Criação de novos testes unitários

DEVinHouse

Parcerias para desenvolver a sua carreira

SENAI

<LAB365>

AGENDA

- Configuração de PyTest.Fixtures
- Novos testes unitários

Configuração de PyTest.Fixtures

```
# No arquivo conftest.py, adicione os seguintes blocos de código, para podermos testar
endpoints autenticados:


@pytest.fixture
def logged_in_client(client):
    data = {
        'email': "luislopes@gmail.com",
        'password': "123Mudar!"
    }
    response = client.post('user/login', data=json.dumps(data), headers=headers)
    return response.json['token']
```

Configuração de PyTest.Fixtures

```
# No arquivo conftest.py, adicione os seguintes blocos de código, para impedir de
salvar dados invalidados novamente:


@pytest.fixture(scope="function", autouse=True)
def session(app):
    with app.app_context():
        connection = _db.engine.connect()
        transaction = connection.begin()
        options = dict(bind=connection, binds={})
        sess = _db.create_scoped_session(options=options)
        sess.begin_nested()
```

Configuração de PyTest.Fixtures

```
@event.listens_for(sess(), 'after_transaction_end')
        def restart_savepoint(sess2, trans):
            if trans.nested and not trans._parent.nested:
                sess2.expire_all()
                sess2.begin_nested()
        _db.session = sess
        yield sess
        sess.remove()
        transaction.rollback()
        connection.close()
```

Novos testes unitários

```
# Criar uma pasta chama users dentro da pasta tests e criar um arquivo chamado
test_create para criarmos os casos de testes para o endpoint user/create:


from flask import json
import random


mimetype = 'application/json'
headers = {
    'Content-Type': mimetype,
    'Accept': mimetype
}
url = '/user/create'
```

Novos testes unitários

```
def test_create_user_not_authorized(client):
    response = client.post(url, headers=headers)
    assert response.status_code == 403
    assert response.json['error'] == "Você não tem permissão"
```

Novos testes unitários

```python
def test_create_user_authorized_and_missing_parameters(client, logged_in_client):
    headers['Authorization'] = f"Bearer {logged_in_client}"
    keys = ["city_id", "name", "age", "email", "password"]
    keys_not_have_in_request = keys.pop(random.randrange(len(keys)))
    data = { 'email': "loivaci.lopes@example.com", 'city_id': 2,'name': 'Luis Lopes',
        'age': 30, 'password': '123Mudar!'
    }
    del data[keys_not_have_in_request]
    response = client.post(url, data=json.dumps(data), headers=headers)
    assert response.status_code == 422
    assert response.json['error'] == f"Está faltando o item ['{keys_not_have_in_request}']"
```

Novos testes unitários

```python
def test_create_user_failed_exist_user(client, logged_in_client):
    headers['Authorization'] = f"Bearer {logged_in_client}"
    data = {
        'email': "luislopes@gmail.com",
        'city_id': 2,
        'name': 'Luis Lopes',
        'age': 30,
        'password': '123Mudar!'
    }
    response = client.post(url, data=json.dumps(data), headers=headers)
    assert response.status_code == 400
    assert response.json['message'] == "Não foi possível criar o usuário"
```

Novos testes unitários

```
def test_create_user_success(client, logged_in_client):
    headers['Authorization'] = f"Bearer {logged_in_client}"
    data = {
        'email': "loivaci.lopes@example.com",
        'city_id': 2,
        'name': 'Luis Lopes',
        'age': 30,
        'password': '123Mudar!'
    }
    response = client.post(url, data=json.dumps(data), headers=headers)
    assert response.status_code == 201
    assert response.json['message'] == "Usuário foi criado com sucesso."
```

Novos testes unitários

```
# Ainda na pasta users dentro da pasta tests, iremos criar um arquivo chamado
test_login para criarmos os casos de testes para o endpoint user/login:


from flask import json


mimetype = 'application/json'
headers = {
    'Content-Type': mimetype,
    'Accept': mimetype
}
url = '/user/login'
```

Novos testes unitários

```
def test_login_failed_error_password(client):
    data = {
        'email': "luislopes@gmail.com",
        'password': "123123434"
    }
    response = client.post(url, data=json.dumps(data), headers=headers)
    assert response.status_code == 401
    assert response.json['error'] == "Suas credênciais estão incorretas!"
```

Novos testes unitários

```
def test_login_failed_error_email(client):
    data = {
        'email': "loivaci.lopes@example",
        'password': "123Mudar!"
    }
    response = client.post('/user/login', data=json.dumps(data), headers=headers)
    assert response.status_code == 500
    assert response.json['error'] == "404 Not Found: The requested URL was not found
on the server. If you entered the URL manually please check your spelling and try
again."
```

Novos testes unitários

```
def test_login_failed_error_missing_password(client):
    data = { 'email': "loivaci.lopes@example" }
    response = client.post('/user/login', data=json.dumps(data), headers=headers)
    assert response.status_code == 422
    assert response.json['error'] == "Está faltando o item ['password']"


def test_login_failed_error_missing_email(client):
    data = { 'password': "123Mudar!" }
    response = client.post('/user/login', data=json.dumps(data), headers=headers)
    assert response.status_code == 422
    assert response.json['error'] == "Está faltando o item ['email']"
```

Novos testes unitários

```python
def test_login_failed_error_missing_all_parameters(client):
    data = {}
    response = client.post('/user/login', data=json.dumps(data), headers=headers)
    assert response.status_code == 422
    assert response.json['error'] == "Está faltando o item ['email', 'password']"


def test_login_success(client):
    data = { 'email': "luislopes@gmail.com", 'password': "123Mudar!" }
    response = client.post(url, data=json.dumps(data), headers=headers)
    assert response.status_code == 200
    assert "token" in response.json
```

DEVinHouse

Parcerias para desenvolver a sua carreira

OBRIGADO!

SENAI

<LAB365>