**DOCUMENTAÇÃO TÉCNICA E FUNCIONAL DO SISTEMA CONDOFÁCIL**

**1. Introdução**

O Condofácil é um sistema de gestão de condomínios no modelo SaaS (Software como Serviço) que visa oferecer uma solução completa e diferenciada para a administração de condomínios de diversas arquiteturas, incluindo edifícios com mais de 15 andares, moradias geminadas, isoladas e mistas. O sistema foi desenvolvido com as melhores práticas de desenvolvimento, utilizando Next.js no frontend e Django Ninja e Django REST Framework no backend para a construção de uma API robusta e escalável. O banco de dados utilizado é o MySQL ou PostgreSQL, com suporte para upload de arquivos como comprovativos de pagamentos, faturas e fotos de perfil.

**Arquitetura do Sistema**

* **Frontend**

Next.js: Framework React para construção de interfaces modernas e responsivas.

Tailwind CSS: Para estilização e design responsivo.

Axios: Para consumo da API.

React Hook Form: Para gerenciamento de formulários.

Zustand: Para gerenciamento de estado global.

* **Backend**

Django Ninja: Para construção de APIs RESTful de forma rápida e eficiente.

Django REST Framework: Para construção de APIs RESTful com funcionalidades avançadas.

Django ORM: Para interação com o banco de dados.

Celery: Para tarefas assíncronas, como envio de e-mails e processamento de arquivos.

Redis: Como broker para o Celery.

* **Banco de Dados**

MySQL/PostgreSQL: Para armazenamento de dados.

Amazon S3: Para armazenamento de arquivos como comprovativos de pagamentos, faturas e fotos de perfil.

* **Infraestrutura**

Docker: Para conteinerização da aplicação.

Kubernetes: Para orquestração de contêineres.

Nginx: Como proxy reverso.

AWS/GCP: Para hospedagem e infraestrutura em nuvem.

**Funcionalidades do Sistema**

* **Cadastro e Autenticação**

Cadastro de Proprietários: O proprietário cadastra o condomínio e seus dados pessoais.

Cadastro de Moradores: Moradores se cadastram individualmente e solicitam associação ao condomínio.

Autenticação: Autenticação via JWT (JSON Web Token) para segurança e controle de acesso.

* **Gestão de Condomínios**

Cadastro de Condomínios: O proprietário cadastra o condomínio, incluindo detalhes como endereço, número de edifícios, andares, etc.

Centro de Custos e Receitas: Cadastro de despesas e receitas, incluindo despesas com pessoal, manutenção de máquinas, etc.

Gestão de Moradores: Cadastro e gestão de moradores, incluindo permissões e funções.

Gestão de Funcionários: Cadastro de funcionários como zelador, empregada de limpeza, etc., incluindo salários, contratos e pagamentos.

* **Pagamentos e Financeiro**

Pagamento de Taxas Condominiais: Moradores podem pagar taxas condominiais mensalmente.

Pagamento de Reservas: Pagamento por reserva de espaços públicos como salão de festas, estacionamento, etc.

Doações: Moradores podem fazer doações para o condomínio.

Comprovativos de Pagamento: Upload de comprovativos de pagamento e faturas.

* **Gestão de Eventos e Reservas**

Reserva de Espaços: Moradores podem reservar espaços como salão de festas, parques de diversão, etc.

Gestão de Eventos: Criação e gestão de eventos no condomínio.

* **Relatórios e Análises**

Relatórios Financeiros: Geração de relatórios financeiros detalhados.

Relatórios de Manutenção: Relatórios de manutenção de máquinas e equipamentos.

Análise de Desempenho: Análise de desempenho financeiro e operacional do condomínio.

* **Notificações e Comunicação**

Notificações: Notificações em tempo real para moradores e funcionários.

Comunicação Interna: Sistema de mensagens internas para comunicação entre moradores e funcionários.

**Diferenciais do Sistema**

* **Design Moderno e Intuitivo**

Interface moderna e responsiva, com foco na experiência do usuário.

Design personalizável para cada condomínio.

* **Escalabilidade**

Capacidade de gerenciar condomínios de diversas arquiteturas e tamanhos.

Suporte para múltiplos condomínios na mesma plataforma.

* **Segurança**

Autenticação via JWT.

Criptografia de dados sensíveis.

Backup automático e recuperação de desastres.

* **Integração com Sistemas Externos**

API para integração com sistemas de pagamento, ERP, etc.

Futura integração com aplicativo móvel.

* **Customização**

Possibilidade de customização de fluxos de trabalho e permissões.

Suporte para múltiplos idiomas e moedas.

**Modelo de Dados**

* **Entidades Principais**

Condomínio: Informações sobre o condomínio, como endereço, número de edifícios, etc.

Morador: Informações sobre os moradores, incluindo dados pessoais e permissões.

Funcionário: Informações sobre os funcionários, incluindo salários, contratos, etc.

Centro de Custos: Despesas e receitas do condomínio.

Pagamento: Informações sobre pagamentos de taxas condominiais, reservas, etc.

Evento: Informações sobre eventos e reservas de espaços.

* **Relacionamentos**

Um Condomínio pode ter vários Moradores e Funcionários.

Um Morador pode fazer vários Pagamentos.

Um Funcionário pode estar associado a vários Centros de Custos.

Um Evento pode ser associado a um Condomínio e a vários Moradores.

**Exemplos de API Endpoints**

Autenticação

POST /api/auth/register: Cadastro de usuários.

POST /api/auth/login: Login de usuários.

POST /api/auth/refresh: Atualização de token JWT.

6.2. Condomínios

GET /api/condominios: Lista todos os condomínios.

POST /api/condominios: Cria um novo condomínio.

GET /api/condominios/{id}: Retorna detalhes de um condomínio.

PUT /api/condominios/{id}: Atualiza um condomínio.

DELETE /api/condominios/{id}: Remove um condomínio.

Moradores

GET /api/moradores: Lista todos os moradores.

POST /api/moradores: Cria um novo morador.

GET /api/moradores/{id}: Retorna detalhes de um morador.

PUT /api/moradores/{id}: Atualiza um morador.

DELETE /api/moradores/{id}: Remove um morador.

Pagamentos

GET /api/pagamentos: Lista todos os pagamentos.

POST /api/pagamentos: Cria um novo pagamento.

GET /api/pagamentos/{id}: Retorna detalhes de um pagamento.

PUT /api/pagamentos/{id}: Atualiza um pagamento.

DELETE /api/pagamentos/{id}: Remove um pagamento.

Eventos

GET /api/eventos: Lista todos os eventos.

POST /api/eventos: Cria um novo evento.

GET /api/eventos/{id}: Retorna detalhes de um evento.

PUT /api/eventos/{id}: Atualiza um evento.

DELETE /api/eventos/{id}: Remove um evento.

**Fluxo de Trabalho**

**Cadastro de Condomínio**

O proprietário acessa a plataforma e cadastra o condomínio.

O sistema redireciona o proprietário para completar o cadastro com seus dados pessoais.

Após o cadastro, o proprietário é direcionado para cadastrar centros de custos e receitas.

Cadastro de Moradores

O morador se cadastra na plataforma.

O morador solicita associação ao condomínio.

O proprietário valida a associação do morador.

**Pagamento de Taxas Condominiais**

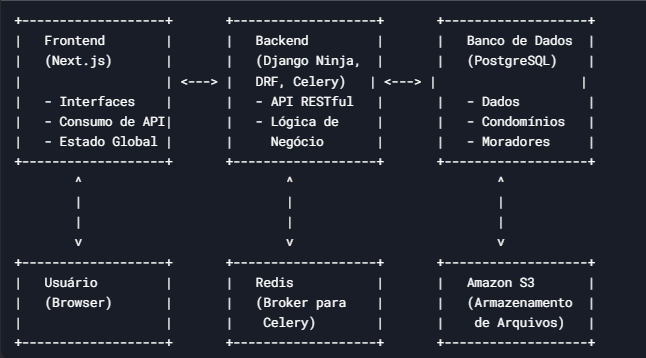
O morador acessa a plataforma e seleciona a opção de pagamento de taxas.

O sistema gera uma fatura e o morador realiza o pagamento.

O morador pode fazer upload do comprovativo de pagamento.

**Representação Gráfica da Arquitetura do Sistema Condofácil**

**Diagrama de Arquitetura**

****

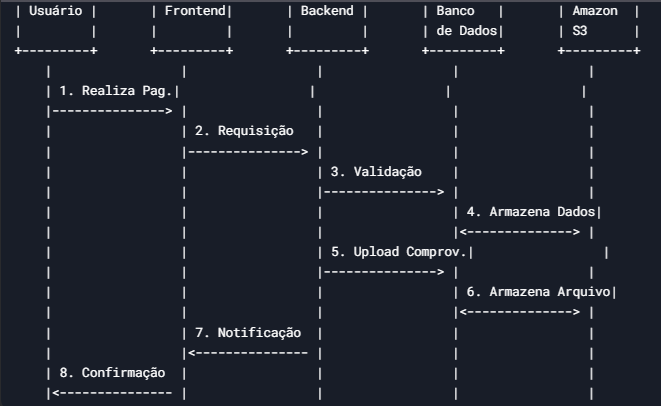
**Descrição dos Componentes**

1. **Frontend (Next.js):**
   * Responsável pela interface do usuário.
   * Consome a API do backend para exibir dados e interagir com o sistema.
   * Utiliza Tailwind CSS para estilização e Zustand para gerenciamento de estado global.
2. **Backend (Django Ninja, Django REST Framework, Celery):**
   * Fornece a API RESTful para o frontend.
   * Gerência a lógica de negócio, como cadastro de condomínios, moradores, pagamentos, etc.
   * Utiliza Celery para tarefas assíncronas (envio de notificações, processamento de arquivos).
   * Redis é usado como broker para o Celery.
3. **Banco de Dados (PostgreSQL):**
   * Armazena dados estruturados, como informações de condomínios, moradores, funcionários, pagamentos, etc.
   * Garante a persistência e integridade dos dados.
4. **Amazon S3:**
   * Armazena arquivos digitais, como comprovativos de pagamento, faturas e fotos de perfil.
   * Oferece escalabilidade e segurança para o armazenamento de arquivos**.**
5. **Redis:**
   * Atua como broker para o Celery, permitindo a execução de tarefas assíncronas.
   * Também pode ser usado para cache de dados frequentes.
6. **Usuário (Browser):**
   * Interface final onde os usuários interagem com o sistema.
   * Acessa o frontend via navegador.

**Fluxo de Dados**

1. **Frontend:**
   * O usuário interage com a interface (Next.js).
   * O frontend faz requisições à API do backend (Django Ninja/DRF) para buscar ou enviar dados.
2. **Backend:**
   * Recebe as requisições do frontend.
   * Processa a lógica de negócio (cadastro de condomínios, moradores, pagamentos, etc.).
   * Armazena dados no banco de dados (PostgreSQL) e arquivos no Amazon S3.
   * Envia notificações e executa tarefas assíncronas via Celery e Redis.
3. **Banco de Dados:**
   * Armazena informações estruturadas, como condomínios, moradores, funcionários, pagamentos, etc.
   * Fornece dados para o backend quando necessário.
4. **Amazon S3:**
   * Armazena arquivos digitais enviados pelos usuários (comprovativos, faturas, fotos de perfil).
   * Fornece links de acesso para download ou visualização.
5. **Redis:**
   * Gerência filas de tarefas assíncronas (envio de notificações, processamento de arquivos).
   * Melhora a performance do sistema com cache de dados frequentes.

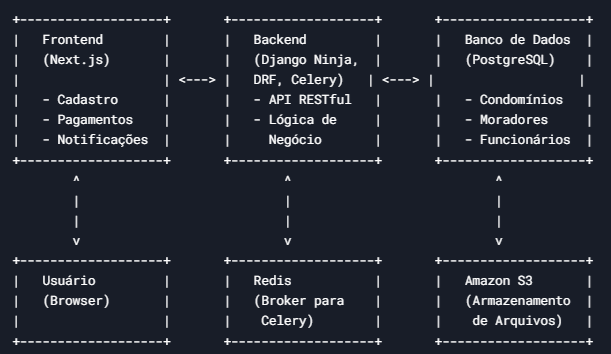
**Diagrama de Sequência (Fluxo de Pagamento de Taxa Condominial)**

****

**Legenda do Fluxo**

1. O usuário realiza um pagamento de taxa condominial através da interface do frontend.
2. O frontend faz uma requisição ao backend para registrar o pagamento.
3. O backend valida os dados e armazena as informações no banco de dados.
4. O backend faz o upload do comprovativo de pagamento no Amazon S3.
5. O backend notifica o frontend sobre o sucesso do pagamento.
6. O frontend exibe uma confirmação para o usuário.

**Diagrama de Componentes Detalhado**

****

**Legenda do Diagrama de Componentes**

1. **Frontend:**
   * Interface do usuário construída com Next.js.
   * Responsável por exibir dados e interagir com o backend.
2. **Backend:**
   * API RESTful construída com Django Ninja e Django REST Framework.
   * Gerência a lógica de negócio e integra-se com o banco de dados e Amazon S3.
3. **Banco de Dados:**
   * Armazena todas as informações do sistema, como condomínios, moradores, funcionários, pagamentos, etc.
4. **Amazon S3:**
   * Armazena arquivos digitais, como comprovativos de pagamento, faturas e fotos de perfil.
5. **Redis:**
   * Gerência tarefas assíncronas, como envio de notificações e processamento de arquivos.
6. **Usuário:**
   * Interface final onde os usuários interagem com o sistema via navegador.

**Considerações Finais**

O Condofácil será desenvolvido com o objetivo de oferecer uma solução completa e diferenciada para a gestão de condomínios, com foco em design moderno, escalabilidade e segurança.

A utilização de tecnologias como Next.js, Django Ninja e Django REST Framework garante uma experiência de usuário fluida e uma API robusta e escalável.

O sistema está preparado para futuras expansões, incluindo a integração com um aplicativo móvel e suporte para novas funcionalidades.