Documentação Técnica

Documentação das Rotas da API

Usuários

Registro de Usuário

- **Endpoint**: POST /usuarios/register
- **Descrição**: Registra um novo usuário.
- **Requisição**:
  - **Body**:

```
{
  "nome": "João Silva",
  "email": "joao.silva@example.com",
  "senha": "SenhaSegura@123"
}
```

- **Resposta (200)**:

```
{
  "id": 1,
  "nome": "João Silva",
  "email": "joao.silva@example.com"
}
```

Node.js Example:

```
const axios = require('axios');
const registerUser = async () => {
  try {
    const response = await axios.post('http://localhost:3000/usuarios/register',
{
      nome: "João Silva",
      email: "joao.silva@example.com",
      senha: "SenhaSegura@123"
    });
    console.log(response.data);
  } catch (error) {
    console.error(error.response.data);
  }
};

registerUser();
```

cURL Example:

```
curl -X POST http://localhost:3000/usuarios/register \
 -H "Content-Type: application/json" \
 -d '{"nome": "João Silva", "email": "joao.silva@example.com", "senha":
"SenhaSegura@123"}'
```

Login do Usuário

- **Endpoint**: POST /usuarios/login
- **Descrição**: Faz login e retorna um token JWT.
- **Requisição**:
  - **Body**:

```
{
  "email": "joao.silva@example.com",
  "senha": "SenhaSegura@123"
}
```

- **Resposta (200)**:

```
{
  "token": "eyJhbGciOiJIUzI1..."
}
```

Node.js Example:

```
const axios = require('axios');
const loginUser = async () => {
  try {
    const response = await axios.post('http://localhost:3000/usuarios/login', {
      email: "joao.silva@example.com",
      senha: "SenhaSegura@123"
    });
    console.log(response.data);
  } catch (error) {
    console.error(error.response.data);
  }
};

loginUser();
```

cURL Example:

```
curl -X POST http://localhost:3000/usuarios/login \
 -H "Content-Type: application/json" \
 -d '{"email": "joao.silva@example.com", "senha": "SenhaSegura@123"}’
```

Publicações

Buscar Publicações com Filtros

- **Endpoint**: GET /publicacoes
- **Descrição**: Busca publicações com base em filtros opcionais.
- **Requisição**:
  - **Headers**:

```
{
  "Authorization": "Bearer <seu_token>"
}
```

  - **Query Parameters** (opcional):
    - search: Palavra-chave para busca (string).
    - dataInicio: Data inicial (YYYY-MM-DD).
    - dataFim: Data final (YYYY-MM-DD).
    - offset: Paginação, registros a pular.
    - limit: Paginação, número máximo de registros.
- **Resposta (200)**:

```
{
  "nova": {
    "total": 2,
    "publicacoes": [
      {
        "id": 1,
        "processo": "123456",
        "autores": "Fulano de Tal",
        "status": "nova",
        "data_disponibilizacao": "2023-12-01"
      }
    ]
  }
}
```

Node.js Example:

```
const axios = require('axios');
const fetchPublicacoes = async () => {
  try {
    const response = await axios.get('http://localhost:3000/publicacoes', {
      headers: {
        Authorization: `Bearer <seu_token>`
      },
```

```
      params: {
        search: "123456",
        dataInicio: "2023-12-01",
        dataFim: "2023-12-10",
        limit: 10
      }
    });
    console.log(response.data);
  } catch (error) {
    console.error(error.response.data);
  }
};

fetchPublicacoes();
```

cURL Example:

```
curl -X GET
"http://localhost:3000/publicacoes?search=123456&dataInicio=2023-12-01&
dataFim=2023-12-10&limit=10" \
 -H "Authorization: Bearer <seu_token>"
```

Atualizar Status de Publicação

- **Endpoint**: PUT /publicacoes/{id}/status
- **Descrição**: Atualiza o status de uma publicação.
- **Requisição**:
  - **Headers**:

```
{
  "Authorization": "Bearer <seu_token>"
}
```

  - **Body**:

```
{
  "status": "lida"
}
```

- **Resposta (200)**:

```
{
  "id": 1,
  "processo": "123456",
  "autores": "Fulano de Tal",
```

```
    "status": "lida",
    "data_disponibilizacao": "2023-12-01"
  }
```

Node.js Example:

```
const axios = require('axios');
const updateStatus = async () => {
  try {
    const response = await axios.put('http://localhost:3000/publicacoes/1/status', {
      status: "lida"
    }, {
      headers: {
        Authorization: `Bearer <seu_token>`
      }
    });
    console.log(response.data);
  } catch (error) {
    console.error(error.response.data);
  }
};

updateStatus();
```

cURL Example:

```
curl -X PUT http://localhost:3000/publicacoes/1/status \
-H "Authorization: Bearer <seu_token>" \
-H "Content-Type: application/json" \
-d '{"status": "lida"}'
```

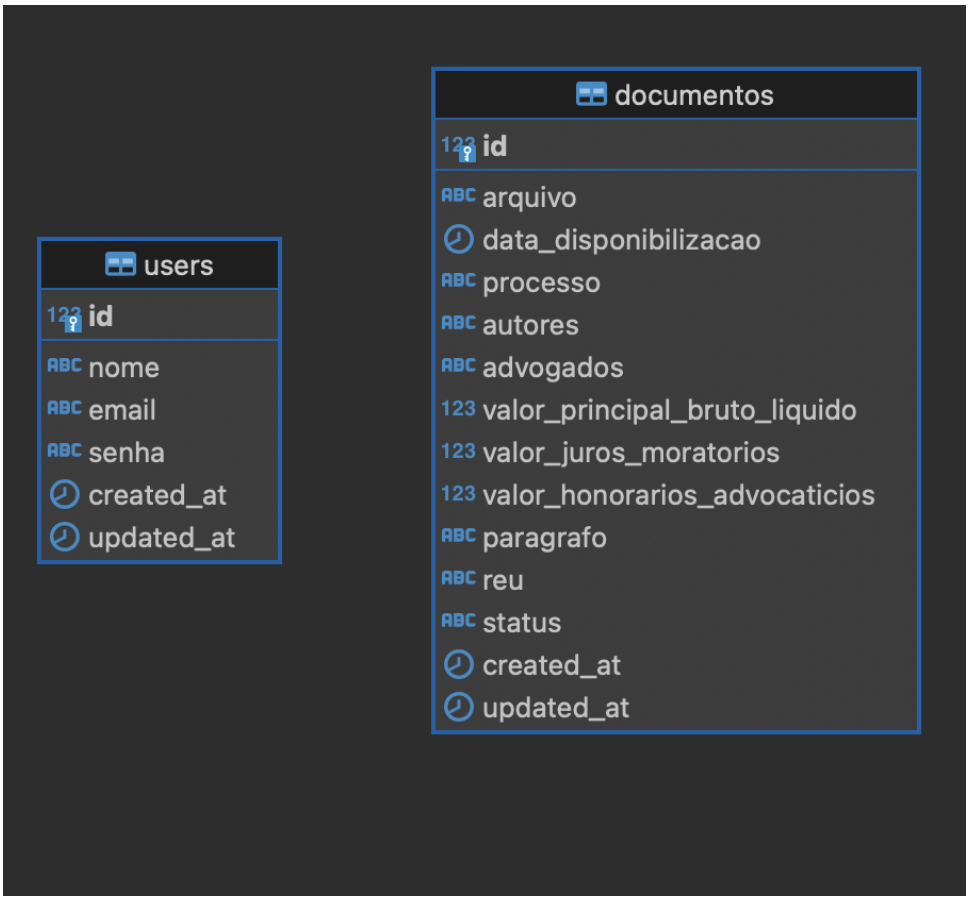
Observações Gerais

1. **Autenticação**:
   - Utilize o token JWT retornado pelo login (/usuarios/login) para autenticar as rotas protegidas (Authorization: Bearer <seu_token>).
2. **Paginação**:
   - Utilize os parâmetros offset e limit nas rotas de busca para lidar com grandes volumes de dados.
3. **Erros Comuns**:
   - 401 Unauthorized: Certifique-se de passar o token JWT corretamente no cabeçalho.
   - 404 Not Found: Certifique-se de usar IDs válidos nas requisições.

Estrutura do Banco de Dados

Esta seção descreve a estrutura do banco de dados do sistema, explicando as tabelas e seus relacionamentos.

Diagrama de Entidade-Relacionamento



Descrição das Tabelas

Tabela: users

- **Descrição**: Contém informações dos usuários que acessam o sistema.
- **Colunas**:
  - id: Identificador único do usuário (chave primária).
  - nome: Nome completo do usuário.
  - email: Endereço de e-mail único do usuário (chave única).
  - senha: Hash da senha do usuário.
  - created_at: Data e hora de criação do registro.
  - updated_at: Data e hora da última atualização do registro.

Restrições:

- users_email_key: Garante que o e-mail seja único no sistema.
- users_pkey: Define a chave primária da tabela.

Tabela: documentos

- **Descrição**: Contém informações detalhadas dos documentos processuais extraídos do sistema.
- **Colunas**:
  - id: Identificador único do documento (chave primária).
  - arquivo: Caminho ou conteúdo do arquivo relacionado ao documento.
  - data_disponibilizacao: Data em que o documento foi disponibilizado.
  - processo: Número ou identificador do processo judicial.
  - autores: Parte autora no processo.
  - advogados: Advogados responsáveis pelo caso.
  - valor_principal_bruto_liquido: Valor principal do processo (bruto e líquido).
  - valor_juros_moratorios: Valor dos juros moratórios aplicados.
  - valor_honorarios_advocaticios: Valor dos honorários advocatícios.
  - paragrafo: Parágrafo ou trecho relevante do documento.
  - reu: Parte ré no processo.
  - status: Status do documento (ex.: pendente, concluído).
  - created_at: Data e hora de criação do registro.
  - updated_at: Data e hora da última atualização do registro.

Restrições:

- documentos_pkey: Define a chave primária da tabela.

Relacionamentos

Atualmente, não há relacionamentos diretos definidos entre as tabelas users e documentos. Cada tabela opera de forma independente, mas futuras iterações do sistema podem exigir relações, como associar documentos específicos a usuários.

Modelo de Dados

Segue o modelo ERD (Entity-Relationship Diagram) com a representação das tabelas e colunas. **O diagrama anexado mostra graficamente como as tabelas estão estruturadas e suas colunas.**

SQL para Criação das Tabelas

Tabela users:

```
CREATE TABLE public.users (
    id serial4 NOT NULL,
    nome varchar(100) NOT NULL,
    email varchar(255) NOT NULL,
    senha varchar(255) NOT NULL,
    created_at timestamp DEFAULT now() NULL,
    updated_at timestamp DEFAULT now() NULL,
    CONSTRAINT users_email_key UNIQUE (email),
    CONSTRAINT users_pkey PRIMARY KEY (id)
);
```

Tabela documentos:

```
CREATE TABLE public.documentos (
    id serial4 NOT NULL,
    arquivo text NULL,
    data_disponibilizacao date NULL,
    processo text NULL,
    autores text NULL,
    advogados text NULL,
    valor_principal_bruto_liquido numeric(15, 2) NULL,
    valor_juros_moratorios numeric(15, 2) NULL,
    valor_honorarios_advocaticios numeric(15, 2) NULL,
    paragrafo text NULL,
    reu text NULL,
    status text NULL,
    created_at timestamp DEFAULT now() NULL,
    updated_at timestamp DEFAULT now() NULL,
    CONSTRAINT documentos_pkey PRIMARY KEY (id)
);
```

Fluxos de Automação e Scraping

Esta seção descreve a implementação de um sistema automatizado para realizar scraping de documentos PDF de um site público, processar as informações contidas nos documentos e armazenar os dados extraídos em um banco de dados PostgreSQL. A solução foi desenvolvida utilizando Python e integrada com Docker para facilitar a execução em ambientes controlados.

Visão Geral

1. **Automação via Cron:**
   - O crontab é configurado para executar o script diariamente às 23h.
   - Comando adicionado ao crontab:

     ```
     0 23 * * * docker run --network network_scraper python_scraper
     ```

2. **Execução Principal:**
   - O script principal (main.py) realiza as seguintes etapas:
     - Configuração inicial do banco de dados.
     - Execução do scraper para baixar arquivos PDF.
     - Processamento dos PDFs baixados e armazenamento das informações no banco de dados.
3. **Scraping de PDFs:**
   - O módulo scraper.py realiza as seguintes tarefas:
     - Navega pelas páginas do site-alvo utilizando sessões HTTP.
     - Identifica links de documentos PDF disponíveis para download.
     - Realiza o download dos documentos encontrados, salvando-os em um diretório local.
4. **Processamento de PDFs:**
   - O módulo pdf_processor.py utiliza pdfplumber para extrair informações relevantes dos PDFs, como:
     - Data de disponibilização.
     - Número do processo.
     - Autores e advogados.
     - Valores financeiros relacionados ao processo (bruto, juros e honorários).
5. **Armazenamento no Banco de Dados:**
   - Os dados extraídos dos PDFs são armazenados em uma tabela PostgreSQL, estruturada com campos para todas as informações processadas.

Pré-requisitos

- Docker e Docker Compose instalados.
- Acesso à rede para criação de contêineres.

Passo a Passo para a Execução Local

- Clonagem do Repositório
  git clone https://github.com/LucasAro/JuscashCase.git
  cd JuscashCase

- Build do Projeto
  docker compose build
  Subir o Projeto
  docker compose up -d

- Configurar Rede Compartilhada
  docker network create network_scraper
  docker network connect network_scraper meu_postgres

- Scraper (Python)
  cd scraper

- Configuração Inicial
  - Modifique `DATA_INICIAL` em `config.py` ao rodar a primeira vez para buscar as informações de 19/11/2024 até o dia atual.

  - Build do Contêiner

    docker build -t python_scraper .

- Automação com Crontab
  - Edite o crontab:

    crontab -e

- Adicione a linha para executar o scraper diariamente às 23h:

  0 23 * * * docker run --network network_scraper python_scraper

- Teste Manual
  docker run --network network_scraper python_scraper

- Após rodar a primeira vez, lembre-se de modificar `DATA_INICIAL` em `config.py` e buildar novamente o python_scraper
  docker build -t python_scraper .