Documentação Técnica - Juscash

1. Rotas da API (Swagger/OpenAPI)

yaml
CopiarEditar

```yaml
openapi: 3.0.1
info:
  title: Juscash API
  version: 1.0.0

paths:
  /api/auth/register:
    post:
      summary: Cadastra novo usuário
      requestBody:
        required: true
        content:
          application/json:
            schema:
              type: object
              required: [name, email, password]
              properties:
                name: { type: string }
                email: { type: string, format: email }
                password: { type: string }
      responses:
        '201': { description: Usuário criado com sucesso }
        '400': { description: Dados inválidos ou e-mail já existe }

  /api/auth/login:
    post:
      summary: Autentica usuário e retorna JWT
      requestBody:
        required: true
        content:
          application/json:
            schema:
              type: object
```

```
            required: [email, password]
            properties:
              email: { type: string, format: email }
              password: { type: string }
    responses:
      '200':
        description: Sucesso
        content:
          application/json:
            schema:
              type: object
              properties:
                access_token: { type: string }
      '401': { description: Credenciais inválidas }

/api/publications:
  get:
    summary: Lista/filtra publicações
    parameters:
      - in: query
        name: status
        schema:
          type: string
          enum: [new, read, sent_adv, done]
      - in: query
        name: query
        schema: { type: string }
      - in: query
        name: from
        schema: { type: string, format: date }
      - in: query
        name: to
        schema: { type: string, format: date }
    responses:
      '200':
        description: Lista de publicações
        content:
          application/json:
            schema:
              type: array
              items:
```

```
                $ref: '#/components/schemas/Publication'

  /api/publications/{id}:
    get:
      summary: Busca publicação por ID
      parameters:
        - in: path
          name: id
          required: true
          schema: { type: integer }
      responses:
        '200':
          description: Objeto Publication
        '404': { description: Não encontrada }

  /api/publications/{id}/status:
    patch:
      summary: Atualiza status de publicação
      parameters:
        - in: path
          name: id
          required: true
          schema: { type: integer }
      requestBody:
        required: true
        content:
          application/json:
            schema:
              type: object
              required: [status]
              properties:
                status:
                  type: string
                  enum: [new, read, sent_adv, done]
      responses:
        '200': { description: Status atualizado }
        '400': { description: Requisição inválida }
        '404': { description: Publicação não encontrada }
```

2. Estrutura do banco de dados

- **Tabela `users`**

Coluna	Tipo	Atributos
id	SERIAL	PK
name	VARCHAR	NOT NULL
email	VARCHAR	UNIQUE, NOT NULL
password_hash	VARCHAR	NOT NULL
created_at	TIMESTAMP TZ	DEFAULT now()
updated_at	TIMESTAMP TZ	DEFAULT now(), ON UPDATE

-

- **Tabela `publications`**

Coluna	Tipo	Atributos
id	SERIAL	PK
process_number	VARCHAR	UNIQUE, NOT NULL
publication_date	DATE	
authors	TEXT	
lawyers	TEXT	
content	TEXT	
gross_value	NUMERIC	
net_value	NUMERIC	
interest_value	NUMERIC	
attorney_fees	NUMERIC	
defendant	VARCHAR	DEFAULT 'INSS', NOT NULL
status	ENUM (4 valores)	DEFAULT 'new', NOT NULL
created_at	TIMESTAMP TZ	DEFAULT now()
updated_at	TIMESTAMP TZ	DEFAULT now(), ON UPDATE

-

(Diagrama ER pode ser incluído como anexo gráfico.)

3. Fluxos de automação e scraping

- **Inicialização do BD** (`scripts/init_db.py`):
    - Carrega `.env`, conecta ao PostgreSQL via SQLAlchemy e executa `Base.metadata.create_all()`.
- **Execução única** (`run_scraper.py`):
    - Realiza busca avançada na página de consulta do DJE com `requests` e `BeautifulSoup`.
    - Lê total de resultados, calcula páginas e extrai links de pop-up.
    - Gera URL do PDF, faz download e extrai texto via `pdfminer.six`.
    - Parse de campos essenciais (processo, data, valores) e criação de objetos `Publication`.
    - Verifica duplicatas e persiste em lote no BD usando `bulk_save_objects()`.
- **Agendamento diário** (`scheduler.py`):
    - Usa APScheduler configurado para rodar `scrape_page()` todo dia às 06:00 (fuso `America/Sao_Paulo`).

4. Dependências e instruções de configuração

Backend (Node.js)

shell
CopiarEditar
```
npm install
# .env: DB_HOST, DB_PORT, DB_NAME, DB_USER, DB_PASS, JWT_SECRET, PORT
```

-

Frontend (React)

shell
CopiarEditar

```
npm install
# .env: VITE_API_URL
```

-

Scraper (Python)

shell
CopiarEditar

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

- `# .env: DATABASE_URL`