Projeto Secretaria
Documento do Projeto

Integrantes: Raick, Henrique, João Gabriel e Rafael

Version: 5.0 10/11/2025

Sumário

1. Introdução

O **SkyPen** é um **sistema web Full-Stack (React + Django) com autenticação JWT e painel administrativo** voltado à **gestão escolar** integrada, desenvolvido para otimizar os processos administrativos e acadêmicos de instituições de ensino. A plataforma permite o cadastro e gerenciamento de alunos, responsáveis, professores, notas e empréstimos de livros, reunindo todas as informações em um ambiente digital seguro e acessível. Além de apoiar o trabalho da secretaria escolar, o sistema oferece acesso personalizado aos alunos, permitindo que acompanhem suas notas, desempenho e histórico de empréstimos. Dessa forma, o **SkyPen** promove agilidade, transparência e organização no controle das atividades escolares, contribuindo para a modernização e digitalização da rotina institucional. O público-alvo são escolas e instituições de ensino que desejam informatizar seus processos, reduzindo o uso de papel, minimizando erros manuais e aprimorando a comunicação entre secretaria, professores e alunos.

---

2. Objetivos

2.1. Objetivo Geral

Desenvolver uma aplicação web que gerencie alunos, professores, notas e empréstimos de livros, com diferentes níveis de acesso e uma interface prática e acessível.

2.2. Objetivos Específicos

- Cadastrar e gerenciar alunos, responsáveis e professores.
- Registrar e consultar notas dos alunos.
- Controlar empréstimos e devoluções de livros.
- Permitir acesso diferenciado conforme o tipo de usuário (admin, funcionário ou aluno).
- Garantir segurança e integridade dos dados.

---

3. Requisitos do Sistema

3.1. Requisitos Funcionais

- RF01: Cadastro e login de usuários com diferentes níveis de acesso.
- RF02: Cadastro, edição e exclusão de alunos, professores e responsáveis.
- RF03: Registro e consulta de notas.
- RF04: Controle de empréstimos e devoluções de livros.
- RF05: Acesso restrito para alunos visualizarem suas próprias informações.
- RF06: Relatórios de desempenho e controle de livros.

3.2. Requisitos Não Funcionais

- RNF01: Interface responsiva (desktop e mobile).
- RNF02: Backend em Django com API RESTful.
- RNF03: Banco de dados relacional (MySQL).
- RNF04: Armazenamento seguro de senhas.
- RNF05: Código e documentação padronizados.

- RNF06: Tempo de resposta médio de até 3 segundos.

---

4. Tecnologias

- Frontend: JavaScript, JSX, CSS, Vite
- Backend: Python, Django
- Banco de Dados: MySQL
- Controle de Versão: GitHub
- Bibliotecas: Axios (integração API), React Router (navegação)

---

5. Arquitetura do Sistema

Arquitetura em camadas, modular e preparada para escala:

- Frontend (React + Vite)
    - SPA com roteamento via React Router.
    - Comunicação com API via Axios; arquitetura de componentes (ui / domain / pages).
    - Estado: React Query ou Redux Toolkit para cache e sincronização.
    - Assets e uploads tratados por backend ou diretamente em S3 (pré-signed URLs).
- API (Django + Django REST Framework)
    - Endpoints RESTful organizados por recursos (users, alunos, notas, livros, emprestimos).
    - Autenticação/Autorização: JWT (djangorestframework-simplejwt) ou tokens com HttpOnly cookies.

    - Camadas: serializers → services (lógica de negócio) → repositories (querysets/ORM).
    - Background jobs com Celery + Redis para tarefas assíncronas (notificações, envio de e-mail, geração de relatórios).
- Banco de Dados (MySQL)
    - Modelos normalizados: Usuário (com papel), Aluno, Responsável, Professor, Disciplina, Nota, Livro, Emprestimo, Auditoria/Log.
    - Índices em campos de busca (email, matrícula) e foreign keys para integridade.
    - Backups incrementais e rotações (daily full + hourly incremental).
- Cache / Mensageria
    - Redis: cache de consultas frequentes, filas Celery e rate-limiting.
    - Opcional: ElasticSearch para buscas avançadas (relatórios, histórico).
- Infra / Deploy
    - Containerização: Docker para frontend, backend e worker.
    - Gunicorn/Uvicorn + Nginx (proxy reverso, TLS termination).
    - CI/CD: GitHub Actions (test → build → deploy). Deploy em VPS, Kubernetes ou PaaS.
    - Observability: logs centralizados (ELK/Graylog), métricas (Prometheus/Grafana), erros (Sentry).
- Segurança e Rede
    - TLS obrigatório, WAF opcional, regras de firewall para portas não públicas.
    - Segregação de ambientes: dev / staging / production.

---

6. Divisão de Tarefas

- Backend (Rafael): Configuração do servidor Django, rotas, autenticação e permissões.
- Frontend (João Gabriel): Criação das telas, formulários e integração com API.
- Banco de Dados (Raick): Modelagem, migrações e relacionamentos.
- Documentação (Henrique): Elaboração e atualização dos documentos do projeto.
  Data de término: 12/11/2025

---

7. Cronograma de Desenvolvimento

Etapa	Responsável	Data de Entrega
Backend Django	Rafael	12/11/2025
Frontend React	João Gabriel	12/11/2025
Banco de Dados	Raick	12/11/2025
Documentação	Henrique	12/11/2025

---

8. Passo a Passo Técnico

Arquitetura em camadas, modular e preparada para escala:

- Frontend (React + Vite)
    - SPA com roteamento via React Router.
    - Comunicação com API via Axios; arquitetura de componentes (ui / domain / pages).
    - Estado: React Query ou Redux Toolkit para cache e sincronização.
    - Assets e uploads tratados por backend ou diretamente em S3 (pré-signed URLs).
- API (Django + Django REST Framework)

    - Endpoints RESTful organizados por recursos (users, alunos, notas, livros, emprestimos).
    - Autenticação/Autorização: JWT (djangorestframework-simplejwt) ou tokens com HttpOnly cookies.
    - Camadas: serializers → services (lógica de negócio) → repositories (querysets/ORM).
    - Background jobs com Celery + Redis para tarefas assíncronas (notificações, envio de e-mail, geração de relatórios).
- Banco de Dados (MySQL)
    - Modelos normalizados: Usuário (com papel), Aluno, Responsável, Professor, Disciplina, Nota, Livro, Emprestimo, Auditoria/Log.
    - Índices em campos de busca (email, matrícula) e foreign keys para integridade.
    - Backups incrementais e rotações (daily full + hourly incremental).
- Cache / Mensageria
    - Redis: cache de consultas frequentes, filas Celery e rate-limiting.
    - Opcional: ElasticSearch para buscas avançadas (relatórios, histórico).
- Infra / Deploy
    - Containerização: Docker para frontend, backend e worker.
    - Gunicorn/Uvicorn + Nginx (proxy reverso, TLS termination).
    - CI/CD: GitHub Actions (test → build → deploy). Deploy em VPS, Kubernetes ou PaaS.
    - Observability: logs centralizados (ELK/Graylog), métricas (Prometheus/Grafana), erros (Sentry).
- Segurança e Rede
    - TLS obrigatório, WAF opcional, regras de firewall para portas não públicas.
    - Segregação de ambientes: dev / staging / production.

---

9. Autenticação

Estratégia recomendada

- **Primary option (mais seguro):** Autenticação com **HttpOnly, Secure cookies** (session-like) + CSRF. Backend gera JWT de acesso e refresh, mas armazena em cookies HttpOnly; o frontend usa chamadas fetch/Axios enviando cookies.

- **Alternative:** JWT em Authorization header (Bearer) — mais simples para APIs públicas, porém expõe risco se token for armazenado em localStorage.

Fluxo de autenticação (exemplo com JWT + refresh)

1. Usuário envia POST /api/auth/login com credenciais sobre HTTPS.

2. Backend valida credenciais; retorna:

    - cookie HttpOnly access_token (expira em 15m)

    - cookie HttpOnly refresh_token (expira em 7d)

    - (opcional) CSRF token em cookie não HttpOnly

3. Requisições subsequentes usam cookies automaticamente; backend valida token.

4. Quando access_token expira, frontend chama POST /api/auth/refresh (ou o middleware intercepta) para obter novo access token usando refresh_token.

5. Logout: POST /api/auth/logout invalida/rotaciona refresh token (persistir tokens em DB para blacklist/rotacionamento).

Boas práticas

- Token short-lived (ex.: 10–15 minutos) + refresh.

- Rotacionar refresh tokens e manter blacklist/allowlist.

- Rejeitar refresh token se houver alteração de senha recente.

- Limitar número de dispositivos/refresh tokens por usuário.

10. Segurança

Criptografia e senhas

- Use *Argon2* (preferido) ou PBKDF2 com salt para hashing de senhas (Django suporta Argon2).
- Armazenar segredos via environment variables ou secret manager (AWS Secrets Manager, HashiCorp Vault).

Transporte e headers

- TLS v1.2+ obrigatório.
- Configurar headers: Strict-Transport-Security, Content-Security-Policy, X-Frame-Options: DENY, X-Content-Type-Options: nosniff, Referrer-Policy.

Cookies e CSRF

- Cookies: HttpOnly, Secure, SameSite=Strict ou Lax conforme UX.
- CSRF protection habilitada para endpoints state-changing (Django já fornece CSRF middleware).
- Para APIs, considere usar double-submit cookie ou CSRF token em header.

Validação e prevenção de injeção

- Use ORM (Django ORM) para evitar SQL injection; paremetrize consultas quando usar raw SQL.
- Sanitize e validar todos os inputs (backend sempre confia menos que frontend).
- Proteja contra XSS escapando dados exibidos no frontend; usar templates seguros.

Segurança de APIs

- Rate limiting (e.g., Django Ratelimit/DRF throttling) por IP/usuário para endpoints sensíveis (login, refresh).
- Bloqueio temporário após N tentativas de login (e-mail de notificação).

- Verificação de dependências (dependabot, Snyk) e atualização periódica.

Logs e auditoria

- Registrar mudanças críticas e acessos (quem, o quê, quando). Não gravar senhas ou tokens em logs.
- Enviar erros para Sentry; logs estruturados para análise.

Backup e recuperação

- Backups diários + testes regulares de restore.
- Política de retenção e encriptação de backups.

---

11. Testes e Qualidade

Estratégia de testes

- **Backend**
  - Unit tests: models, serializers, serviços (pytest + pytest-django).
  - Integration tests: endpoints (client DRF) e fluxos críticos (login, registro de nota, empréstimo).
  - Testes de segurança: autenticação, permissões, CSRF, inputs maliciosos.
- **Frontend**
  - Unit/component tests: React Testing Library + Jest.
  - E2E tests: Cypress para validar fluxos reais (login, CRUD aluno, empréstimo).
- **Cobertura e métricas**
  - Meta de cobertura mínima (ex.: 70–80%); focar em lógica crítica.
  - Executar cobertura em CI; bloquear merge se regressões críticas.

Qualidade de código

- Linters e formatters: ESLint + Prettier (frontend), flake8/black/isort (backend).
- Type checking: TypeScript para frontend (opcional) ou PropTypes; mypy para backend (opcional).
- Pull Request: code review obrigatório e checklist (testes, lint, doc).

CI/CD

- Pipeline recomendado:
    1. Lint + Static Analysis
    2. Run unit tests
    3. Build frontend assets
    4. Run integration/e2e in staging (opcional)
    5. Deploy to staging/prod (manual approval para produção)
- Notificar falhas no canal da equipe (Slack, Teams).

Test data e fixtures

- Fixtures realistas e factories (factory_boy) para testes unitários.
- Ambiente isolado para testes (DB ephemeral, Redis mock).

---

12. Deploy

O processo de deploy do sistema SkyPen é totalmente automatizado e integrado via CI/CD (Continuous Integration / Continuous Deployment), garantindo entregas ágeis, seguras e rastreáveis em todos os ambientes.

12.1. Infraestrutura e Serviços

- Backend: hospedado na Railway, com build automatizado via GitHub Actions.

- Frontend: hospedado na Vercel, integrado ao repositório GitHub para auto-deploy a cada push na branch principal.

- Banco de Dados: PlanetScale (MySQL escalável), com controle de branches, backups automáticos e migrações seguras.

- Armazenamento de Mídia (opcional): Amazon S3 ou equivalente, para uploads de imagens e documentos.

12.2. Pipeline de CI/CD

O processo de integração contínua é orquestrado pelo GitHub Actions, dividido em estágios:

1. Build e Lint: validação de dependências, análise de código e verificação de padrões (flake8, ESLint, Prettier).

2. Testes: execução de testes unitários e de integração (pytest / Jest / Cypress).

3. Build Final: geração dos artefatos do frontend (Vite) e do backend (Django collectstatic + migrations).

4. Deploy Automático:

    - Frontend: publicado automaticamente na Vercel.

    - Backend: publicado na Railway após aprovação.

    - Banco: migrações aplicadas automaticamente via Railway Deploy Hooks ou PlanetScale CLI.

12.3. Variáveis de Ambiente e Configurações

Todas as credenciais sensíveis são armazenadas em variáveis de ambiente protegidas:

- SECRET_KEY, DATABASE_URL, ALLOWED_HOSTS, JWT_SECRET, EMAIL_HOST_USER, EMAIL_HOST_PASSWORD.

- Gerenciamento via Railway Secrets e Vercel Environment Variables.

- As variáveis são versionadas apenas em ambiente de desenvolvimento seguro.

12.4. Monitoramento e Observabilidade

Para garantir estabilidade e performance, o sistema utiliza ferramentas de monitoramento:

- Logs: Railway e Vercel Logs centralizados e exportáveis para Grafana ou ELK Stack.

- Métricas: Prometheus + Grafana (ou Railway Metrics) para CPU, memória e latência.

- Erros: Integração com Sentry para rastrear exceções em tempo real.
- Uptime Monitoring: UptimeRobot ou Pingdom para verificação de disponibilidade 24/7.

12.5. Rollback e Recuperação

- Rollback automatizado: disponível tanto na Vercel quanto na Railway (versões anteriores mantidas por build).
- Banco de dados: PlanetScale permite rollback via deploy requests e snapshots automáticos.
- Backup:
    - Banco: backups incrementais diários e retenção de 7 dias.
    - Mídia: replicação em bucket secundário (S3 versãoing).
- Recuperação: scripts restore.sh documentados e testados em ambiente staging.

12.6. Segurança em Produção

- HTTPS obrigatório (TLS v1.2+).
- Headers de segurança configurados (CSP, HSTS, X-Frame-Options, etc.).
- Rotação de segredos a cada 90 dias.
- Deploys somente via CI/CD (sem acesso manual a produção).
- Logs de auditoria de build e acessos administrativos.

12.7. Ambientes

- Development: ambiente local, usado para testes e homologação.
- Staging: ambiente intermediário (opcional), espelhando produção.
- Production: ambiente final, com domínio público, HTTPS e cache otimizado.