para automação de testes, builds e deploy.
- Docker: Containerização para garantir ambientes consistentes.
- Prometheus: Coleta de métricas e monitoramento de aplicações.
- Grafana: Dashboards e visualização de métricas em tempo real.
- VS Code: Editor de código usado no desenvolvimento.
- Postman: Testes de API REST durante o desenvolvimento.

Documentação e Artigos

- Django Docs: Para detalhes de configuração e boas práticas do framework.
- DRF Quickstart: Guia para construção de APIs REST.
- React Learn: Documentação oficial e guias de melhores práticas.
- Tailwind Docs: Guia oficial do Tailwind CSS.
- Celery Docs: Documentação para filas e tarefas assíncronas.

CLAUDINEI DIAS

EDICARSIA BARBIERO PILLON

GLAUCO VINICIUS SCHEFFEL