

# Traffic Incident Management API

## Visão Geral

Esta aplicação é um conjunto de testes automatizados para a Traffic Incident Management API. Ela utiliza o framework Cucumber para definir cenários de teste em linguagem natural e o Rest Assured para realizar requisições HTTP e validar respostas.

Link da documentação da API hospedada na Microsoft Azure:

<https://traffic-incident-api-dev-dtbtfvg2e7e7a8eq.eastus2-01.azurewebsites.net/swagger-ui/index.html>

Link do repositório do GitHub da API:

<https://github.com/viniciusleone/traffic-incident-management-api>

Link do repositório do GitHub da aplicação:

[https://github.com/viniciusleone/testes-bdd-ger\\_de\\_traf](https://github.com/viniciusleone/testes-bdd-ger_de_traf)

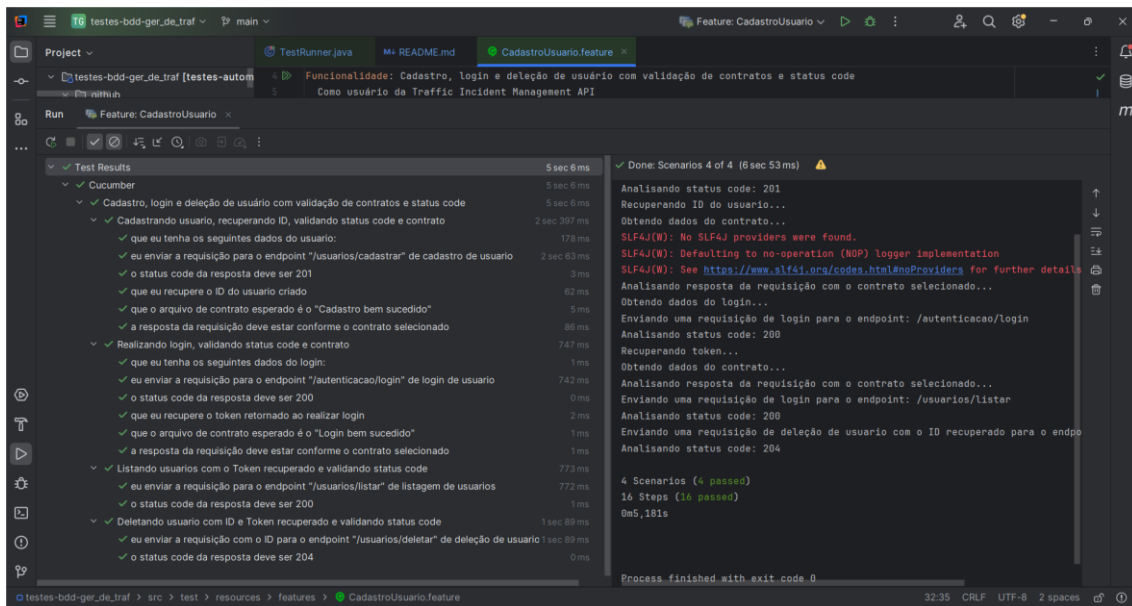
**OBS:** Para que as funcionalidades de Cadastro de usuário e de Registro de Acidentes funcionem, é preciso que seja realizado o cadastro e login de usuário para a obtenção do token de autenticação conforme descrito nos cenários.

---

## Descrição dos Cenários de Teste em Gherkin

### **Funcionalidade: Cadastro, login e deleção de usuário com validação de contratos e status code**

Esta funcionalidade contém três cenários de teste que garantem o funcionamento correto do processo de cadastro, login e deleção de usuários na Traffic Incident Management API. Cada cenário é descrito em termos de pré-condições, ações e resultados esperados.



## Cenário 1: Cadastrando usuário, recuperando ID, validando status code e contrato

- **Dado** que eu tenha os seguintes dados do usuário:
  - email
  - senha
  - role
- **Quando** eu enviar a requisição para o endpoint `"/usuarios/cadastrar"` de cadastro de usuário
- **Então** o status code da resposta deve ser 201 (Criado)
- **Dado** que eu recupere o ID do usuário criado (Recupera o ID para uso no Cenário 3)
- **E** que o arquivo de contrato esperado é o "Cadastro bem-sucedido"
- **Então** a resposta da requisição deve estar conforme o contrato selecionado

## Cenário 2: Realizando login, validando status code e contrato

- **Dado** que eu tenha os seguintes dados do login:
  - email
  - senha
- **Quando** eu enviar a requisição para o endpoint `"/autenticacao/login"` de login de usuário
- **Então** o status code da resposta deve ser 200 (OK)
- **Dado** que eu recupere o token retornado ao realizar login (Recupera o Token para uso no Cenário 3)
- **E** que o arquivo de contrato esperado é o "Login bem-sucedido"

- **Então** a resposta da requisição deve estar conforme o contrato selecionado

### Cenário 3: Deletando usuário com ID e Token recuperado e validando status code

- **Quando** eu enviar a requisição com o ID para o endpoint "/usuarios/deletar" de deleção de usuário (Aqui é enviado o token de autenticação e o ID do usuário)
- **Então** o status code da resposta deve ser 204 (Sem Conteúdo)

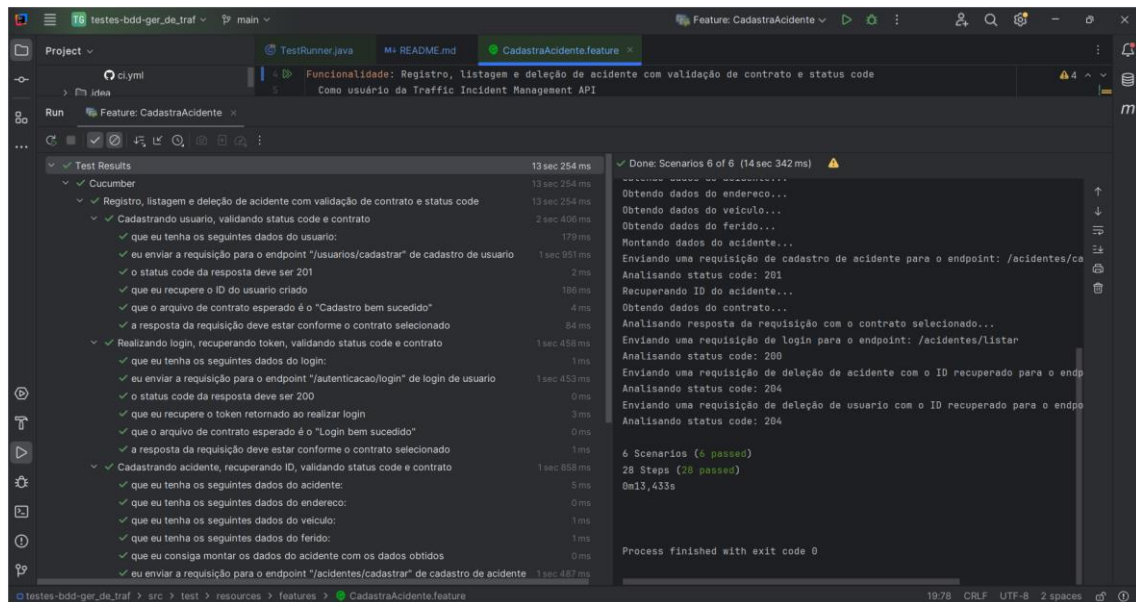
### Conclusão

Esses cenários de teste são fundamentais para garantir que as operações de cadastro, login e deleção de usuários funcionem corretamente, assegurando a integridade dos dados e a conformidade com os contratos esperados.

## Funcionalidade: Registro, listagem e deleção de acidente com validação de contrato e status code

Esta funcionalidade contém vários cenários de teste que garantem o funcionamento correto do processo de registro de acidentes, listagem e deleção de acidentes na Traffic Incident Management API. Cada cenário é descrito em termos de pré-condições, ações e resultados esperados.

Para que seja possível realizar as requisições é necessário cadastrar um usuário e realizar o login para a obtenção do token de autenticação. Ao final dos testes o usuário é deletado.



### Cenário 1: Cadastrando usuário, validando status code e contrato

- **Dado** que eu tenha os seguintes dados do usuário:
  - email
  - senha

- role
- **Quando** eu enviar a requisição para o endpoint "/usuarios/cadastrar" de cadastro de usuário
- **Então** o status code da resposta deve ser 201 (Criado)
- **Dado** que eu recupere o ID do usuário criado (Recupera o ID para uso no Cenário 6)
- **E** que o arquivo de contrato esperado é o "Cadastro bem-sucedido"
- **Então** a resposta da requisição deve estar em conformidade com o contrato selecionado

### **Cenário 2: Realizando login, recuperando token, validando status code e contrato**

- **Dado** que eu tenha os seguintes dados do login:
  - email
  - senha
- **Quando** eu enviar a requisição para o endpoint "/autenticacao/login" de login de usuário
- **Então** o status code da resposta deve ser 200 (OK)
- **Dado** que eu recupere o token retornado ao realizar login (Recupera o Token para uso em todos os Cenários seguintes)
- **E** que o arquivo de contrato esperado é o "Login bem-sucedido"
- **Então** a resposta da requisição deve estar em conformidade com o contrato selecionado

### **Cenário 3: Cadastrando acidente, recuperando ID, validando status code e contrato**

- **Dado** que eu tenha os seguintes dados do acidente:
  - dataHora
  - gravidade
- **E** que eu tenha os seguintes dados do endereço:
  - logradouro
  - numero
  - bairro
  - cep
  - cidade
  - estado
- **E** que eu tenha os seguintes dados do veículo:

- placa
  - modelo
  - ano
  - cor
- **E** que eu tenha os seguintes dados do ferido:
  - nome
  - cpf
  - gravidade
- **E** que eu consiga montar os dados do acidente com os dados obtidos
- **Quando** eu enviar a requisição para o endpoint "/acidentes/cadastrar" de cadastro de acidente (Aqui é enviado o token de autenticação)
- **Então** o status code da resposta deve ser 201 (Criado)
- **Dado** que eu recupere o ID do acidente registrado (Recupera o ID para uso no Cenário 5)
- **E** que o arquivo de contrato esperado é o "Acidente cadastrado"
- **Então** a resposta da requisição deve estar conforme o contrato selecionado

#### **Cenário 4: Listando acidentes com token recuperado e validando status code**

- **Quando** eu enviar a requisição para o endpoint "/acidentes/listar" de listagem de acidentes (Aqui é enviado o token de autenticação)
- **Então** o status code da resposta deve ser 200 (OK)

#### **Cenário 5: Deletando acidente com token e ID recuperados e validando status code**

- **Quando** eu enviar a requisição com o ID para o endpoint "/acidentes/deletar" de deleção de acidente (Aqui é enviado o token de autenticação e o ID do acidente)
- **Então** o status code da resposta deve ser 204 (Sem Conteúdo)

#### **Cenário 6: Deletando usuário com ID recuperado e validando status code**

- **Quando** eu enviar a requisição com o ID para o endpoint "/usuarios/deletar" de deleção de usuário (Aqui é enviado o token de autenticação e o ID do usuário)
- **Então** o status code da resposta deve ser 204 (Sem Conteúdo)

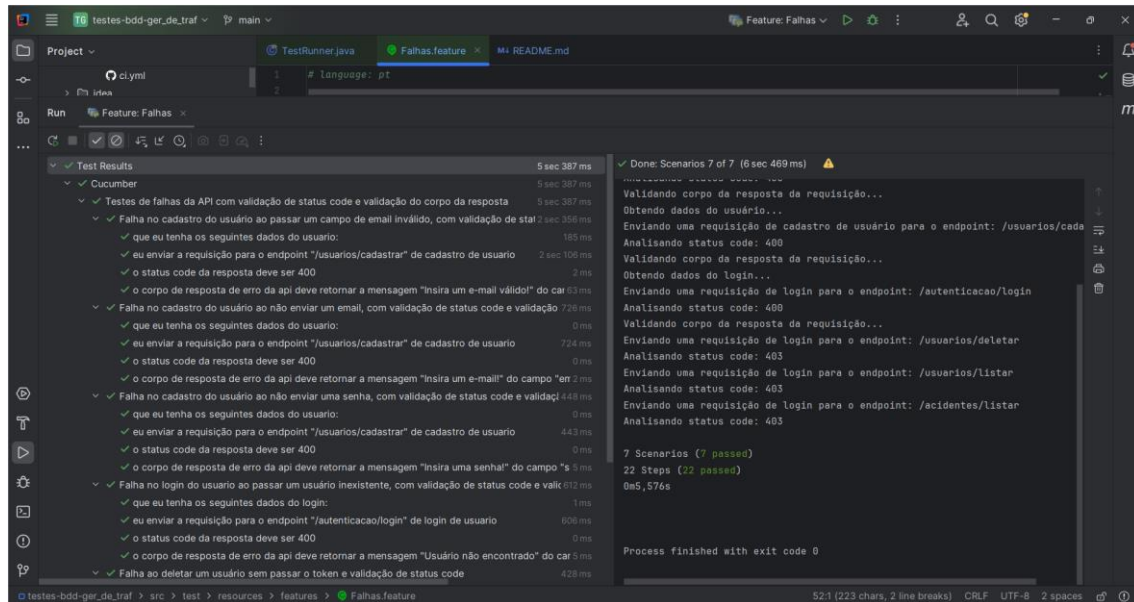
#### **Conclusão**

Esses cenários de teste são essenciais para garantir que as operações de registro, listagem e deleção de acidentes funcionem corretamente, assegurando a integridade dos dados e a conformidade com os contratos esperados.

---

## Funcionalidade: Testes de falhas da API com validação de status code e validação do corpo da resposta

Esta funcionalidade contém vários cenários de teste que garantem que a Traffic Incident Management API responda corretamente a requisições incorretas. Cada cenário é descrito em termos de pré-condições, ações e resultados esperados.



### Cenário 1: Falha no cadastro do usuário ao passar um campo de email inválido, com validação de status code e validação de mensagem de erro

- **Dado** que eu tenha os seguintes dados do usuário:
  - email (inválido)
  - senha
  - role
- **Quando** eu enviar a requisição para o endpoint `/usuarios/cadastrar` de cadastro de usuário
- **Então** o status code da resposta deve ser 400 (Requisição Inválida)
- **E** o corpo de resposta de erro da API deve retornar a mensagem "Insira um e-mail válido!" do campo "email"

### Cenário 2: Falha no cadastro do usuário ao não enviar um email, com validação de status code e validação de mensagem de erro

- **Dado** que eu tenha os seguintes dados do usuário:
  - email (vazio)
  - senha
  - role

- **Quando** eu enviar a requisição para o endpoint "/usuarios/cadastrar" de cadastro de usuário
- **Então** o status code da resposta deve ser 400 (Requisição Inválida)
- **E** o corpo de resposta de erro da API deve retornar a mensagem "Insira um e-mail!" do campo "email"

**Cenário 3: Falha no cadastro do usuário ao não enviar uma senha, com validação de status code e validação de mensagem de erro**

- **Dado** que eu tenha os seguintes dados do usuário:
  - email
  - senha (vazio)
  - role
- **Quando** eu enviar a requisição para o endpoint "/usuarios/cadastrar" de cadastro de usuário
- **Então** o status code da resposta deve ser 400 (Requisição Inválida)
- **E** o corpo de resposta de erro da API deve retornar a mensagem "Insira uma senha!" do campo "senha"

**Cenário 4: Falha no login do usuário ao passar um usuário inexistente, com validação de status code e validação de mensagem de erro**

- **Dado** que eu tenha os seguintes dados do login:
  - email (inexistente)
  - senha
  - role
- **Quando** eu enviar a requisição para o endpoint "/autenticacao/login" de login de usuário
- **Então** o status code da resposta deve ser 400 (Requisição Inválida)
- **E** o corpo de resposta de erro da API deve retornar a mensagem "Usuário não encontrado" do campo "message"

**Cenário 5: Falha ao deletar um usuário sem passar o token e validação de status code**

- **Quando** eu enviar a requisição para o endpoint "/usuarios/deletar" de deleção de usuário
- **Então** o status code da resposta deve ser 403 (Proibido)

**Cenário 6: Falha ao listar usuários sem passar o token e validação de status code**

- **Quando** eu enviar a requisição para o endpoint "/usuarios/listar" de listagem de usuários
- **Então** o status code da resposta deve ser 403 (Proibido)

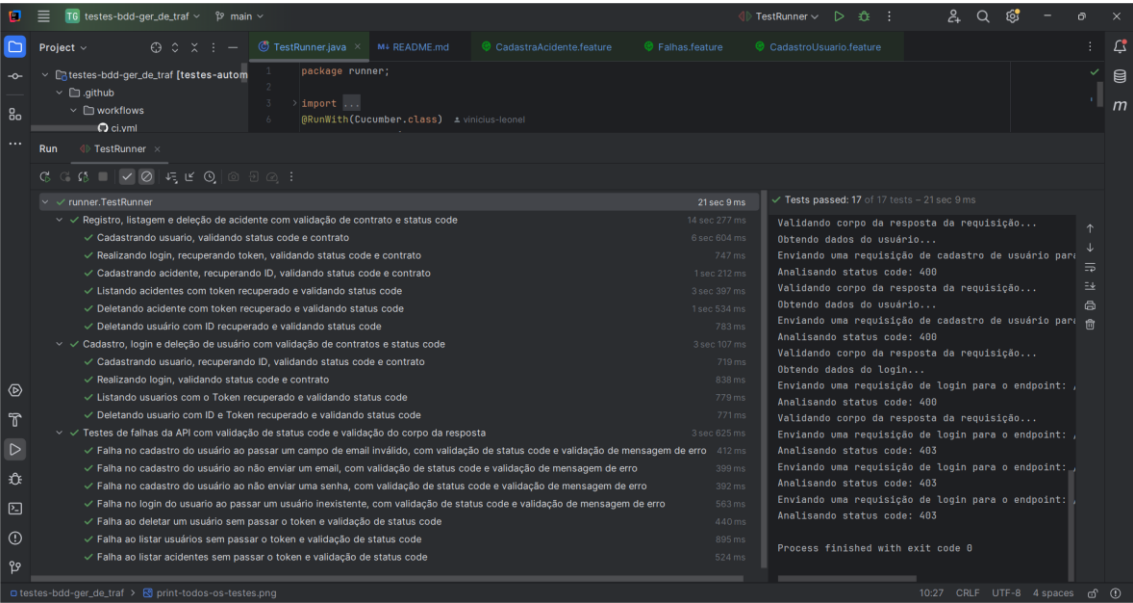
Cenário 7: Falha ao listar acidentes sem passar o token e validação de status code

- Quando eu enviar a requisição para o endpoint "/acidentes/listar" de listagem de acidentes
- Então o status code da resposta deve ser 403 (Proibido)

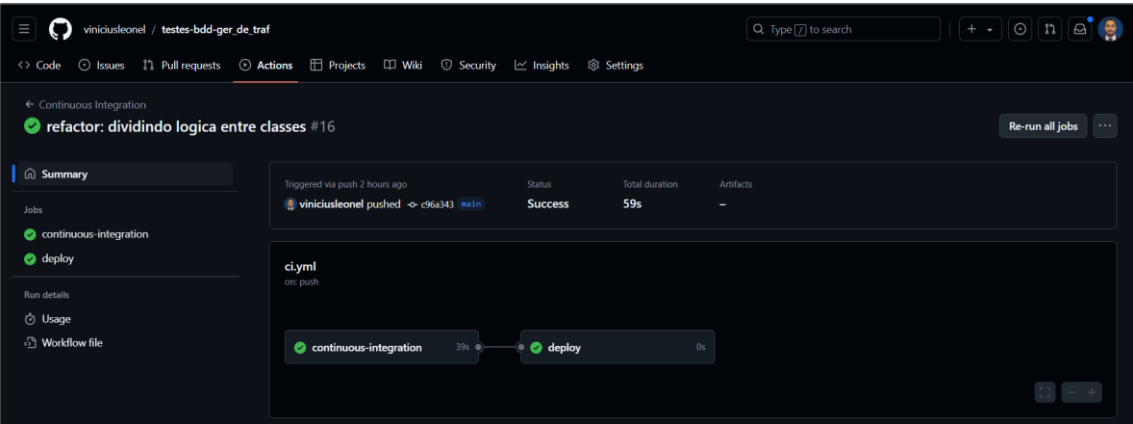
Conclusão

Esses cenários de teste são cruciais para garantir que a API responda adequadamente a requisições incorretas, assegurando que os status codes e as mensagens de erro estejam em conformidade com o esperado.

Print de todos os testes concluídos:



Print GitHub Actions:





# Print Cucumber Reports:

17 PASSED

100% passed

17 executed

27 seconds ago

last run

17 seconds

duration

Windows 11

OpenJDK 64-Bit Server VM 22.0.2+9-70

cucumber-jvm 7.18.1

Search with text or @tags

You can search with plain text or [Cucumber Tag Expressions](#) to filter the output

> file:///C:/Users/vinicius/Desktop/my-portfolio/iaap/testes-bdd-ger\_de\_traf/src/test/resources/features/CadastreAcidente.feature

> file:///C:/Users/vinicius/Desktop/my-portfolio/iaap/testes-bdd-ger\_de\_traf/src/test/resources/features/CadastreUsuario.feature

> file:///C:/Users/vinicius/Desktop/my-portfolio/iaap/testes-bdd-ger\_de\_traf/src/test/resources/features/Falha.feature

@regressivo

**Funcionalidade:** Testes de falhas da API com validação de status code e validação do corpo da resposta

Como usuário da Traffic Incident Management API  
Quero testar as falhas ao realizar requisições incorretas  
Para garantir que as respostas da API estejam em conformidade com o esperado

**Cenário:** Falha no cadastro do usuário ao passar um campo de email inválido, com validação de status code e validação de mensagem de erro

✓ Dado que eu tenha os seguintes dados do usuário:

campo | valor

# Print Cl.yml

testes-bdd-ger\_de\_traf

main

TestRunner

TestRunner.java

ci.yml

README.md

```
1 name: Continuous Integration
2 on:
3   pull_request: # Aciona para pull requests
4   push: # Aciona para pushes
5   branches:
6     - '*' # Isso significa qualquer branch
7 jobs:
8   continuous-integration:
9     runs-on: ubuntu-latest
10    steps:
11      - name: Checkout code
12        uses: actions/checkout@v3
13      - name: Set up JDK
14        uses: actions/setup-java@v2
15        with:
16          java-version: '22'
17          distribution: 'adopt'
18      - name: Build and test
19        run: mvn clean test #Esse comando vai rodar os testes
20  deploy:
21    runs-on: ubuntu-latest
22    needs: continuous-integration # Esse job depende do anterior
23    if: success() # Esse job só será executado se o job anterior for bem-sucedido
24    steps:
25      - name: Deploy application
26        run: |
27          echo "Deploying application..."
28
```

testes-bdd-ger\_de\_traf

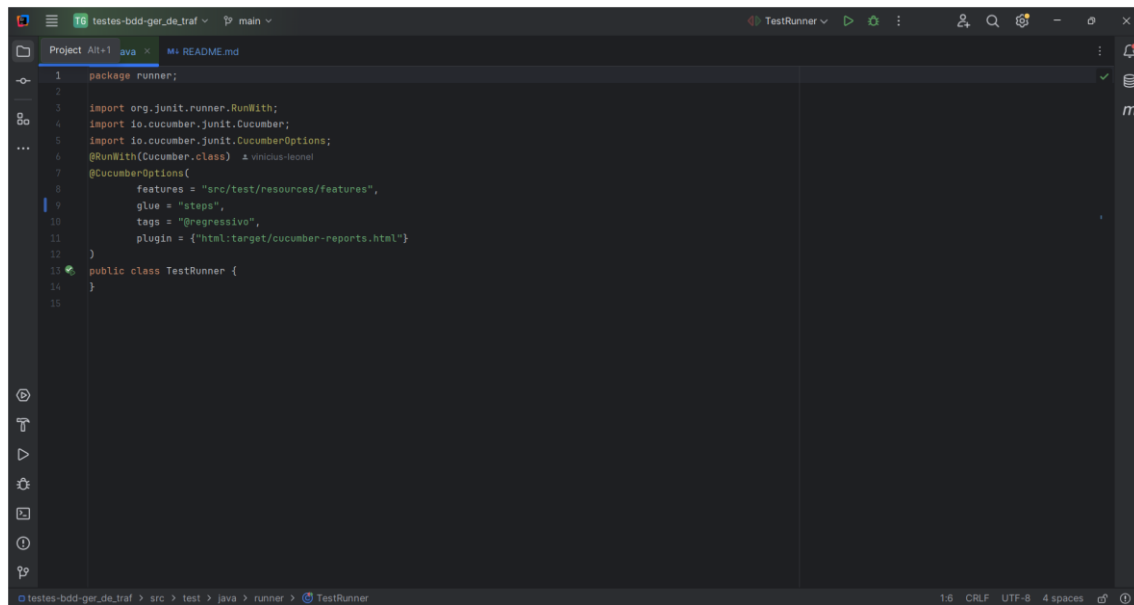
github

workflows

ci.yml

28:11 CRLF UTF-8 2 spaces Schema: github-workflow.json

## Print TestRunner

A screenshot of an IDE window titled 'testes-bdd-ger\_de\_traf' with a 'main' tab. The editor shows a Java file named 'TestRunner.java'. The code is as follows:

```
1 package runner;
2
3 import org.junit.runner.RunWith;
4 import io.cucumber.junit.Cucumber;
5 import io.cucumber.junit.CucumberOptions;
6 @RunWith(Cucumber.class) // vinicius-leonel
7 @CucumberOptions(
8     features = "src/test/resources/features",
9     glue = "steps",
10    tags = "@regressivo",
11    plugin = {"html:target/cucumber-reports.html"}
12 )
13 public class TestRunner {
14 }
15
```

The IDE interface includes a sidebar on the left with icons for Explorer, Search, Run and Debug, and Test Explorer. The bottom status bar shows '1.6 CRLF UTF-8 4 spaces'.

Gustavo Moreira da Silva  
gustavomoreira565@gmail.com  
RM98715

Kaique Teixeira Monteiro  
kaique.tmonteiro@hotmail.com  
RM551020

Otto Balieiro F. Andrade  
ottobfa@gmail.com  
RM98390

Pedro Lopes Ribeiro  
pedrolopes\_40@hotmail.com  
RM99938

Vinícius Leonel Pereira Silva  
viniciuslps.cms@gmail.com  
RM97971