

Documentação Técnica - JusCash

1. Visão Geral	2
2. Requisitos para Execução Local	3
3. Instruções de Instalação e Execução	4
3.1 - Passo 1: Clone o Repositório	5
3.2 - Passo 2: Verifique os Arquivos Dockerfile	5
3.3 - Passo 3: Inicie os Serviços	5
3.4 - Passo 4: Execute a Migration	5
3.5 - Passo 5: Rodar a aplicação scraping (Python)	5
3.6 - Passo 6: Verifique os Contêineres em Execução	6
3.7 - Passo 7: Acesse os Serviços	6
3.8 Passo 8: Parar os Serviços	6
4. Exemplos de Requisições à API	6
4.1 - Registro de Usuário	6
Requisição:	6
Resposta:	6
4.2 - Autenticação	7
Requisição:	7
Resposta:	7
4.3 - Busca Todas as Publicações	7
Requisição:	7
Parâmetros de Query	7
Resposta:	8
4.4 - Busca Publicação por ID	9
Descrição	9
URL	9
Parâmetros de Path	9
Resposta	9
Exemplo de Resposta Sucesso (200)	9
4.5 Atualizar Status da Publicação	10
Descrição	10
URL	10
Parâmetros de Path	10
Corpo da Requisição	10
Resposta	10
Exemplo de Resposta Sucesso (200)	10
5. Conclusão	12

1. Visão Geral

O **JusCash** é um sistema completo que integra uma API, um banco de dados PostgreSQL e uma interface frontend. Este projeto utiliza o **Docker Compose** para simplificar o processo de configuração e execução local dos seus componentes. A API do JusCash permite o gerenciamento de publicações judiciais extraídas do Diário da Justiça Eletrônico (DJE), permitindo sua organização, busca e atualização.

2. Requisitos para Execução Local

Certifique-se de que seu ambiente atenda aos seguintes requisitos:

- **Docker:** Versão 20.10 ou superior
 - **Docker Compose:** Versão 1.29 ou superior
 - **Porta 5432** livre para o banco de dados
 - **Porta 5000** livre para a API
 - **Porta 3000** livre para o frontend
-

3. Instruções de Instalação e Execução

3.1 - Passo 1: Clone o Repositório

Clone o repositório e acesse o diretório:

```
git clone https://github.com/lucas-rocha/juscashcase.git
cd juscashcase
```

3.2 - Passo 2: Verifique os Arquivos `Dockerfile`

Certifique-se de que os arquivos `Dockerfile` estão configurados corretamente nos diretórios `juscash-api` e `juscash-front`.

3.3 - Passo 3: Inicie os Serviços

Execute o comando abaixo para iniciar todos os serviços com o Docker Compose:

```
docker-compose up -d
```

Isso iniciará:

- O banco de dados PostgreSQL na porta 5432
- A API na porta 5000
- O frontend na porta 3000

3.4 - Passo 4: Execute a Migration

Execute os comandos abaixo para executar as migrations:

```
docker exec -it juscash_api /bin/bash
npx prisma migrate dev
```

3.5 - Passo 5: Rodar a aplicação scraping (Python)

Execute os comandos abaixo para criar um rede compartilhada entre o banco de dados e a aplicação em Python

```
docker network create juscash_network
docker network connect juscash_network juscash_database
```

Execute o comando abaixo para construir a imagem da aplicação:

```
docker build -t scraping .
```

Executando o container Docker:

O container scraping utiliza o parâmetro PARAMS_CONSULTA para realizar consultas. Existem dois casos de uso para este parâmetro. Siga as instruções abaixo para executar o container corretamente.

Passo 1: Executar com o código inicial

Certifique-se de que o código no arquivo onde o PARAMS_CONSULTA está definido tenha a seguinte configuração no arquivo config.py:

```
PARAMS_CONSULTA =  
f'dadosConsulta.dtInicio=19%2F12%2F2024&dadosConsulta.dtFim=19%2F12%2F2024&da  
dosConsulta.cdCaderno=12&dadosConsulta.pesquisaLivre=%22RPV%22+e+%22pagament  
o+pelo+INSS%22&pagina='
```

Assim, a aplicação buscará os dados usando a data 19/12/2024.

Execute o seguinte comando no terminal para rodar o container:

```
docker run --rm --network juscash_network scraping
```

Passo 2: Executar com o código alternativo

Edite o código para que a configuração do PARAMS_CONSULTA esteja como mostrado abaixo no config.py:

```
PARAMS_CONSULTA =  
f'dadosConsulta.dtInicio={actual_data}&dadosConsulta.dtFim={actual_data}&dadosConsulta  
.cdCaderno=12&dadosConsulta.pesquisaLivre=%22RPV%22+e+%22pagamento+pelo+INS  
S%22&pagina='
```

Assim, a aplicação buscará os dados usando a data atual. A partir disso, devemos criar um job que executa o seguinte comando para rodar o container diariamente às 00:00.

Passo 3: Criar e executar o código alternativo com o job

Iremos usar o crontab que automatiza todo o processo. Execute:

```
crontab -e
```

Assim irá abrir um editor (Vim ou nano). Cole o seguinte código e salve:

```
0 24 * * * docker run --rm --network juscash_network scraping
```

3.6 - Passo 6: Verifique os Contêineres em Execução

Use o comando para verificar se os contêineres estão funcionando corretamente:

```
docker ps
```

3.7 - Passo 7: Acesse os Serviços

Agora você pode acessar os seguintes serviços:

- **API:** <http://localhost:5000>
- **Frontend:** <http://localhost:3000>

3.8 - Passo 8: Parar os Serviços

Quando quiser parar os serviços, use o comando:

```
docker-compose down
```

4. Exemplos de Requisições à API

4.1 - Registro de Usuário

Requisição:

bash

Copiar código

POST api/users

Content-Type: application/json

```
{
  "fullname": "Lucas Souza Rocha",
  "email": "lucasrocha.dv@gmail.com",
  "password": "senha"
}
```

Resposta:

```
{
  "message": "Usuário criado com sucesso.",
  "user": {
    "id": "65ddf774-b39f-4406-a21f-5309d12a73e1",
    "fullname": "Lucas Souza Rocha",
    "email": "lucasrocha.dv@gmail.com"
  }
}
```

4.2 - Autenticação

Requisição:

POST api/users/login

Content-Type: application/json

```
{
  "email": "usuario",
  "password": "senha"
}
```

Resposta:

```
{
```

```
"message": "Usuário logado com sucesso.",
"user": {
  "token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX2lkIjoiaYjdjMDZlYzYtNTVkZi00ZDQ1LWFiMmYtNzEwN2M4ZGY0MzFmIiwiaWF0IjoxNzM1NjkwMDgyfQ.RFB-pqu5WmnYKaLQDMQiZkDu_F9ebIVX7A-605HqQq8"
}
}
```

4.3 - Busca Todas as Publicações

Requisição:

GET api/publications

Content-Type: application/json

Parâmetros de Query

Nome	Tipo	Obrigatório	Descrição
search	String	Não	Texto para realizar a busca nas publicações. Se não informado, retorna todas as publicações
dataInicio	String	Não	Data de início para filtrar publicações, no formato YYYY-MM-DD. Se não informado, sem filtro de data.
dataFim	String	Não	Data de término para filtrar publicações, no formato YYYY-MM-DD. Se não informado, sem filtro de data.
offset	Integer	Não	Índice inicial para a paginação, padrão é 0. Utilizado para buscar publicações a partir de uma posição específica.

limit	Integer	Não	Limite de publicações por página, padrão é 30. Controla o número de publicações retornadas.
-------	---------	-----	--

Resposta:

```
{
  "nova": {
    "title": "Publicações novas",
    "id": "nova",
    "tasks": [
      {
        "id": "123",
        "processNumber": "000123456789",
        "authors": "Autor Exemplo",
        "content": "Conteúdo da publicação.",
        "updatedAt": "2025-01-02T12:00:00Z"
      }
    ]
  },
  "lida": {
    "title": "Publicações Lidas",
    "id": "lida",
    "tasks": []
  },
  "enviada_adv": {
    "title": "Publicações Enviadas para o Advogado",
    "id": "enviada_adv",
    "tasks": []
  },
  "concluida": {
    "title": "Publicações Concluídas",
    "id": "concluida",
    "tasks": []
  }
}
```

4.4 - Busca Publicação por ID

Descrição

Este endpoint permite buscar uma publicação específica com base no seu ID.

URL

GET /api/publications/:id

Parâmetros de Path

Nome	Tipo	Obrigatório	Descrição
id	String	Sim	ID da publicação que será buscada.

Resposta

Exemplo de Resposta Sucesso (200)

```
{
  "id": "123",
  "processNumber": "000123456789",
  "authors": "Autor Exemplo",
  "content": "Conteúdo da publicação.",
  "status": "nova",
  "updatedAt": "2025-01-02T12:00:00Z"
}
```

4.5 Atualizar Status da Publicação

Descrição

Este endpoint permite atualizar o status de uma publicação específica com base no seu ID.

URL

PUT /api/publications/:id

Parâmetros de Path

Nome	Tipo	Obrigatório	Descrição
------	------	-------------	-----------

id	String	Sim	ID da publicação que será atualizada
----	--------	-----	--------------------------------------

Corpo da Requisição

Nome	Tipo	Obrigatório	Descrição
status	String	Sim	Novo status da publicação. Os valores válidos são: nova, lida, enviada_adv, concluida.

Resposta

Exemplo de Resposta Sucesso (200)

```
{
  "id": "123",
  "processNumber": "000123456789",
  "authors": "Autor Exemplo",
  "content": "Conteúdo da publicação.",
  "status": "lida",
  "updatedAt": "2025-01-02T12:00:00Z"
}
```

5. Conclusão

O **JusCash** é uma ferramenta robusta para o gerenciamento de publicações judiciais, proporcionando uma integração eficiente entre frontend, API e banco de dados. Com o uso do Docker, a execução local é simplificada, permitindo uma fácil configuração e manutenção do sistema. Para quaisquer dúvidas ou problemas, o time de suporte está à disposição para ajudar.