



Documentação da aplicação do Desafio



Sumário

1.	SolFácil	3
2.	Desafio	3
3.	Tecnologias	3
4.	Fluxo da Aplicação	3
5.	Rodando Aplicação	4



1. SolFácil

“Somos a primeira fintech solar da América Latina e o maior ecossistema de soluções solares do país.” ([SolFácil](#)).

2. Desafio

“Nosso cliente interno precisa atualizar rotineiramente os dados de nossos parceiros. O problema acontece que para atualizar, ele precisa entrar na página de edição de cada um dos parceiros. Isso é um trabalho muito tedioso e demorado.” ([GitHub-SolFácil](#)).

3. Tecnologias

3.1. [Python](#)

Linguagem de Programação usada para completar o desafio.

3.2. [Flask](#)

Framework python usado para a criação dos endpoints.

3.3. [PostgreSQL](#)

Banco de dados usado para armazenar os parceiros.

3.4. [Docker](#)

Container usado para rodar a aplicação em um único ambiente e configurável.

3.5. [Visual Studio Code](#)

Programa usado para codificar e iniciar a aplicação.

4. Fluxo da Aplicação

Na figura 1 abaixo, temos a apresentação ilustrativa do fluxo da aplicação.

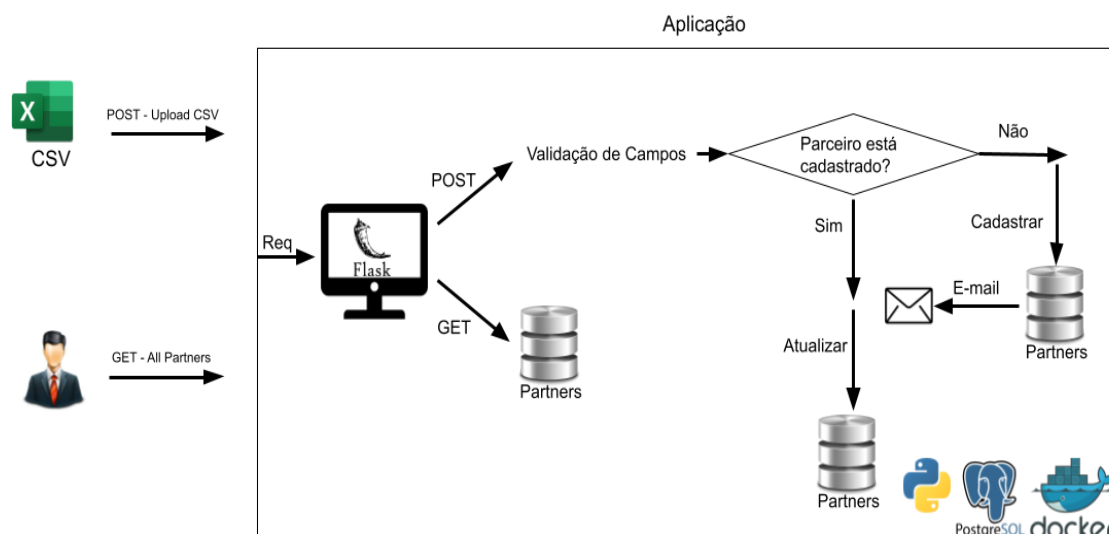


Figura 1- Fluxo da Aplicação.

A aplicação roda com um container, onde encontra-se o banco de dados e a aplicação web feita em Flask numa arquitetura MVC (Model, View e Controller), que recebe as requisições de ‘Upload CSV’ e ‘All Partners’.

Primeiro, tem-se o fluxo de carregar um csv, através de um ‘POST’, que a aplicação web irá receber e fazer a validação dos dados dos parceiros (CNPJ, Telefone E-mail e etc), após a validação, a aplicação irá verificar se o parceiro está cadastrado ou não, caso esteja cadastrado, o mesmo irá ser atualizado, caso não esteja, o parceiro será cadastrado e um e-mail será enviado confirmando o cadastro, por fim, a aplicação irá retornar a quantidade de parceiros atualizados ou salvos.

O segundo fluxo, tem-se a listagem de parceiros que estão cadastrados no banco de dados, para isso é preciso realizar uma requisição ‘GET’, onde a aplicação web irá buscar os parceiros no banco, retornando os parceiros cadastrados.

5. Rodando a Aplicação.

Nesta parte, será mostrado como executar a aplicação localmente na sua máquina.

5.1 Clonar o repositório da aplicação.

- Nesta será necessário clonar o repositório da aplicação, cujo repositório se encontra neste [link](#).
- Clique neste [link](#), caso queira saber como clonar.

5.2 Visual Studio Code.

- A partir desta etapa a execução da aplicação será toda feita através do Visual Studio Code, caso já tenha instalada vá para a próxima etapa.
- Baixe e instale o programa neste [link](#).
- Abra o projeto clonado no Visual Studio, clique neste [link](#), caso queira saber como abrir o projeto.

5.3 Extensão Visual Studio Code

- Para facilitar a execução da nossa aplicação usaremos uma extensão do Docker no Visual Studio, para subir o nosso container.
- Com o visual Studio aberta, clica na ferramenta de ‘Extensões’



Figura 2- Extensões.

- Procure por “Docker” e instale a verificada

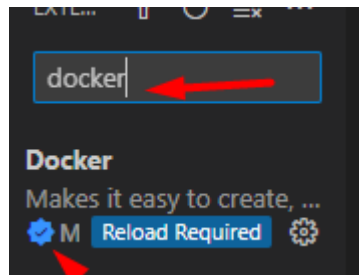
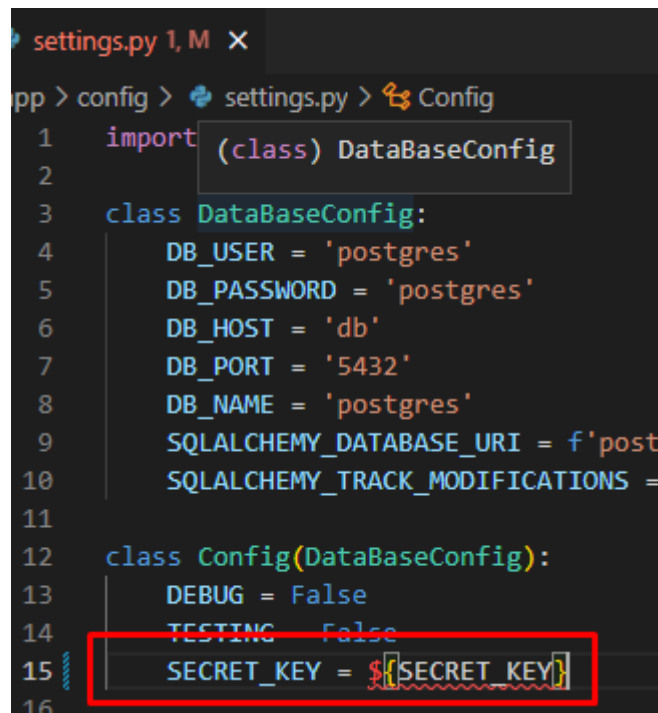


Figura 3- Extensão Docker verificada.

5.4 SECRET_KEY

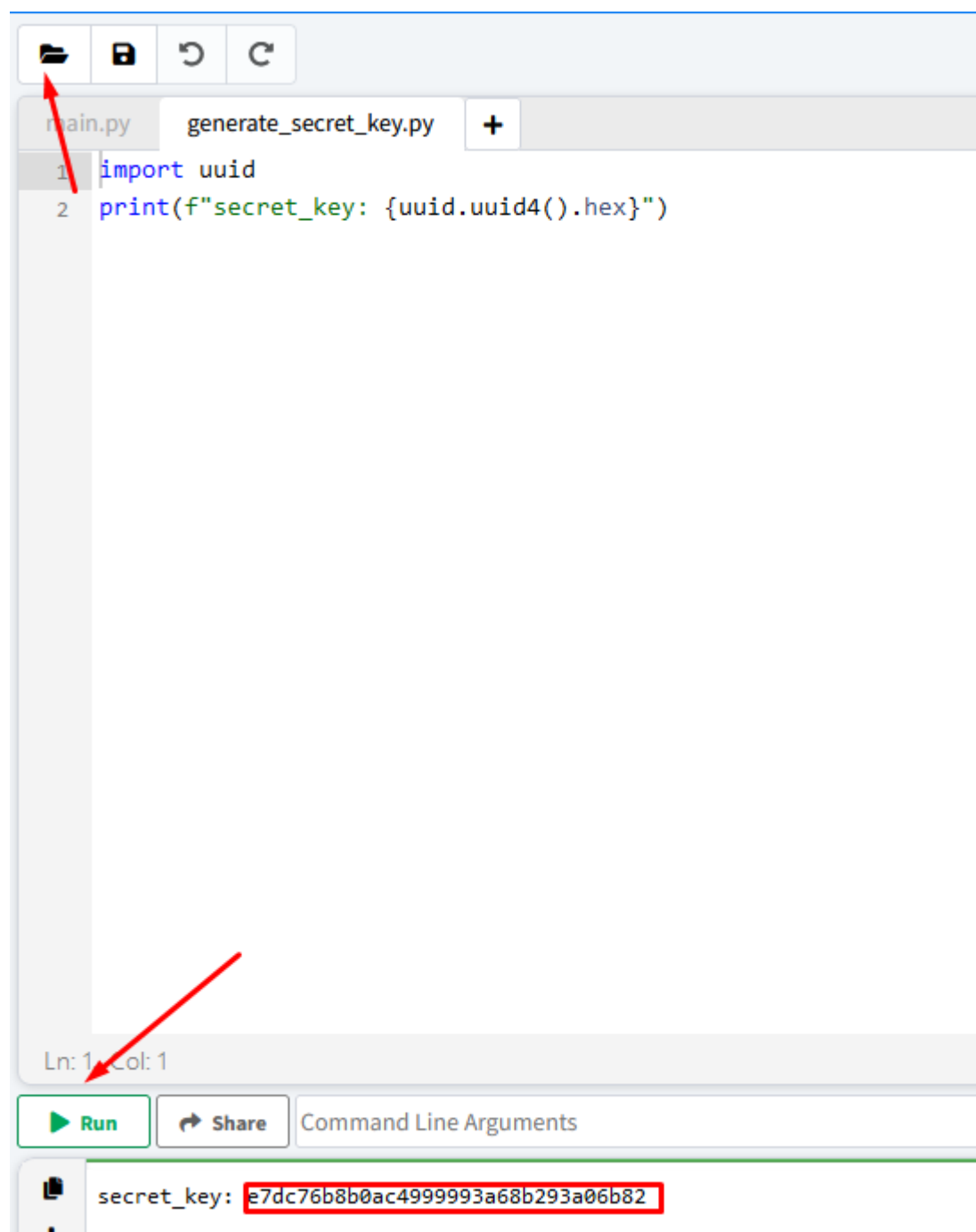
- Abra o arquivo “app/config/settings.py” e altere “\${SECRET_KEY}”, pela sua chave.



```
settings.py 1, M X
pp > config > settings.py > Config
1 import (class) DataBaseConfig
2
3 class DataBaseConfig:
4     DB_USER = 'postgres'
5     DB_PASSWORD = 'postgres'
6     DB_HOST = 'db'
7     DB_PORT = '5432'
8     DB_NAME = 'postgres'
9     SQLALCHEMY_DATABASE_URI = f'post
10     SQLALCHEMY_TRACK_MODIFICATIONS =
11
12 class Config(DataBaseConfig):
13     DEBUG = False
14     TESTING = False
15     SECRET_KEY = "${SECRET_KEY}"
16
```

Figura 4 - `${SECRET_KEY}`.

- Caso queira gerar uma acesse o [link](#), abra o arquivo “./generate_secret_key.py” e rode o programa, pegue o resultado e altere “`${SECRET_KEY}`”, pela chave gerada entre aspas duplas.



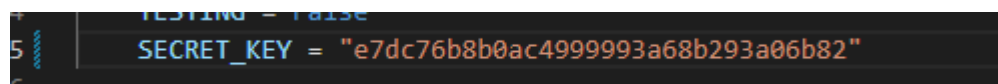
```
main.py generate_secret_key.py +
1 import uuid
2 print(f"secret_key: {uuid.uuid4().hex}")
```

Ln: 1 Col: 1

Run Share Command Line Arguments

secret_key: e7dc76b8b0ac4999993a68b293a06b82

Figura 5 - Gerando a SECRET_KEY.



```
TESTING = False
SECRET_KEY = "e7dc76b8b0ac4999993a68b293a06b82"
```

Figura 6 - Valor SECRET_KEY alterado.

5.5 Variáveis do Banco

- As variáveis do banco foram definidas como padrão, caso queira mudar, acesse “app/config/settings.py” e “./docker-compose.yml”.

```
class DatabaseConfig:
    DB_USER = 'postgres'
    DB_PASSWORD = 'postgres'
    DB_HOST = 'db'
    DB_PORT = '5432'
    DB_NAME = 'postgres'
```

Figura 7 - Variáveis database em ‘app/config/settings.py’.

```
depends_on:
  - db
db:
  image: postgres:latest
  environment:
    POSTGRES_USER: postgres
    POSTGRES_PASSWORD: postgres
    POSTGRES_DB: postgres
  ports:
    - "5432:5432"
```

Figura 8 - Variáveis database em ‘./docker-compose.yml’.

5.6 Subindo a aplicação

Com as variáveis configuradas, podemos rodar a nossa aplicação, para isso acesse o arquivo “./docker-compose.yml”, clique com o botão direito no arquivo e logo em seguida em “Compose Up”.

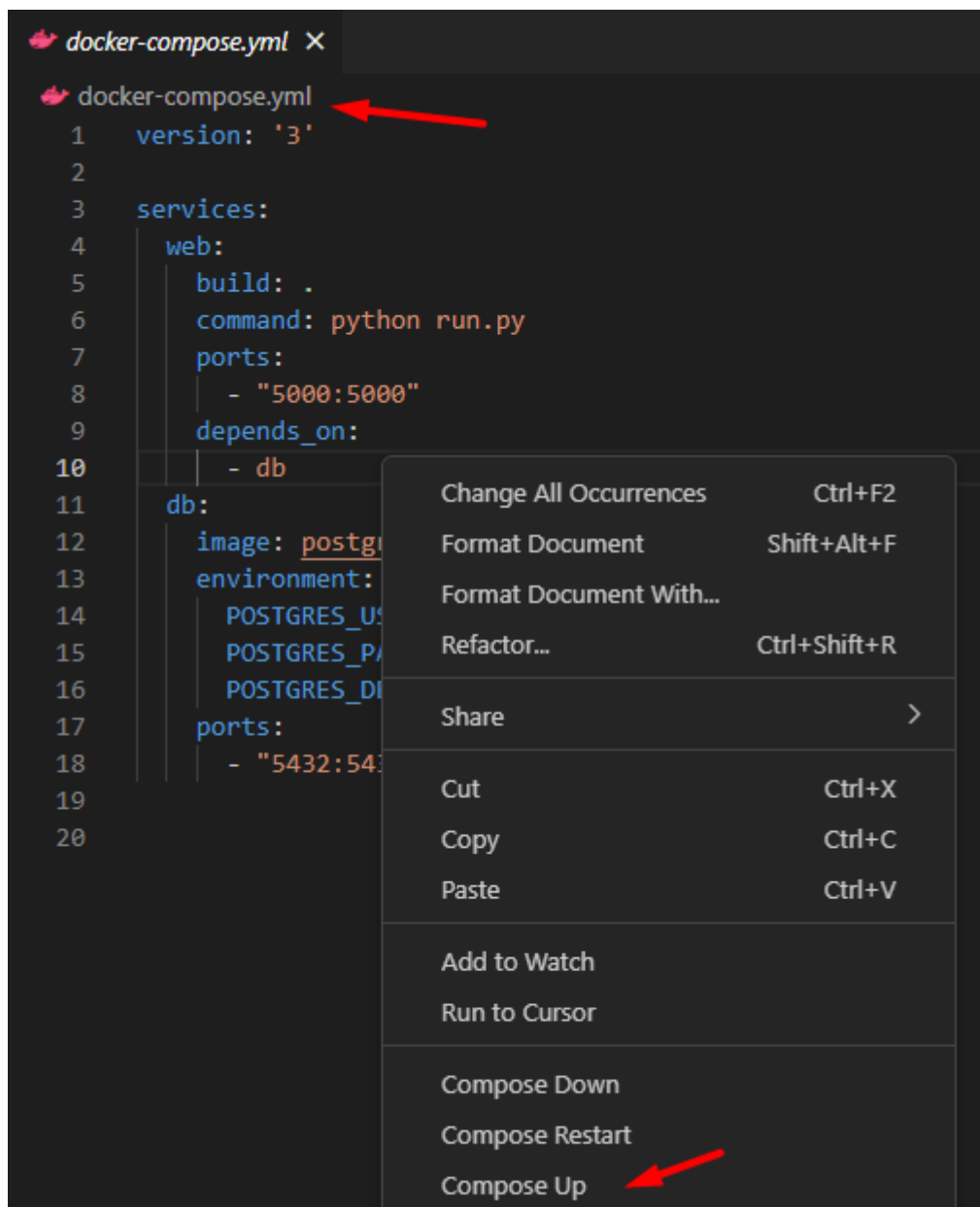


Figura 9 - Subindo container.

Acesse a extensão do Docker para verificar se o container está ativo, se tudo estiver verde, a aplicação já está de pé, pronto para uso.

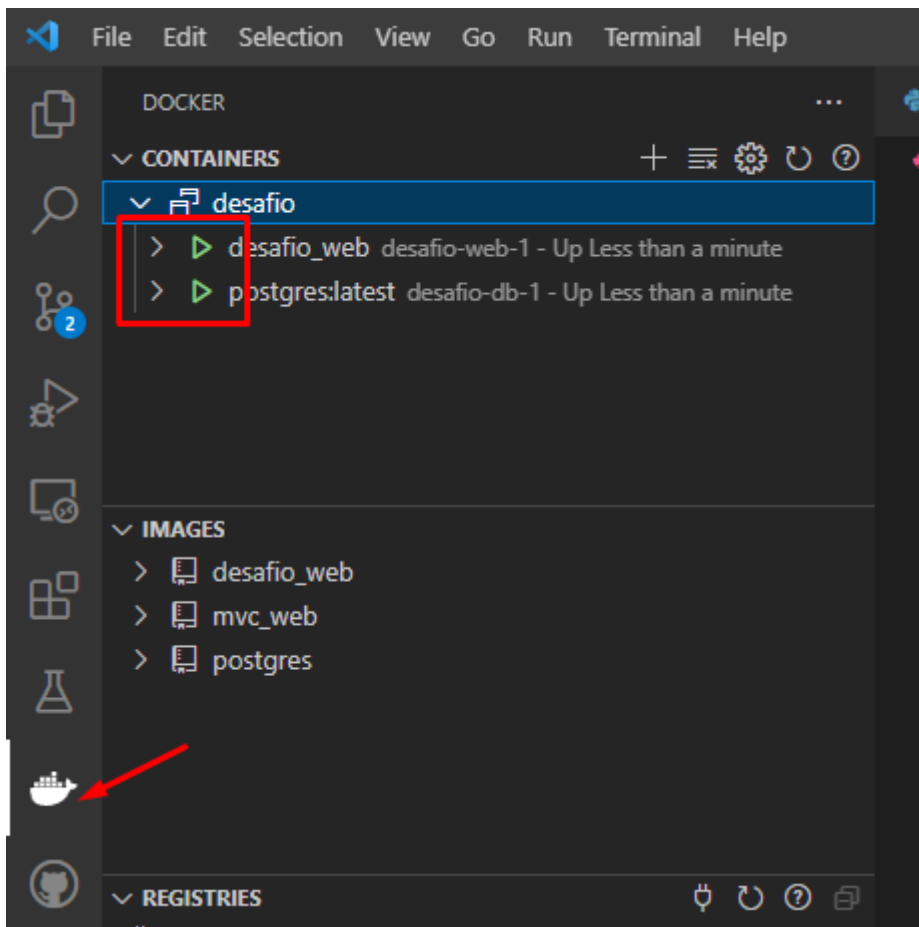


Figura 10 - Containers pronto para uso.

5.7 Testes

Alguns testes foram criados para aplicação, para testar execute o container, clicando na opção e logo em seguida com o botão direito em “Attach Shell”, isso irá abrir um prompt com o sinal “#”, ao digitar o comando ‘pytest’, todos os nossos arquivos testes serão executados.

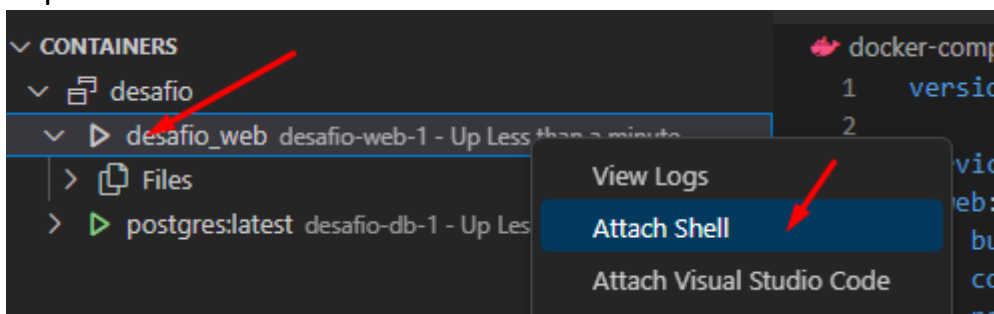
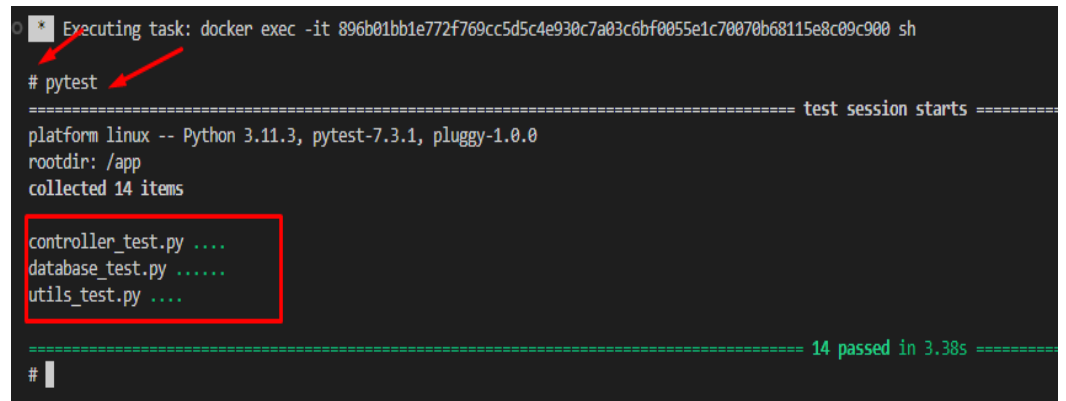


Figura 11 - Attach Shell.



```
Executing task: docker exec -it 896b01bb1e772f769cc5d5c4e930c7a03c6bf0055e1c70070b68115e8c09c900 sh
# pytest
===== test session starts =====
platform linux -- Python 3.11.3, pytest-7.3.1, pluggy-1.0.0
rootdir: /app
collected 14 items

controller_test.py ....
database_test.py .....
utils_test.py ....

===== 14 passed in 3.38s =====
#
```

Figura 12 - Execução de testes.

5.8 Endpoints

Existe duas rotas na aplicação:

- GET - /list_all_partners
- POST - /upload_partner

5.9 Testando a Aplicação

- Para testar a rota “/list_all_partners”, utilize algum software de requisições como [PostMan](#) ou [Insomnia](#), para buscar todos os parceiros.
- Para testar a rota “/upload_partner”, abra o arquivo “./test.html”, insere algum csv que encontra-se na pasta como “empty_example.csv”, “invalid_field_example.csv” e “example.csv”.