📦 NonConformity - Sistema de Gestão de Não Conformidades

Bem-vindo ao **NonConformity**, um sistema completo para o gerenciamento de não conformidades de recebimento (NCR), com foco na rastreabilidade, controle e comunicação eficiente entre as áreas envolvidas.
Criado por **Diego Bernardes**, este projeto visa otimizar o tratamento de não conformidades, garantindo a qualidade e a conformidade dos processos.

✨ Funcionalidades Principais

NCR - Gestão de Não Conformidades de Recebimento

✅ Criação, Leitura, Atualização e Exclusão (CRUD) de NCRs.
✅ Detalhamento completo: Nº Pedido, Cód. Material, Nota Fiscal, Datas, Descrição, Motivo, Área Responsável, Fornecedor, Comprador, Status, Observações.
✅ Cálculo e exibição de "dias em aberto" para rastreamento.
✅ Associação de NCR a um usuário criador e um comprador responsável.
✅ Upload, visualização e gerenciamento de arquivos anexos às NCRs.

👮 Controle de Acesso e Perfis

✅ Três perfis de usuário: `Almoxarifado` (cria NCRs), `Comprador` (trata NCRs) e `Admin` (controle total).
✅ Autenticação segura por nome de usuário e senha.
✅ Registro de novos usuários restrito a Administradores.
✅ Autorização granular baseada no perfil do usuário e na propriedade da NCR.

⚙️ Fluxo da NCR

✅ Criação de NCRs por usuários do Almoxarifado ou Admin.
✅ Atribuição da NCR a um Comprador para tratamento e resolução.
✅ Atualização de status e informações da NCR pelo Comprador.

📧 Notificações

✅ Notificação por email aos Compradores quando uma nova NCR lhes é atribuída.

🛡️ Segurança

✅ Armazenamento seguro de senhas utilizando hashing.
✅ Proteção contra ataques CSRF em formulários web.
✅ Chave secreta (`SECRET_KEY`) para segurança das sessões.

🔗 API REST para Não Conformidades

✅ Endpoints para CRUD de NCRs.
✅ Paginação, filtros e ordenação de resultados.
✅ Autenticação e autorização para acesso seguro à API.
✅ Documentação da API com Swagger/OpenAPI.

🛠️ Validação e Tratamento de Erros

✅ Validação de dados no backend e frontend.
✅ Mensagens de erro claras e personalizadas em português.
✅ Páginas de erro customizadas para 404 e 500.
✅ Sistema de logging para eventos e erros da aplicação.

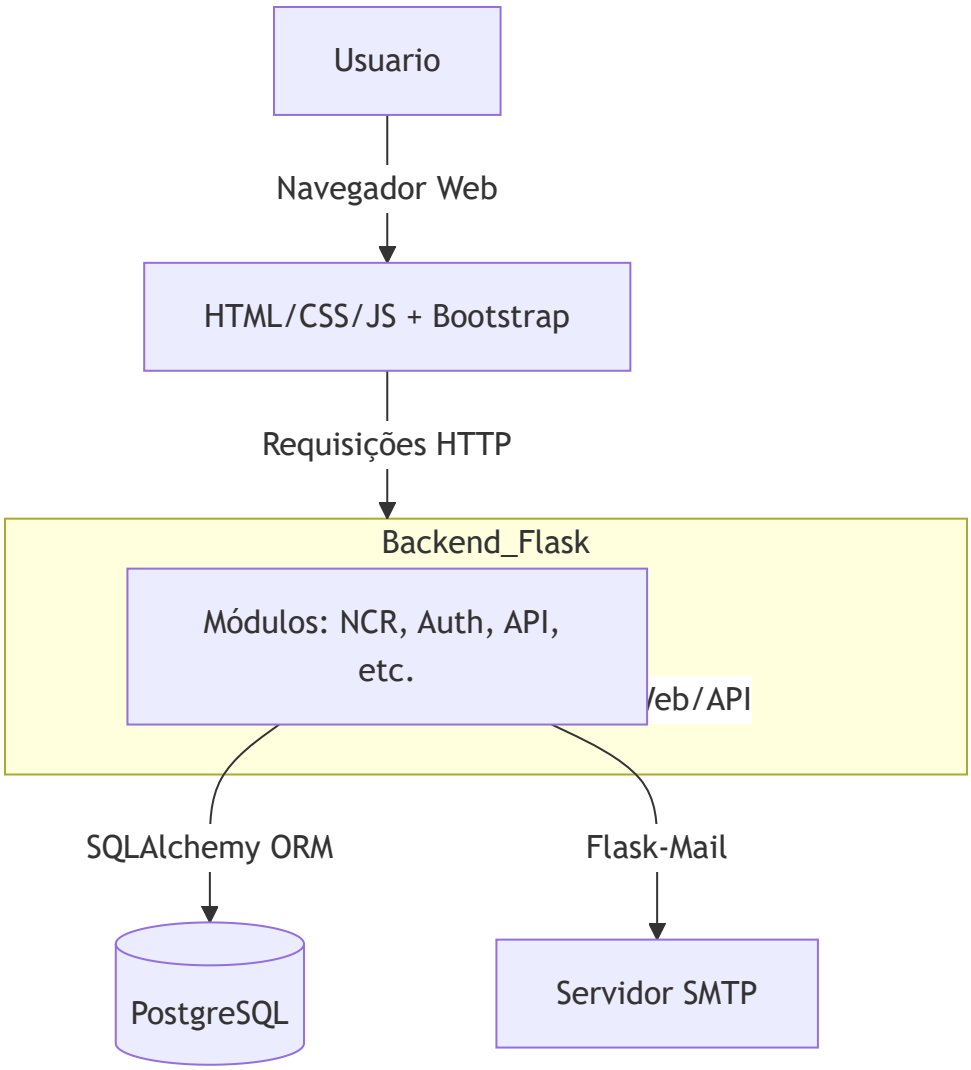
💻 Tecnologias Utilizadas

Categoria	Tecnologias
Backend	Python, Flask, Flask-SQLAlchemy, Flask-Migrate, Flask-Login, Flask-Mail, Flask-WTF
Frontend	HTML, CSS, Bootstrap 5, JavaScript
Banco Dados	PostgreSQL, SQLAlchemy ORM, Alembic
DevOps	Docker (Conteinerização), GitLab CI (CI/CD), Kubernetes (Orquestração de Contêineres)

📌 Pré-requisitos

- Python 3.11+ (Verificar `requirements.txt` para versões exatas dos pacotes Flask, etc.)
- PostgreSQL
- Navegador moderno (Chrome, Firefox, Edge)

🧩 Arquitetura do Sistema (Simplificada)



🚀 Instalação

1. **Clone o repositório** (ajuste a URL se for diferente)

```
git clone https://github.com/seu_usuario/NonConformity.git
cd NonConformity
```

2. **Crie e ative um ambiente virtual** (recomendado)

```
python -m venv .venv
source .venv/bin/activate  # No Linux/macOS
# .venv\Scripts\activate    # No Windows
```

3. **Instale as dependências**

```
pip install -r requirements.txt
```

4. **Configure as variáveis de ambiente**
   Crie um arquivo `.env` na raiz do projeto (baseado em um `.env.example` se existir) com as seguintes variáveis:

```
FLASK_APP=main.py
FLASK_ENV=development # ou production
SECRET_KEY=sua_chave_secreta_super_segura
DATABASE_URL=postgresql://usuario_db:senha_db@host_db:porta_db/nome_banco_nonconformity

# Configurações de Email (para notificações)
MAIL_SERVER=smtp.example.com
MAIL_PORT=587
MAIL_USE_TLS=True
MAIL_USERNAME=seu_email@example.com
MAIL_PASSWORD=sua_senha_email
MAIL_DEFAULT_SENDER=('Seu Nome ou Nome App', 'noreply@example.com')
ADMIN_EMAIL=admin@example.com # Email do administrador para alguns alertas
```

   Adapte `DATABASE_URL` e as configurações `MAIL_*` conforme seu ambiente.

5. **Inicialize e atualize o banco de dados**

```
flask db init  # Apenas na primeira vez, se a pasta migrations não existir
flask db migrate -m "Initial migration for NonConformity" # Ou uma mensagem descritiva
flask db upgrade
```

6. **Crie o primeiro usuário administrador**
   Execute o script fornecido (verifique o nome exato e como executar, pode precisar de `python criar_admin.py` ou `flask run-script criar_admin`):

```
python criar_admin.py
```

   Siga as instruções do script.

7. **Execute o servidor de desenvolvimento Flask**

```
flask run
```

Acesse a aplicação em: `http://127.0.0.1:5000` (ou a porta configurada).

📂 Estrutura do Projeto

```
NonConformity/
├── app/                  # Código fonte da aplicação
│   ├── api/              # Módulos da API REST (endpoints, lógica)
│   ├── auth/             # Módulos de autenticação (rotas, formulários, lógica)
│   ├── core/             # Lógica de negócio central, modelos de dados, validações
│   ├── main/             # Rotas principais da aplicação e blueprints
│   ├── services/         # Lógica de serviços específicos (ex: notificações, integrações)
│   ├── static/           # Arquivos estáticos (CSS, JavaScript, imagens)
│   ├── templates/        # Templates HTML (Jinja2)
│   │   ├── api/          # Documentação da API (Swagger UI)
│   │   ├── auth/         # Templates de autenticação
│   │   ├── errors/       # Páginas de erro personalizadas
│   │   ├── ncr/          # Templates específicos para a gestão de NCRs
│   │   └── ...           # Outros templates (base, etc.)
│   ├── ui/               # Módulos relacionados a componentes de interface ou lógica de fronter
│   └── __init__.py       # Inicializador do aplicativo Flask e blueprints
├── kubernetes/           # Configurações e manifestos para deployment da aplicação com Kubernet
├── logs/                 # Arquivos de log da aplicação
├── migrations/           # Scripts de migração do banco de dados (Alembic)
├── tests/                # Testes automatizados (unitários, integração)
│   ├── integration/
│   └── unit/
├── .dockerignore         # Arquivos e pastas a serem ignorados pelo build Docker
├── .env.example          # Arquivo de exemplo para variáveis de ambiente
├── .gitignore            # Arquivos e pastas a serem ignorados pelo Git
├── .gitlab-ci.yml        # Configuração de Integração Contínua/Entrega Contínua para GitLab
├── config.py             # Configurações da aplicação (carrega do .env)
├── criar_admin.py        # Script para criar o usuário administrador inicial
├── Dockerfile            # Define a imagem Docker para a aplicação
├── main.py               # Ponto de entrada principal da aplicação Flask
├── requirements.txt      # Dependências Python do projeto
├── Roadmap.md            # Documento com o planejamento e progresso do projeto
└── README.md             # Este arquivo
```

🚀 Deployment

A aplicação é continuamente integrada e implantada utilizando GitLab CI/CD. O deployment no ambiente de produção (ou staging) é realizado em um cluster Kubernetes, utilizando os manifestos localizados no diretório `kubernetes/`. O `Dockerfile` na raiz do projeto é usado para construir a imagem da aplicação que é orquestrada pelo Kubernetes.

🐋 Executando com Docker

Um `Dockerfile` está incluído no projeto para facilitar a conteinerização da aplicação.
As instruções abaixo são um exemplo e podem precisar de adaptação ao seu ambiente específico.

Exemplo (baseado no `READMEmodel.md`, precisará de adaptação para `NonConformity`):

```
# Criar network (se necessário)
# docker network create net_nonconformity

# Rodar container do PostgreSQL
# docker run -d --name db_nonconformity --network net_nonconformity \
#   -e POSTGRES_USER=nonconformity_user \
#   -e POSTGRES_PASSWORD=nonconformity_pass \
#   -e POSTGRES_DB=nonconformity_db \
#   -p 5433:5432 postgres:16 # Mapear para porta diferente se 5432 estiver em uso

# Rodar container da aplicação (construir Dockerfile antes)
# docker build -t nonconformity_app .
# docker run -d -p 5000:5000 --network net_nonconformity \
#   -e DATABASE_URL=postgresql://nonconformity_user:nonconformity_pass@db_nonconformity:5432/nor
#   -e SECRET_KEY=sua_chave_secreta_no_docker \
#   # Adicionar outras variáveis de ambiente necessárias (MAIL_*, etc.)
#   --name app_nonconformity nonconformity_app
```

📜 Licença

Este projeto está licenciado sob a licença Apache License 2.0.

💡 Autor

👨‍💻 **Diego Bernardes Silva**
🐙 GitHub: diegobernardessv
🦊 GitLab: diegobernardessv
💼 LinkedIn: diegobernardessv

🔮 Melhorias Futuras (Principais Itens do Roadmap)

- **Testes Abrangentes (Prioridade Alta):**
  - [ ] Testes Unitários: Cobrir modelos, validadores, lógica de permissões e transições de status.
  - [ ] Testes de Integração: Expandir para rotas web, formulários e interações com o banco.
  - [ ] Testes de Sistema (End-to-End): Simular fluxos completos do usuário.
- **Dashboard e Relatórios (Pós-Estabilização):**
  - [ ] Desenvolvimento de um dashboard com métricas e estatísticas sobre NCRs.
  - [ ] Implementação de um sistema de geração de relatórios customizáveis.
- **Histórico de Alterações (Pós-Estabilização):**
  - [ ] Criação de um histórico de alterações para cada NCR (log de auditoria).

(Consulte o `Roadmap.md` para uma lista mais detalhada e o status atualizado dos próximos passos.)