

Software para Inspeção de Ativos

Danilo José da Silva

Universidade Federal de Santa Catarina (UFSC)

I. INTRODUÇÃO

Na industria há uma grande necessidade de monitoramento dos seus equipamentos afim de evitar paradas em sua linha de produção e consequentemente evitar prejuízos. Com base nisso, na disciplina de **Projeto Integrador II** será implementado uma aplicação em *ReactJs* que mostre ao usuário os status dos ativos nas suas respectivas unidades, nele será possível verificar os ativos que estão operando, em alerta ou inativos. Além disso é necessário ter um software intuitivo ao usuário onde apresente de forma gráfica as informações necessarias afim de ajudar no planejamento de futuras manutenções preventivas.

Será desenvolvido uma aplicação que mostrará todos os dados e ações possíveis através de uma API implementada. Sendo que a aplicação desenvolvida terá as funcionalidades:

- Mostrar todas as características do ativos, unidades e usuários;
- Será utilizado gráficos para mostrar os status dos ativos.

A API por sua vez será implementada em Python com deploy na AWS conforme pode ser visto na Fig. 1.

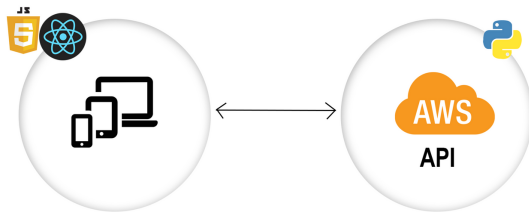


Figura 1. Figura representando a aplicação em ReactJS sendo alimentada por uma API implementada em Python com deploy na AWS.

O repositório do projeto aqui apresentado se encontra disponível no github e pode ser acessado em: <https://github.com/SIA2602/EMB5637-Projeto-Integrador-II>.

II. ESPECIFICAÇÃO DOS REQUISITOS

Nesta especificação de requisitos de software (SRS) o principal objetivo é promover um entendimento para o usuário que fará uso do software e ao desenvolvedor de como será estruturada a aplicação. A aplicação a ser desenvolvida será um facilitador no monitoramento dos ativos da empresa interessada, onde o intuito do seu uso se dará com o fim de evitar paradas inesperadas em seus equipamentos. Com a solução já implantada, haverá usuários cadastrados que

serão responsáveis por determinadas unidades, onde eles terão acesso ao status de seus ativos em tempo real.

A aplicação também será uma ferramenta útil nos planejamentos de manutenções preventivas. Os riscos envolvidos na falha da aplicação se encontra no sistema embracado que coleta os dados das maquinas e o disponibiliza a um API que é consumida pelo software. Já que o foco principal desse trabalho esta apenas no desenvolvimento de software o único risco a ser considerado estará em sua programação.

A. Descrição geral

1) **Necessidades do usuário:** A utilização da aplicação se dará por um usuário primário, sendo que esse é o responsável pela unidade em que esta implantada a aplicação. Esse usuário possui uma necessidade de estar atento a possíveis falhas e paradas de equipamentos não programadas, e com esse intuito o software desenvolvido irá facilitar não só na sua tomada de decisão mas como irá evitar deslocamentos desnecessarios aos equipamentos visto que tudo pode ser acompanhado pelo seu dispositivo móvel. Por todo histórico de cada ativo estar em um banco de dados e disponível para consulta no software será evitado a utilização de anotações em papel.

2) **Dependências:** O software esta intrinsecamente ligado ao hardware que alimenta a API da aplicação, a fatores de compatibilidade e conexão com a internet visto que o software só poderá ser utilizado online.

B. Requisitos do Sistema

1) **Requisitos Funcionais:** Como os requisitos funcionais foca **no que será feito**, então esses requisitos são todos os problemas e necessidades que devem ser atendidos e resolvidos pelo software, sendo que podemos listar as funcionalidades que:

- é responsável pelo consumo dos dados da API;
- gera Cards e Botões contendo as ações relevantes a cada página;
- oculta os itens que não estão sendo utilizadas na página;
- gera os gráficos.

2) **Requisitos não Funcionais:** Nos requisitos não funcionais o foco se dá em **como será feito**, ou seja, são as características técnicas. As quais podemos listar:

- tipos de dispositivos em que o software será utilizado;
- navegadores web compatíveis com a aplicação;
- conexão com a internet.

III. MODELAGEM DO SISTEMA

A etapa de modelagem do sistema consiste em elaborar os diadramas UML para dar uma visão geral de como está estruturado o software e suas interações.

A. Diagrama de Caso de Uso

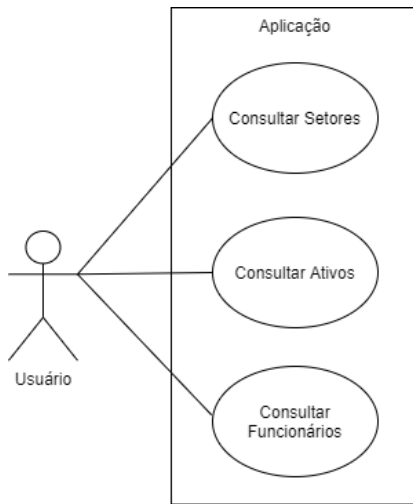


Figura 2. Diagrama de Caso de Uso

B. Diagrama de Classes

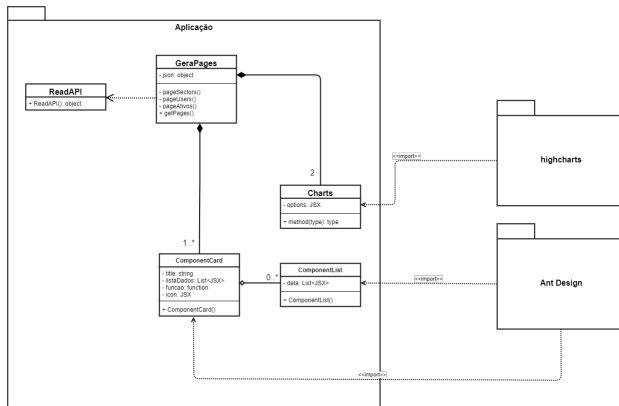


Figura 3. Diagrama de Classes

C. Diagrama Sequencia

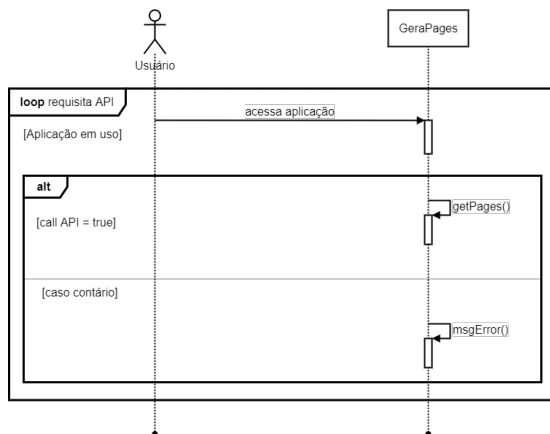


Figura 4. Diagrama de Sequencia

D. Máquina de Estados

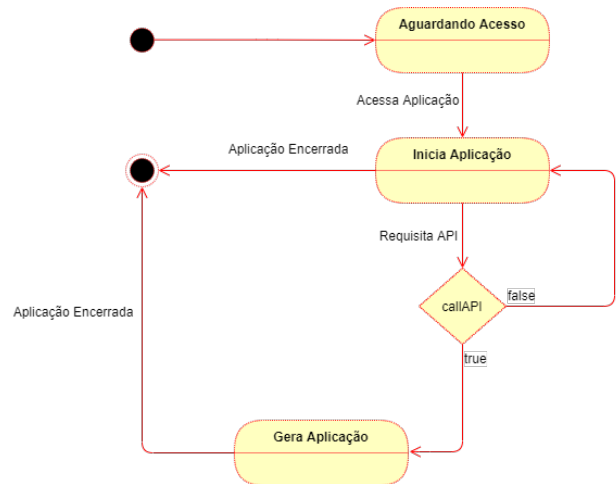


Figura 5. Máquina de Estados

Como o trabalho consiste em desenvolver um software com interface gráfica também foi realizado um estudo de interface de usuário (UI) afim de dar uma visão de como será estruturado a sua programação. Para esse estudo foi utilizado o *figma* que é uma ferramenta amplamente utilizada pelos desenvolvedores frontend. O estudo UI pode ser acessado **AQUI** e também pode ser visto na Fig. 7 disponível no final do relatório.

IV. DESENVOLVIMENTO DO PROJETO

A. Tecnologias utilizadas

A engine utilizada para o desenvolvimento do frontend foi a biblioteca para java script o **ReactJS**, sendo que os gráficos foram gerados com a biblioteca **Highcharts** e cards gerados com a biblioteca **AntDesign**. Para o consumo da API por parte do software foi utilizado a biblioteca **axios** já para o desenvolvimento da API é utilizado o micro-framework **flask** no python.

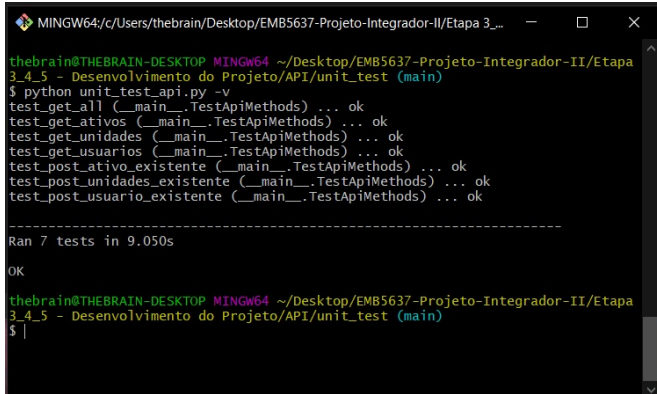
B. Testes Unitários

Como estamos lidando com dois sistemas diferentes sendo um software implementado em java script e a API desenvolvida em python ambos os sistemas precisariam passar por testes unitários. Como o desenvolvimento do software e da api por si só já possui uma certa complexidade foi optado em realizar os testes unitários apenas para a api. Para o teste unitário dos metodos **GET** e **POST** da api é utilizada a biblioteca **unittest** sendo os métodos testados:

- `[test_get_all]` retorno de todos recursos.
- `[test_get_ativos]` retorno do recurso ativos.
- `[test_get_unidades]` retorno do recurso unidades.
- `[test_get_usuarios]` retorno do recurso usuarios.
- `[test_post_ativo_existente]` testa o envio de um ativo com id já existente.
- `[test_post_unidade_existente]` testa o envio de uma unidade com id já existente.

- `[test_post_usuario_existente]` testa o envio de um usuário com id já existente.

A execução do teste unitário pode ser verificada na Fig. 6 e detalhes de sua implementação pode ser acessado no repositório do projeto ou no apêndice ao final do presente relatório.



```
MINGW64/c/Users/thebrain/Desktop/EMB5637-Projeto-Integrador-II/Etapa 3...
thebrain@THEBRAIN-DESKTOP MINGW64 ~/Desktop/EMB5637-Projeto-Integrador-II/Etapa
3.4.5 - Desenvolvimento do Projeto/API/unit_test (main)
$ python unit_test_api.py -v
test_get_all (__main__.TestApiMethods) ... ok
test_get_ativos (__main__.TestApiMethods) ... ok
test_get_unidades (__main__.TestApiMethods) ... ok
test_get_usuarios (__main__.TestApiMethods) ... ok
test_post_ativo_existente (__main__.TestApiMethods) ... ok
test_post_unidades_existente (__main__.TestApiMethods) ... ok
test_post_usuario_existente (__main__.TestApiMethods) ... ok

-----
Ran 7 tests in 9.050s

OK
thebrain@THEBRAIN-DESKTOP MINGW64 ~/Desktop/EMB5637-Projeto-Integrador-II/Etapa
3.4.5 - Desenvolvimento do Projeto/API/unit_test (main)
$ |
```

Figura 6. Execução do teste unitário para a API.

C. Aplicação Final

Ao fim da realização dos passos aqui apresentados foi realizado o deploy tanto do software quanto da api em um servidor na AWS. O acesso a aplicação pode ser feito no endereço: **<http://3.19.181.158/>** e verificado nas Fig. 8 e Fig. 9 já o acesso aos recursos disponíveis na API devem ser realizados por meio de rotas. Para o acesso a todos os recursos deve-se seguir a rota: **<http://3.19.181.158/api/tudo>**, agora para acesso a um recurso em específico com os ativos, usuarios ou unidades deve-se seguir a rota: **<http://3.19.181.158/api/ativos>** trocando a palavra ativos pelo recurso desejado.

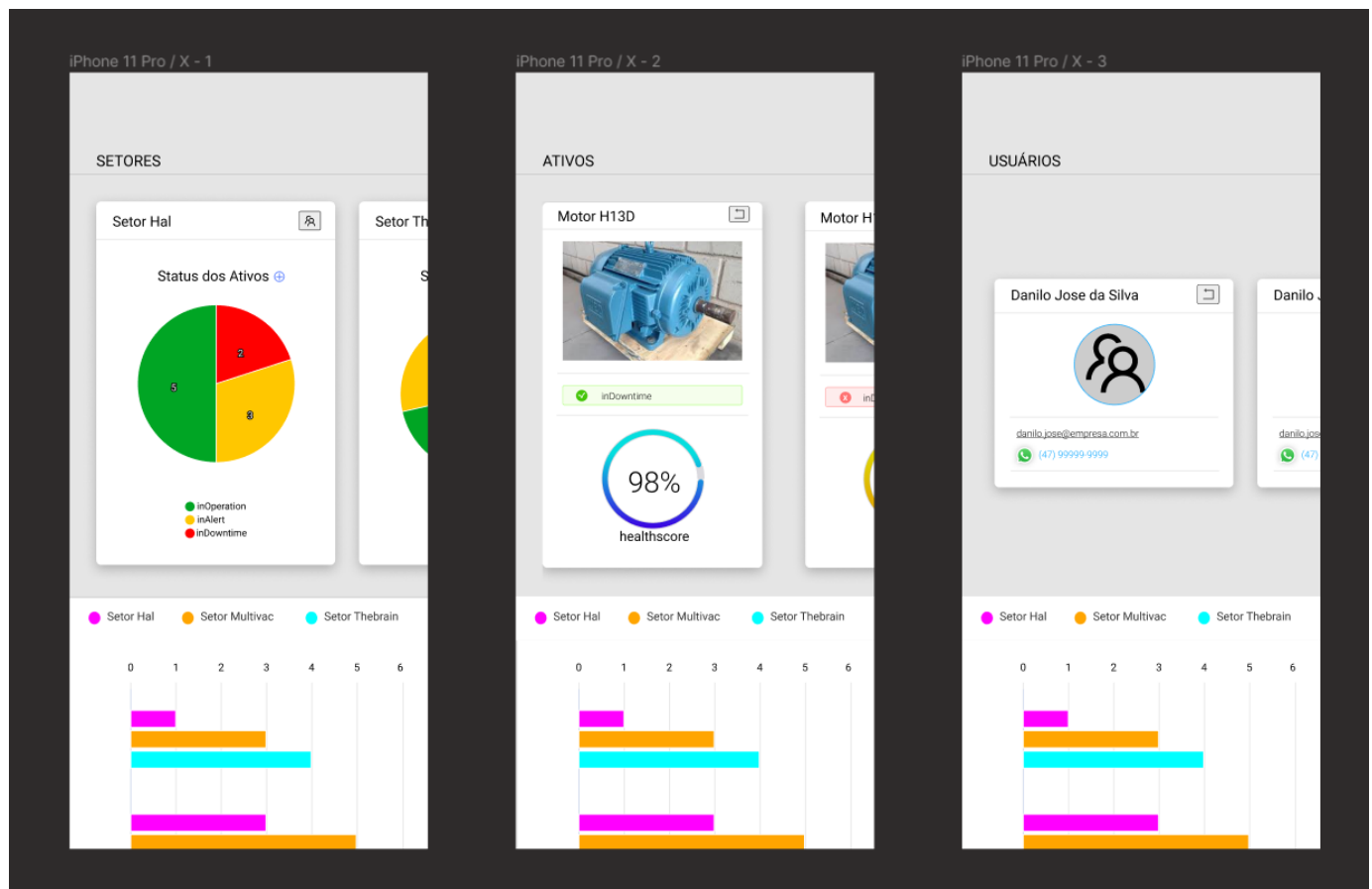


Figura 7. Estudo de Interface de Usuário relativo ao frontend do software disponível AQUI.

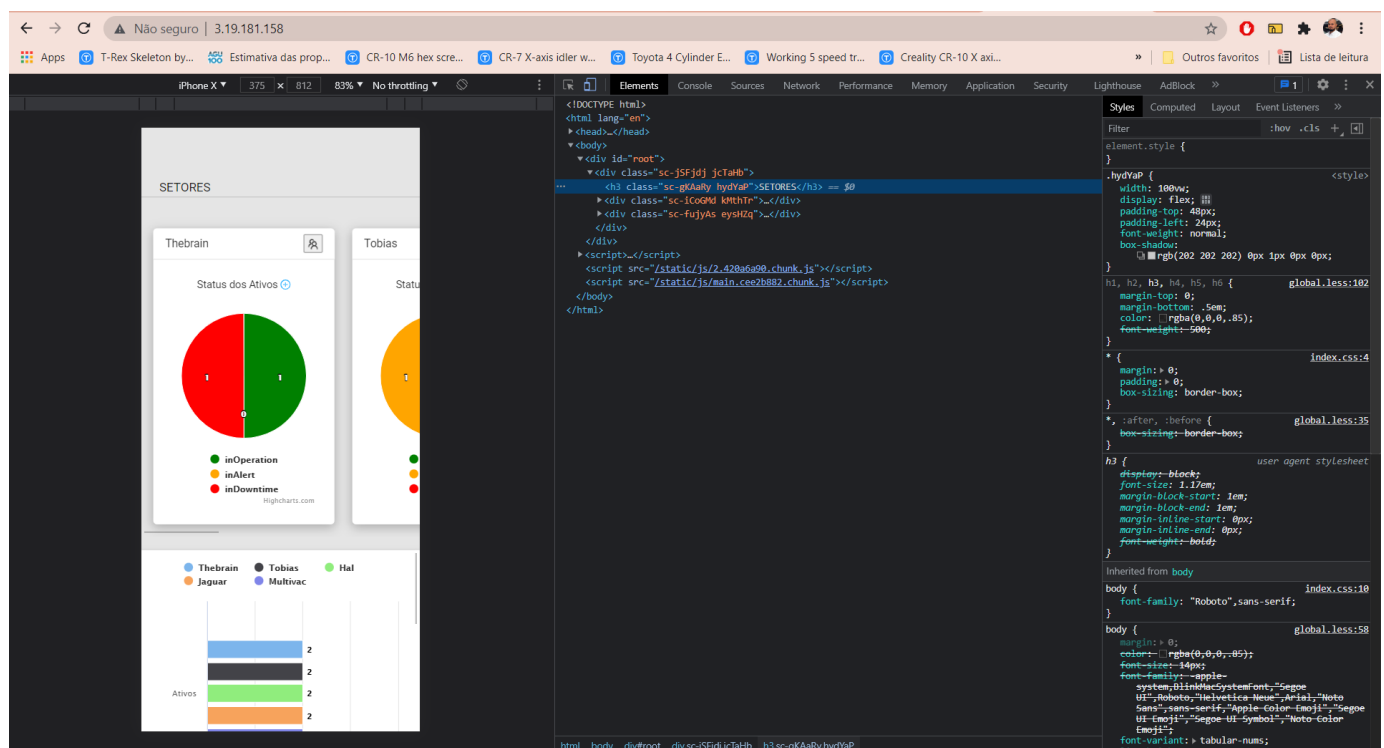


Figura 8. Aplicação acessada em dispositivos mobile.

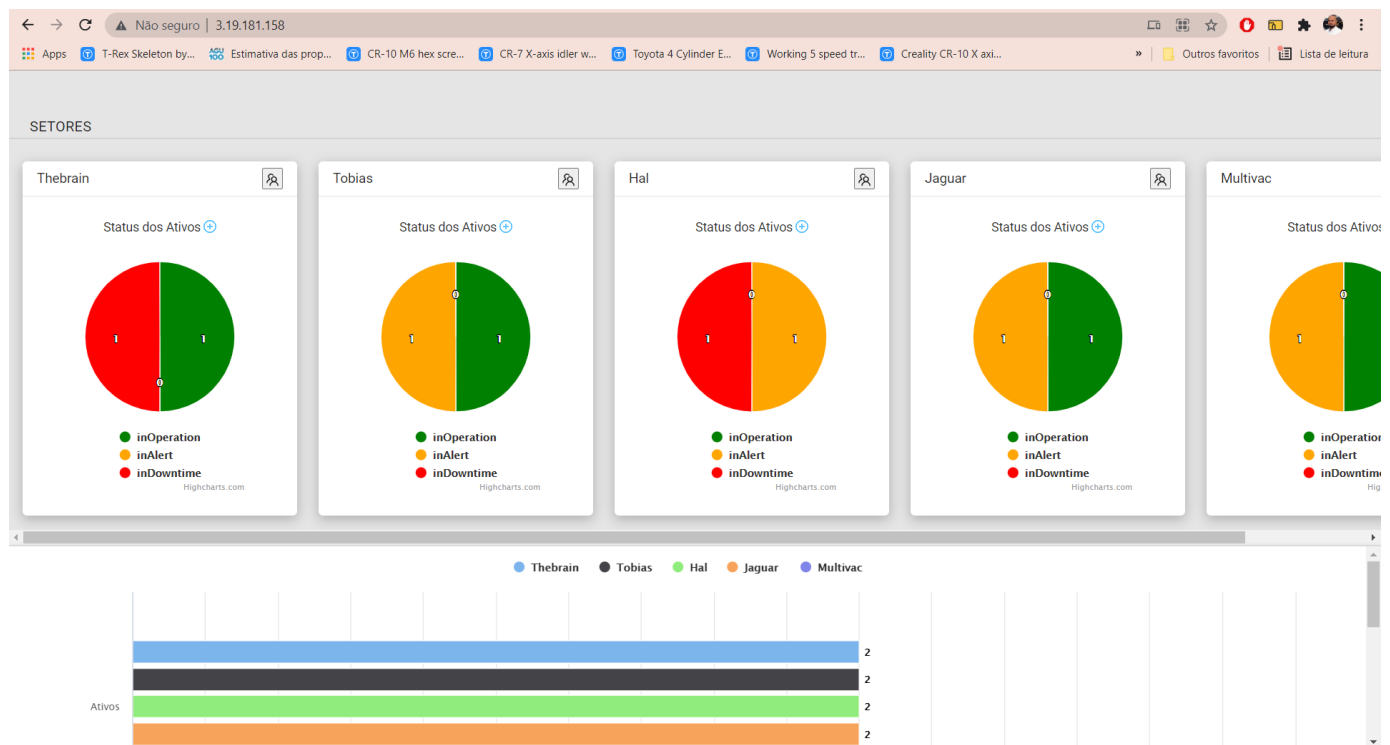


Figura 9. Aplicação acessada em desktop.

APÊNDICE A

A. *unit_test_api.py*

```

1 import unittest
2 import requests
3
4 class TestApiMethods(unittest.TestCase):
5
6     URL = "http://3.19.181.158/"
7     list_id_unidades = []
8     list_id_usuarios = []
9     list_id_ativos = []
10
11     def test_get_all(self):
12         request = requests.get(f'{self.URL}/api/tudo')
13         self.assertEqual(request.status_code, 200)
14
15     def test_get_unidades(self):
16         request = requests.get(f'{self.URL}/api/unidades').json()
17         self.assertEqual(request["status_code"], 200)
18
19         for unidade in request["items"]:
20             self.list_id_unidades.append(unidade["id"])
21
22     def test_post_unidade_existente(self):
23         for i in range(len(self.list_id_unidades)):
24             request = requests.post(f'{self.URL}/api/unidades', json={
25                 "id": self.list_id_unidades[i],
26                 "name": "Test",
27                 "companyId": 0
28             }).json()
29
30             self.assertEqual(request["status_code"], 404)
31

```

```
32 def test_get_usuarios(self):
33     request = requests.get(f'{self.URL}/api/usuarios').json()
34     self.assertEqual(request["status_code"], 200)
35
36     for usuario in request["items"]:
37         self.list_id_usuarios.append(usuario["matricula"])
38
39 def test_post_usuario_existente(self):
40     for i in range(len(self.list_id_usuarios)):
41         request = requests.post(f'{self.URL}/api/usuarios', json={
42             "matricula": self.list_id_usuarios[i]
43         }).json()
44
45         self.assertEqual(request["status_code"], 404)
46
47 def test_get_ativos(self):
48     request = requests.get(f'{self.URL}/api/ativos').json()
49     self.assertEqual(request["status_code"], 200)
50
51     for ativo in request["items"]:
52         self.list_id_ativos.append(ativo["id"])
53
54 def test_post_ativo_existente(self):
55     for i in range(len(self.list_id_ativos)):
56         request = requests.post(f'{self.URL}/api/ativos', json={
57             "id": self.list_id_ativos[i]
58         }).json()
59
60         self.assertEqual(request["status_code"], 404)
61
62 if __name__ == '__main__':
63     unittest.main()
```