Documentação Técnica - Primeira Entrega Back-End

# 1. VISÃO GERAL DO PROJETO

## 1.1 Descrição do Sistema

O sistema tem como objetivo fornecer uma plataforma para apresentação dos dados profissionais de um(a) professor(a), incluindo suas áreas de atuação, projetos desenvolvidos, publicações acadêmicas e orientações realizadas. Na perspectiva do back-end, o foco está na construção de uma API REST robusta, escalável e segura, que permita o gerenciamento dessas informações. A API foi projetada para suportar operações CRUD para todas as entidades principais: Áreas de Pesquisa, Projetos, Publicações, Orientações e Mensagens de Contato. Todos os dados são armazenados em um banco de dados relacional, garantindo integridade e consistência. O desenvolvimento segue boas práticas de arquitetura, segurança e documentação da API.

## 1.2 Requisitos Funcionais do Back-end

- RF01: Gerenciar informações do professor (dados pessoais, formação, áreas de atuação)  
- RF02: Armazenar e fornecer dados de publicações acadêmicas  
- RF03: Gerenciar projetos de pesquisa, ensino e extensão  
- RF04: Armazenar informações sobre orientações (concluídas e em andamento)  
- RF05: Processar mensagens do formulário de contato  
- RF06: Documentação da API utilizando Swagger (OpenAPI)

## 1.3 Requisitos Não-Funcionais

- RNF01: Segurança - Proteção contra injeção de SQL, XSS e CSRF  
- RNF02: Performance - Tempo de resposta máximo de 2 segundos  
- RNF03: Escalabilidade - Arquitetura que permita crescimento futuro  
- RNF04: Manutenibilidade - Código bem documentado, modularizado e organizado  
- RNF05: Acessibilidade via API REST documentada e testada

# 2. ARQUITETURA DO SISTEMA

## 2.1 Visão Geral da Arquitetura

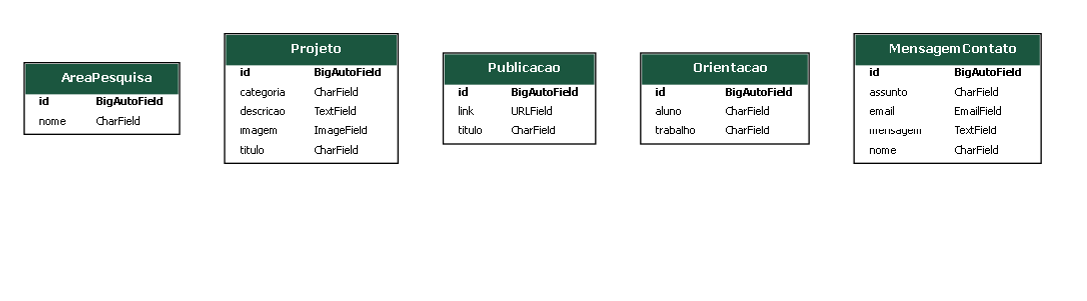
O projeto adota uma arquitetura baseada no padrão MVC (Model-View-Controller) utilizando o framework Django, complementado pela biblioteca Django Ninja para criação de APIs RESTful. A camada de modelo (Model) representa as entidades do sistema. A camada de controle (View) está dividida em arquivos específicos por entidade, oferecendo endpoints CRUD bem organizados. A API é documentada automaticamente via Swagger, fornecendo uma interface de teste para desenvolvedores. A comunicação entre front-end e back-end se dá exclusivamente via API. O banco de dados utilizado é SQLite, acessado através do ORM padrão do Django.

## 2.2 Tecnologias Selecionadas

- Linguagem: Python  
- Framework: Django + Django Ninja  
- Banco de Dados: SQLite  
- ORM: ORM nativo do Django  
- Ferramentas: pip, requirements.txt, pytest, Git, GitHub, Django Admin, PyCharm

# 3. MODELAGEM DO BANCO DE DADOS

## 3.1 Diagrama Entidade-Relacionamento (ER)



## 3.2 Descrição das Entidades

### Entidade: Área de Pesquisa

| Campo | Tipo | Restrições | Descrição |  
|-------|--------------|--------------|-------------------------------|  
| id | INTEGER | PK, NOT NULL | Identificador único da área |  
| nome | VARCHAR(100) | NOT NULL | Nome da área de pesquisa |

### Entidade: Projeto

| Campo | Tipo | Restrições | Descrição |  
|------------|--------------|--------------|---------------------------------------------------|  
| id | INTEGER | PK, NOT NULL | Identificador único do projeto |  
| titulo | VARCHAR(200) | NOT NULL | Título do projeto |  
| descricao | TEXT | NOT NULL | Descrição detalhada do projeto |  
| imagem | VARCHAR(255) | NOT NULL | Caminho da imagem associada ao projeto |  
| categoria | VARCHAR(20) | NOT NULL | Categoria (Pesquisa, Ensino ou Extensão) |

### Entidade: Publicação

| Campo | Tipo | Restrições | Descrição |  
|-------|--------------|--------------|-----------------------------------|  
| id | INTEGER | PK, NOT NULL | Identificador único da publicação |  
| titulo| VARCHAR(200) | NOT NULL | Título da publicação |  
| link | URL | NOT NULL | Link para acesso à publicação |

### Entidade: Orientação

| Campo | Tipo | Restrições | Descrição |  
|----------|--------------|--------------|-----------------------------------------------|  
| id | INTEGER | PK, NOT NULL | Identificador único da orientação |  
| aluno | VARCHAR(100) | NOT NULL | Nome do aluno orientado |  
| trabalho | VARCHAR(200) | NOT NULL | Título do trabalho orientado |

### Entidade: Mensagem de Contato

| Campo | Tipo | Restrições | Descrição |  
|---------|--------------|--------------|-------------------------------------|  
| id | INTEGER | PK, NOT NULL | Identificador único da mensagem |  
| nome | VARCHAR(100) | NOT NULL | Nome do remetente |  
| email | VARCHAR(100) | NOT NULL | Email de contato do remetente |  
| assunto | VARCHAR(100) | NOT NULL | Assunto da mensagem |  
| mensagem| TEXT | NOT NULL | Corpo da mensagem enviada |

# 4. DOCUMENTAÇÃO DA API REST

## 4.1 Visão Geral da API

A API REST foi desenvolvida utilizando Django Ninja, adotando práticas RESTful, com suporte a operações CRUD (Create, Read, Update, Delete) para todas as entidades do sistema. As rotas estão organizadas por grupos no Swagger, permitindo uma navegação limpa e intuitiva. As respostas seguem o padrão JSON. Toda a API está documentada automaticamente e pode ser testada diretamente pela interface Swagger acessível em /api/docs.

## 4.2 Endpoints da API

### Grupo: Áreas de Pesquisa

| Método | Rota | Descrição |
| --- | --- | --- |
| GET | /api/areas/ | Lista todas as areas |
| GET | /api/areas/{id} | Retorna uma area especifica |
| POST | /api/areas/{id}/ | Cria uma nova área |
| PUT | /api/areas/{id}/ | Atualiza uma área existente |
| DELETE | /api/areas/{id}/ | Deleta uma área existente |

### Grupo: Projetos

| Método | Rota | Descrição |
| --- | --- | --- |
| GET | /api/projetos/ | Lista todas os projetos |
| GET | /api/projetos/{id} | Retorna um projeto especifico |
| POST | /api/projetos/{id}/ | Cria uma novo projeto |
| PUT | /api/projetos/{id}/ | Atualiza um projeto existente |
| DELETE | /api/projetos/{id}/ | Deleta um projeto existente |

### Grupo: Publicações

| Método | Rota | Descrição |
| --- | --- | --- |
| GET | /api/publicacoes/ | Lista todas as publicações |
| GET | /api/publicacoes/{id} | Retorna uma publicação específica |
| POST | /api/publicacoes/{id}/ | Cria uma nova publicação |
| PUT | /api/publicacoes/{id}/ | Atualiza uma publicação existente |
| DELETE | /api/publicacoes/{id}/ | Deleta uma publicação existente |

### Grupo: Orientações

| Método | Rota | Descrição |
| --- | --- | --- |
| GET | /api/orientacoes/ | Lista todas as orientações |
| GET | /api/orientacoes/{id} | Retorna uma orientação específica |
| POST | /api/orientacoes/{id}/ | Cria uma nova orientação |
| PUT | /api/orientacoes/{id}/ | Atualiza uma orientação existente |
| DELETE | /api/orientacoes/{id}/ | Deleta uma orientação existente |

### Grupo: Contatos (Mensagens)

| Método | Rota | Descrição |
| --- | --- | --- |
| GET | /api/contatos/ | Lista todos os contatos |
| GET | /api/contatos/{id} | Retorna um contato específico |
| POST | /api/contatos/{id}/ | Cria uma novo contato |
| PUT | /api/contatos/{id}/ | Atualiza um contato existente |
| DELETE | /api/contatos/{id}/ | Deleta um contato existente |

5. SETUP INICIAL DO PROJETO

## 5.1 Estrutura de Diretórios

backend/  
├── Backend/ # Configurações principais do projeto Django  
│ ├── \_\_init\_\_.py  
│ ├── settings.py  
│ ├── urls.py  
│ ├── wsgi.py  
│ └── asgi.py  
├── core/  
│ ├── api/  
│ │ ├── \_\_init\_\_.py  
│ │ ├── schemas.py  
│ │ ├── views\_area.py  
│ │ ├── views\_projeto.py  
│ │ ├── views\_publicacao.py  
│ │ ├── views\_orientacao.py  
│ │ ├── views\_contato.py  
│ ├── migrations/  
│ ├── \_\_init\_\_.py  
│ ├── admin.py  
│ ├── apps.py  
│ ├── forms.py  
│ ├── models.py  
│ ├── urls.py  
│ └── api\_urls.py  
├── manage.py  
└── requirements.txt

## 5.2 Instruções para Execução

1. Clone o repositório  
2. Instale dependências: pip install -r requirements.txt  
3. Execute as migrações: python manage.py migrate  
4. Execute o servidor: python manage.py runserver  
5. Acesse a API em: http://localhost:8000/api/docs