

Relatório Geral de Testes  
Relatório de testes do Projeto Upload

Testador: Marcus Vinicius Carvalho da Silva  
URL do site testado: <https://automacaocombatista.herokuapp.com/>  
<https://github.com/MarcusVini476/ProjetoUpload>

versão 1.1

# Índice

1. Introdução.....	3
• 1.1. Objetivo.....	3
2. Resumo dos resultados de teste.....	3
3. Cobertura do código.....	4
• 3.1 TesteDeUpload.feature e POM.xml.....	4
• 3.2 BaseTest.java e hooks.java.....	6
• 3.3 Web.java.....	7
• 3.4 WebApplication.java.....	7
• 3.5 RobotSelenium.java.....	8
• 3.6 Drivers.java.....	8
• 3.7 Runner.java.....	9
• 3.8 Inicial.java.....	10
• 3.9 Treinamento.java.....	12
• 3.10 Upload.java.....	15
• 3.11 RealizaUpload.java.....	17
4. Resultados.....	20
5. Tendências de defeitos.....	20
6. Ações Sugeridas.....	20

## 1. INTRODUÇÃO

### 1.1. OBJETIVO

Este relatório tem como função apresentar os resultados da automação de Upload utilizado Selenium, junit e Cucumber usando a biblioteca do selenium WebDriver, como navegador o google chrome e também estaremos mostrando evidências, sugestões de melhorias e as ferramentas utilizadas.

## 2. RESUMO DOS RESULTADOS DE TESTES

Utilizando o método BDD (Behavior Driven Development) como forma de escrita para realizar os testes, foi feita a entrada no site Automação com Batista e as interações com seus componentes. Tendo apenas 4 passos a serem seguidos, todos passaram e efetuaram suas funções.

```
Contexto: Acessar a Pagina de Treinamento Automacao Web # C:/Users/Marcus Silva/eclipse-workspace/ProjetoUpload/ProjetoUpload/src/test/resources/TesteDeUpload.feature:
  Dado que acesso Automacao com Batista # InicialSteps.queAcessoAutomacaoComBatista()
terminando automacao

@web @RealizarUpload
Cenário: Realizar o Upload de um arquivo txt presente na pasta ParaUpload # C:/Users/Marcus Silva/eclipse-workspace/ProjetoUpload/ProjetoUpload/src/test/resou
  E navego ate "Outros" e clico em "Upload de arquivos" # TreinamentoSteps.navegoAteEClicoEm(String,String)
  E clico em "File" # UploadSteps.clicoEm(String)
  Entao devo fazer o Upload do arquivo "LEIAME.txt" que esta na pasta "ParaUpload" # RealizaUploadSteps.devoFazerOUploadDoArquivoQueEstaNaPasta(String,String)

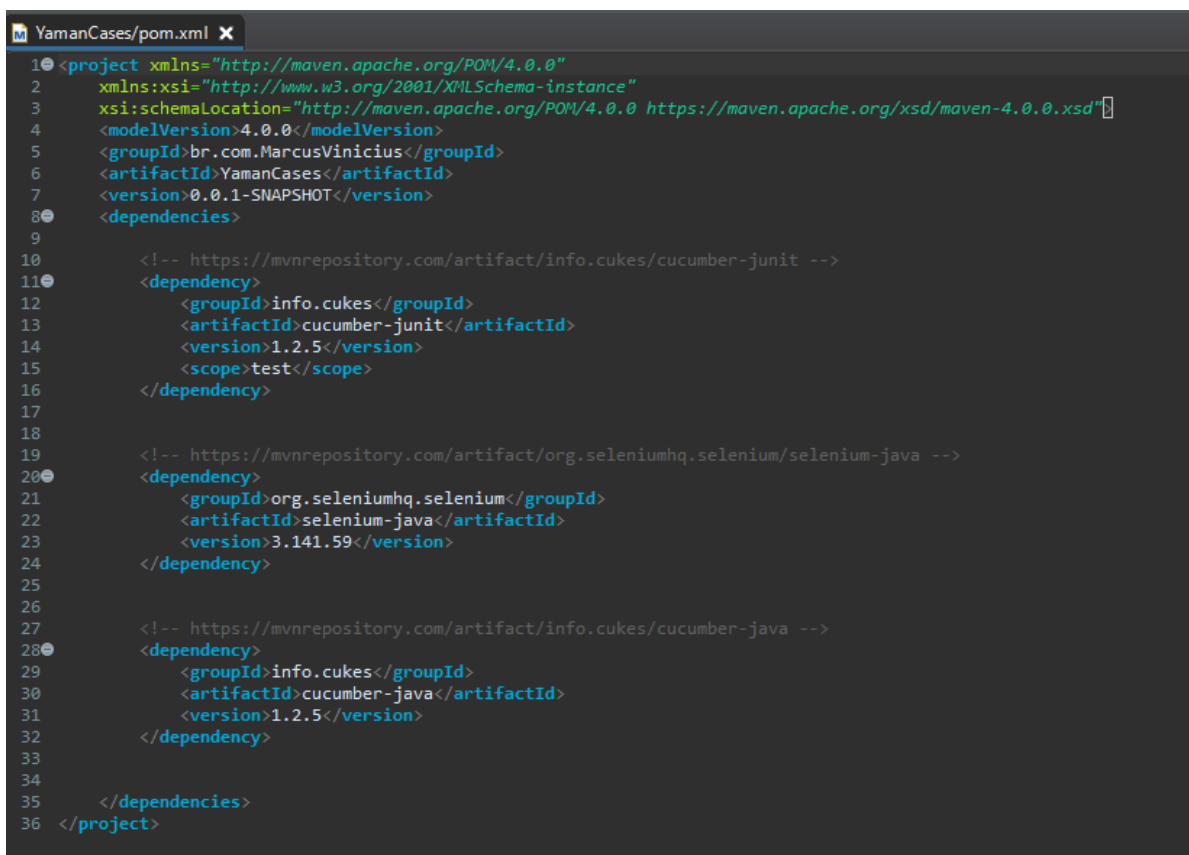
1 Scenarios (1 passed)
4 Steps (4 passed)
0m9,117s
```

## 3. COBERTURA DO CÓDIGO

Na cobertura de códigos será mostrado os prints das classes e quais suas funções no projeto.

### 3.1. TesteDeUpload.feature e POM.xml

O pom.xml é onde está todas as dependências deste projeto, no caso foi usado Cucumber-junit na versão 1.25, Cucumber-java na versão 1.2.5 e selenium-java na versão 3.141.59.



```
1 <project xmlns="http://maven.apache.org/POM/4.0.0"
2   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
4   <modelVersion>4.0.0</modelVersion>
5   <groupId>br.com.MarcusVinicius</groupId>
6   <artifactId>YamanCases</artifactId>
7   <version>0.0.1-SNAPSHOT</version>
8   <dependencies>
9
10    <!-- https://mvnrepository.com/artifact/info.cukes/cucumber-junit -->
11    <dependency>
12      <groupId>info.cukes</groupId>
13      <artifactId>cucumber-junit</artifactId>
14      <version>1.2.5</version>
15      <scope>test</scope>
16    </dependency>
17
18    <!-- https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-java -->
19    <dependency>
20      <groupId>org.seleniumhq.selenium</groupId>
21      <artifactId>selenium-java</artifactId>
22      <version>3.141.59</version>
23    </dependency>
24
25    <!-- https://mvnrepository.com/artifact/info.cukes/cucumber-java -->
26    <dependency>
27      <groupId>info.cukes</groupId>
28      <artifactId>cucumber-java</artifactId>
29      <version>1.2.5</version>
30    </dependency>
31
32  </dependencies>
33
34 </project>
```

O “TesteDeUpload.feature” é o arquivo BDD (Behavior Driven Development) e nele está todos os passos a serem seguidos a história de usuário, transformada em contexto da funcionalidade, o cenário a ser testado e as tags @web @RealizarUpload e @Upload, são reconhecidas pelo Junit ao ler o BDD.

```
1 #language: pt
2 #encoding: UTF-8
3 #author: MarcusVinicius
4 #date: 27/05/2021
5 #version: 1.0
6 @Upload
7 ● Funcionalidade: Upload de um documentos para o site Automacao com Batista
8
9 ● Contexto: Acessar a Pagina de Treinamento Automacao Web
10   Dado que acesso Automacao com Batista
11
12   @web @RealizarUpload
13 ● Cenario: Realizar o Upload de um arquivo presente na pasta do projeto ParaUpload
14   E navego ate "Outros" e clico em "Upload de arquivos"
15   E clico em "File"
16   Entao devo fazer o Upload do arquivo "LEIAME.txt" que esta na pasta "ParaUpload"
17
```

## 3.2 BaseTest.java e hooks

Na BaseTest.java será estabelecido a inicialização, o fechamento do navegador e o tempo de espera para realizar as funções.

```
BaseTest.java X
1 package br.com.MarcusVinicius.configuracoes;
2
3
4
5 import org.openqa.selenium.WebDriver;
9
10 public abstract class BaseTest {
11
12     protected static WebDriver navegador;
13     protected static WebDriverWait espera;
14
15     protected void inicializarAplicacaoWeb(WebApplication webApplication, String url) {
16         if (navegador != null) {
17             navegador.quit();
18         }
19         navegador = webApplication.getWebDriver();
20         navegador.manage().window().maximize();
21         navegador.get(url);
22
23         espera = new WebDriverWait(navegador, 60);
24     }
25
26     protected static void fecharWeb() {
27
28         navegador.close();
29     }
30 }
31 }
32
```

Nos Hooks temos a SetUp e a tearDown, pois quando for iniciado o projeto, a setUp vai cuidar de disponibilizar a URL do site, realizar todos os passos do cucumber e quando tudo for realizado, o tearDown vai cuidar de fechar o navegador, assim finalizando os testes.

```
hooks.java X
1 package br.com.MarcusVinicius.configuracoes;
2
3 import br.com.MarcusVinicius.enums.Web;
6
7 public class hooks extends BaseTest {
8
9     @Before
10     public void setUp() {
11         System.out.println("iniciando automacao");
12         inicializarAplicacaoWeb(Web.CHROME, "https://automacaocombatista.herokuapp.com/");
13     }
14
15     @After
16     public void tearDown() {
17         System.out.println("terminando automacao");
18
19         fecharWeb();
20     }
21
22 }
23
```

### 3.3 Web

Na Web.java estamos colocando uma condição a ser realizada ao abrir o navegador, nesse caso essa condição é inibir notificações de aparecerem durante a execução do teste.

```
Web.java X
1 package br.com.MarcusVinicius.enums;
2
3 import org.openqa.selenium.WebDriver;
4 import org.openqa.selenium.chrome.ChromeDriver;
5 import org.openqa.selenium.chrome.ChromeOptions;
6
7 import br.com.MarcusVinicius.interfaces.WebApplication;
8
9 public enum Web implements WebApplication {
10     CHROME {
11         public WebDriver getWebDriver() {
12             System.setProperty("webdriver.chrome.driver",
13                 "src/main/resources/drivers/chromedriver.exe");
14             ChromeOptions opcoes = new ChromeOptions();
15             opcoes.addArguments("--disable-notifications");
16
17             return new ChromeDriver(opcoes);
18         }
19     }
20 }
21
22
```

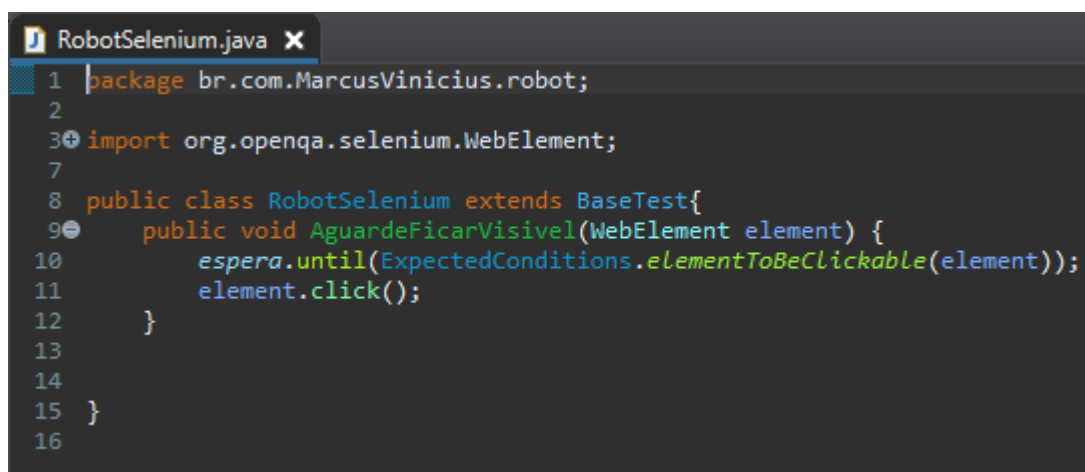
### 3.4 WebApplication

Na WebApplication declaramos um WebDriver constante.

```
WebApplication.java X
1 package br.com.MarcusVinicius.interfaces;
2
3 import org.openqa.selenium.WebDriver;
4
5 public interface WebApplication {
6     WebDriver getWebDriver();
7
8 }
9
```

## 3.5 RobotSelenium

O Robot é o elemento de controle na hora dos testes, pois alguns botões só aparecem na tela depois de certo tempo, e para evitar que a automação efetue erros, esta classe só permite o clique de um certo botão, quando o mesmo aparecer na tela.



```
RobotSelenium.java X
1 package br.com.MarcusVinicius.robot;
2
3 import org.openqa.selenium.WebElement;
4
5
6
7
8 public class RobotSelenium extends BaseTest{
9     public void AguardeFicarVisivel(WebElement element) {
10         espera.until(ExpectedConditions.elementToBeClickable(element));
11         element.click();
12     }
13
14
15 }
16
```

## 3.6 Drivers

A versão do google chrome no período desta automação está na versão 93.0.4577.82, a versão mais recente do ChromeDriver para windows neste teste é a 93.0.5577.63.





Google Chrome



O Google Chrome está atualizado  
Versão 93.0.4577.63 (Versão oficial) 64 bits

## ChromeDriver 93.0.4577.63

Supports Chrome version 93

For more details, please see the [release notes](#).

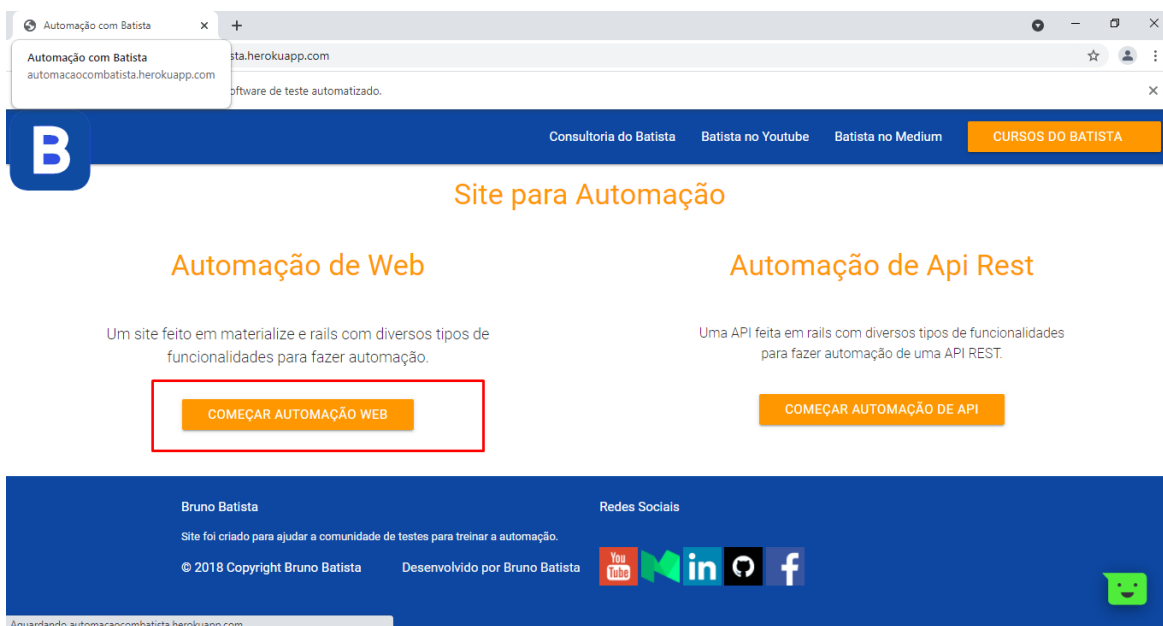
## 3.7 Runner

A Runner está mapeando em sua configuração a pasta resource que é onde se encontra a nossa .feature, então os passos serão executados, e no console nos é devolvido às snippets formatadas no padrão CamelCase, e as mesmas serão implementadas nos arquivos com steps em seus nomes.

```
RunTest.java x
1 package br.com.MarcusVinicius.Runner;
2
3 import org.junit.runner.RunWith;
9
10
11 @RunWith(Cucumber.class)
12 @CucumberOptions(
13     features = "src/test/resources",
14     glue = {"br.com.MarcusVinicius.configuracoes",
15           "br.com.MarcusVinicius.Web"},
16     tags = "@web",
17     plugin = {"pretty", "json:target/cucumber.json", "html:target/cucumber-reports.html" },
18     dryRun = false,
19     snippets = SnippetType.CAMELCASE
20 )
21 public class RunTest extends BaseTest{
22
23 }
24
```

## 3.8 Inicial

Os arquivos de nome Inicial representam a funcionalidade da página principal, onde após o navegador ser aberto, a url ser pesquisada, na pagina que nos é apresentada, devemos clicar no botão “Começar Automação Web”.



Na “InicialPage.java” encontramos o botão “Começar Automação Web” pelo Xpath e então retornando o “WebElement” “getBtnIniciarAutomacao”.

```

1 package br.com.MarcusVinicius.pages;
2
3 import org.openqa.selenium.WebDriver;
4
5 public class InicialPage {
6     public InicialPage(WebDriver navegador) {
7
8         PageFactory.initElements(navegador, this);
9     }
10
11     @FindBy(xpath = "//a[text()='Começar Automação Web']")
12     private WebElement btnIniciarAutomacao;
13
14     public WebElement getBtnIniciarAutomacao() {
15         return btnIniciarAutomacao;
16     }
17 }

```

Na “InicialFuncionalidade.java” é feita a ação na página, nesse caso, ao declararmos um “WebElement” de nome “botao” que recebe o “getBtnIniciarAutomacao” da inicial page, é aguardado o botão ficar visível e então é simulado o “click”.

```

1 package br.com.MarcusVinicius.funcionalidades;
2
3 import org.openqa.selenium.WebElement;
4
5 public class InicialFuncionalidades extends BaseTest{
6     private InicialPage inicialPage;
7
8     public InicialFuncionalidades() {
9         this.inicialPage = new InicialPage(navegador);
10     }
11
12     public void clicaNoBotaoComecarAutomacao() {
13         WebElement botao = this.inicialPage.getBtnIniciarAutomacao();
14         espera.until(ExpectedConditions.visibilityOf(botao));
15         botao.click();
16     }
17 }

```

Na “InicialSteps.java” vai o nosso primeiro snippet que o Runner nos devolveu, e esta classe vai efetuar o “clicaNoBotaoComecarAutomacao”.

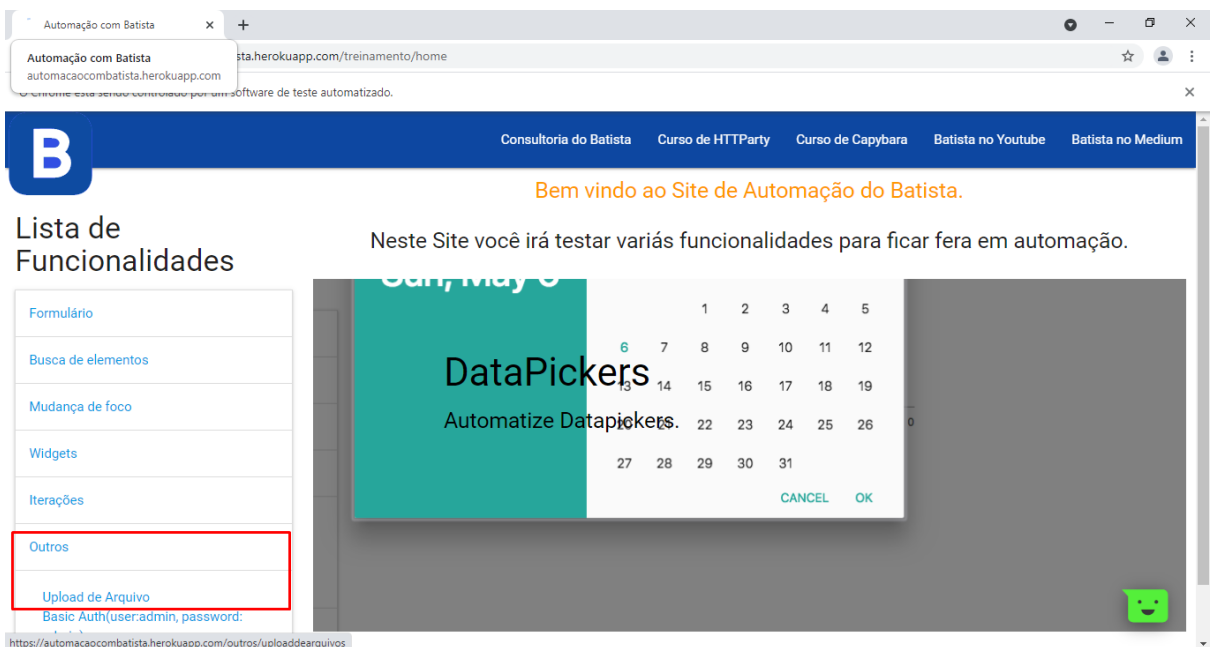
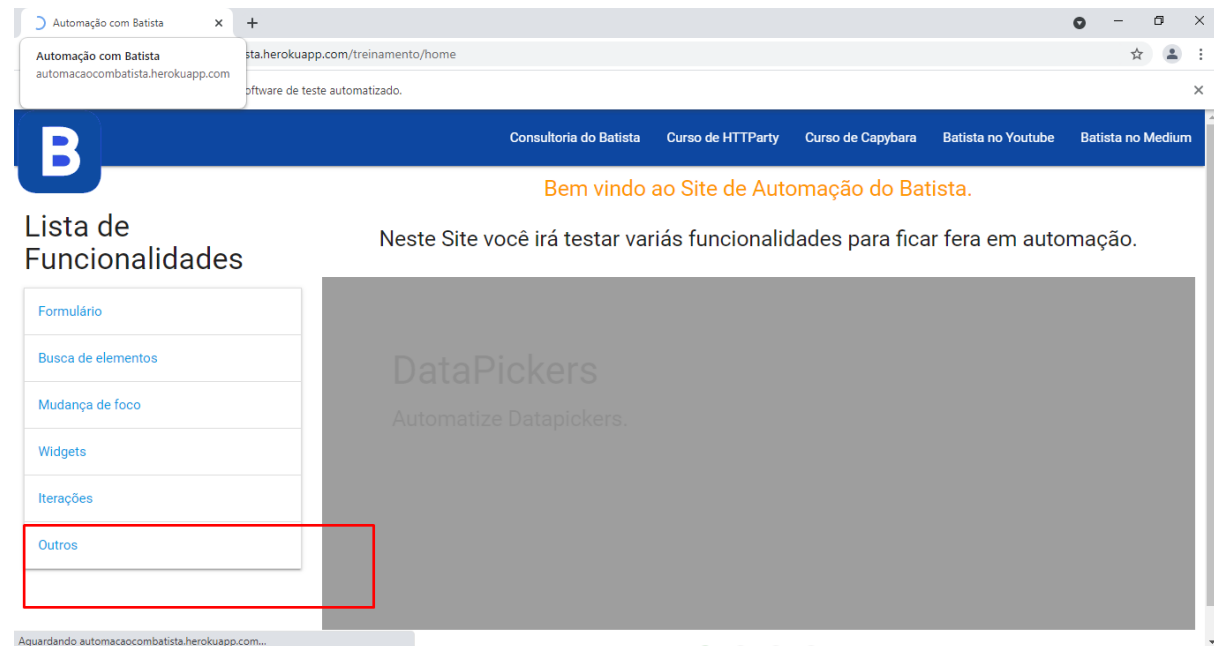
```

InicialPage.java X
1 package br.com.MarcusVinicius.pages;
2
3 import org.openqa.selenium.WebDriver;
4
5
6
7
8 public class InicialPage {
9     public InicialPage(WebDriver navegador) {
10
11         PageFactory.initElements(navegador, this);
12     }
13
14     @FindBy(xpath = "//a[text()='Começar Automação Web']")
15     private WebElement btnIniciarAutomacao;
16
17     public WebElement getBtnIniciarAutomacao() {
18         return btnIniciarAutomacao;
19     }
20
21
22 }
23

```

### 3.9 Treinamento

Passando da Página inicial, entramos na área de treinamento e nos é apresentado uma lista de opções, sabendo que o objetivo deste teste é fazer o Upload de um arquivo que, então devemos clicar em “Outros”, aguardar a sub-opções ficarem visíveis e clicar em “Upload de arquivo”.



Na “TreinamentoPage.java” estamos declarando o caminho do botões “Outros” e “Upload de arquivos” por seus xpaths e retornando seus WebElements “getLinkUpload” e “getLinkUploadDeArquivo”.

```

TreinamentoPage.java X
1 package br.com.MarcusVinicius.pages;
2
3 import org.openqa.selenium.WebDriver;
4
5 public class TreinamentoPage {
6     public TreinamentoPage(WebDriver webDriver) {
7         PageFactory.initElements(webDriver, this);
8     }
9     @FindBy(xpath = "//a[text()='Outros']")
10    private WebElement LinkUpload;
11
12    public WebElement getLinkUpload(){
13        return LinkUpload;
14    }
15    @FindBy(xpath = "//a[text()='Upload de Arquivo']")
16    private WebElement LinkUploadDeArquivo;
17
18    public WebElement getLinkUploadDeArquivo(){
19        return LinkUploadDeArquivo;
20    }
21 }
22
23
24
25
26
27

```

Juntamente com a nossa classe “RobotSelenium.java” a função dessa classe fará com que após clicar em outros, o selenium vai esperar o “getLinkUploadDeArquivo” ficar visível para poder efetuar o “click”.

```

TreinamentoFuncionalidades.java X
1 package br.com.MarcusVinicius.funcionalidades;
2
3
4
5 import br.com.MarcusVinicius.configuracoes.BaseTest;
6
7 public class TreinamentoFuncionalidades extends BaseTest {
8     private TreinamentoPage treinamentoPage;
9     private RobotSelenium robotSelenium;
10
11    public TreinamentoFuncionalidades() {
12        this.treinamentoPage = new TreinamentoPage(navegador);
13        this.robotSelenium = new RobotSelenium();
14    }
15    public void clicaNoLinkOutros() {
16        this.robotSelenium.AguardeFicarVisivel(this.treinamentoPage.getLinkUpload());
17    }
18
19    public void clicaNoLinkUploadDeArquivos() {
20        this.robotSelenium.AguardeFicarVisivel(this.treinamentoPage.getLinkUploadDeArquivo());
21    }
22 }
23
24
25
26
27
28

```

Na “TreinamentoSteps.java”, primeiro será efetuado o “clicaNoLinkOutros” e depois “clicaNoLinkUploadDeArquivos”.

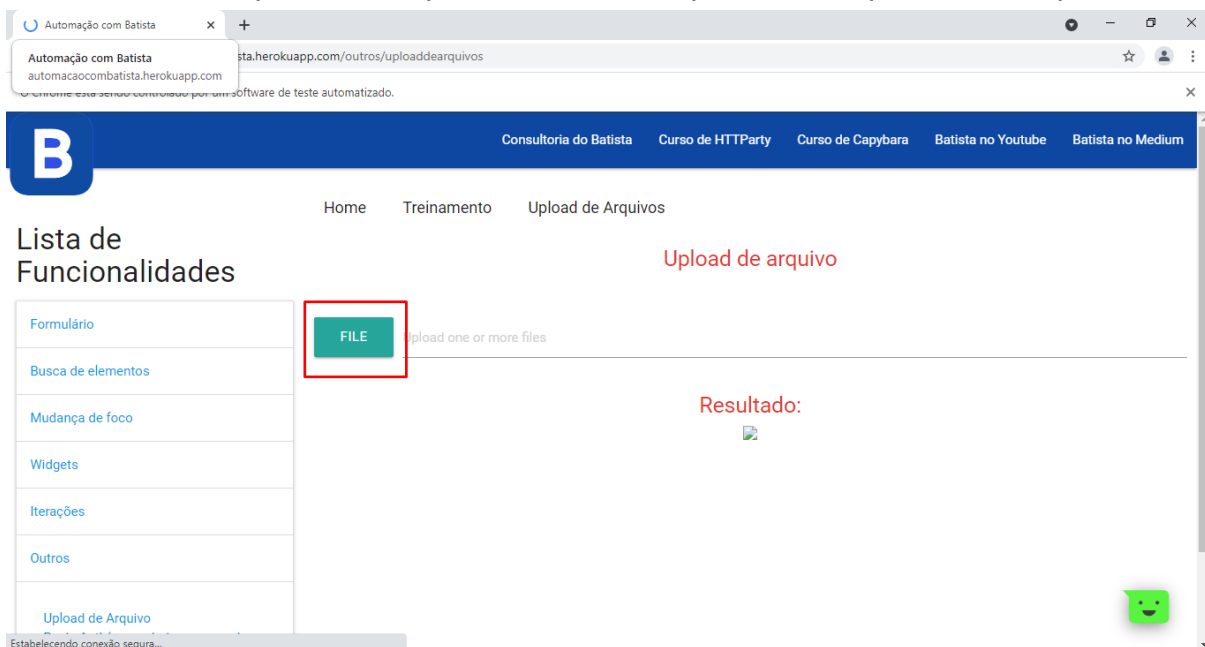
```

TreinamentoSteps.java X
1 package br.com.MarcusVinicius.steps;
2
3 import br.com.MarcusVinicius.funcionalidades.TreinamentoFuncionalidades;
4
5
6 public class TreinamentoSteps {
7     private TreinamentoFuncionalidades treinamentoFuncionalidade;
8
9     public TreinamentoSteps() {
10         this.treinamentoFuncionalidade = new TreinamentoFuncionalidades();
11     }
12
13     @Dado("^navego ate \"([^\"]*)\" e clico em \"([^\"]*)\"$")
14     public void navegoAteEClicoEm(String arg1, String arg2) throws Throwable {
15         this.treinamentoFuncionalidade.clicaNoLinkOutros();
16         this.treinamentoFuncionalidade.clicaNoLinkUploadDeArquivos();
17     }
18 }
19

```

## 3.10 Upload

Agora que conseguimos entrar na página desejada, vamos clicar no botão “FILE” e então realizar o Upload do arquivo “LEIAME.txt” presente na pasta “ParaUpload”.



Em “UploadPage.java” é feita a declaração do xpath de “FILE” e nos é retornado o “WebElement” “getLinkFile”.

```

1 package br.com.MarcusVinicius.pages;
2
3 import org.openqa.selenium.WebDriver;
4
5
6
7
8 public class UploadPage {
9     public UploadPage(WebDriver webDriver) {
10         PageFactory.initElements(webDriver, this);
11     }
12
13     @FindBy(xpath = "//input[@id='upload']")
14     private WebElement LinkFile;
15
16     public WebElement getLinkFile() {
17         return LinkFile;
18     }
19
20
21 }
22

```

Em "UploadFuncionalidades.java" a "getLinkFile" será clicada.

```

1 package br.com.MarcusVinicius.funcionalidades;
2
3 import br.com.MarcusVinicius.configuracoes.BaseTest;
4
5
6
7 public class UploadFuncionalidades extends BaseTest {
8     private UploadPage uploadPage;
9
10
11     public UploadFuncionalidades() {
12         this.uploadPage = new UploadPage(navegador);
13     }
14
15
16     public void clicaEmFile() {
17         this.uploadPage.getLinkFile();
18     }
19
20 }
21

```

A "UploadSteps.java" realiza as funcionalidades de "UploadFuncionalidades.java"



```

UploadSteps.java X
1 package br.com.MarcusVinicius.steps;
2
3 import br.com.MarcusVinicius.funcionalidades.UploadFuncionalidades;
4
5
6
7 public class UploadSteps {
8     private UploadFuncionalidades uploadFuncionalidades;
9
10    public UploadSteps() {
11        this.uploadFuncionalidades = new UploadFuncionalidades();
12    }
13
14    @Dado("^clico em \"([^\"]*)\"$")
15    public void clicoEm(String arg1) throws Throwable {
16
17        this.uploadFuncionalidades.clicaEmFile();
18    }
19
20 }
21

```

## 3.11 RealizaUpload

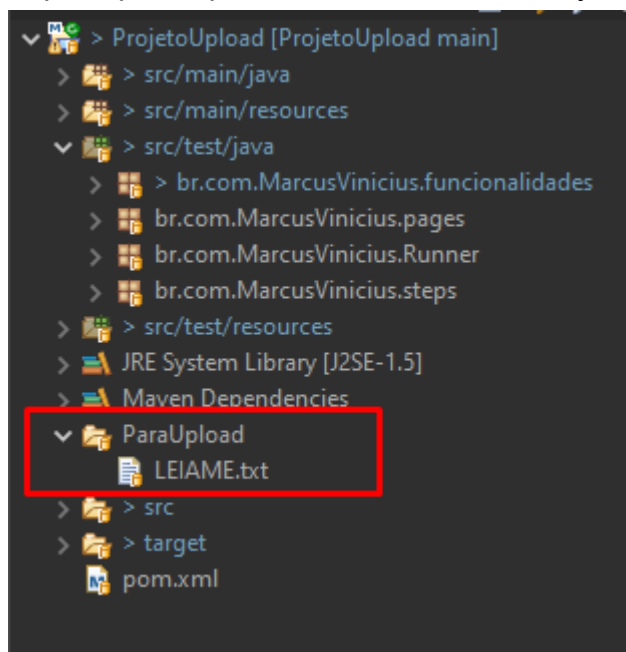
Clicando em “FILE” o próximo e último passo será realizar o Upload do arquivo “LEIAME.txt” que se encontra na pasta “ParaUpload” no site.

> Este Computador > Windows (C:) > Usuários > Marcus Silva > eclipse-workspace > ProjetoUpload > ParaUpload				
	Nome	Data de modificação	Tipo	Tamanho
do abalho	LEIAME	02/08/2021 11:42	Documento de Te...	1 KB

A “RealizaUploadPage.java” está utilizando o mesmo xpath de “FILE” porém sua função não será um clique, mas inserir o caminho em que o arquivo “LEIAME.txt” se encontra.

```
RealizaUploadPage.java X
1 package br.com.MarcusVinicius.pages;
2
3 import org.openqa.selenium.WebDriver;
4
5
6
7
8 public class RealizaUploadPage {
9
10 public RealizaUploadPage(WebDriver webDriver) {
11     PageFactory.initElements(webDriver, this);
12 }
13
14
15 public WebElement getCampoRealizaUpload() {
16     return campoRealizaUpload;
17 }
18
19 @FindBy(xpath = "//input[@id='upload']")
20 private WebElement campoRealizaUpload;
21
22
23
24 }
25
```

Em “RealizaUploadFuncionalidades.java” é enviado a coordenadas de onde está o arquivo para upload, através do “sendKeys”.



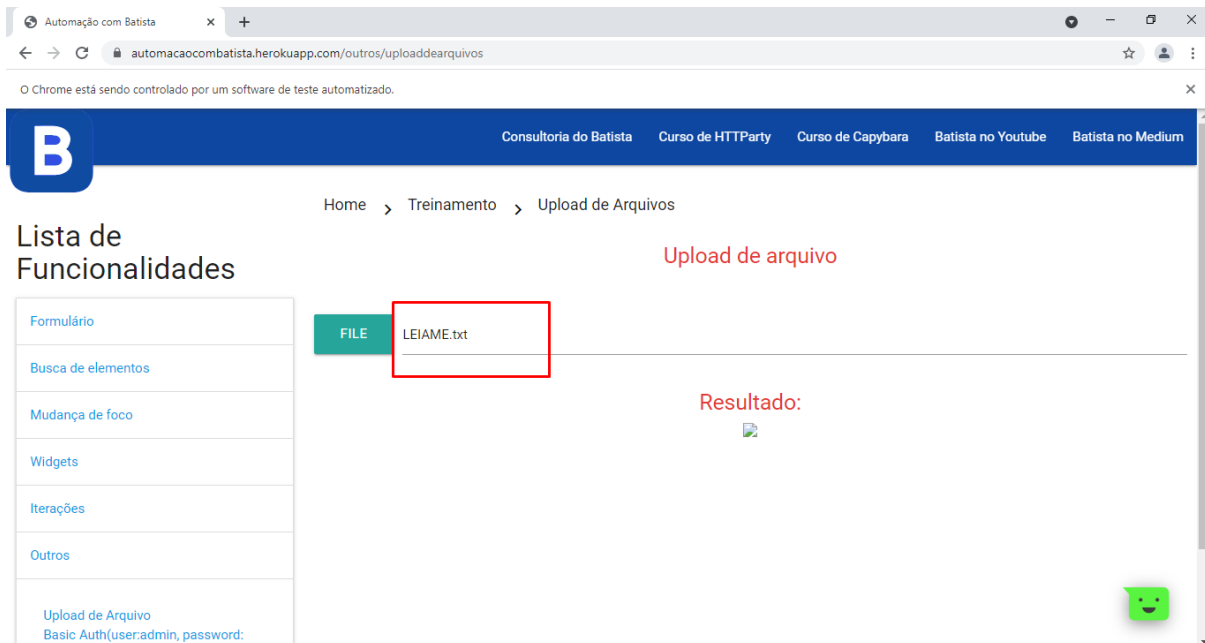
```
RealizaUploadFuncionalidades.java X
1 package br.com.MarcusVinicius.funcionalidades;
2
3
4 import java.io.File;
5
6
7
8
9 public class RealizaUploadFuncionalidades extends BaseTest{
10     private RealizaUploadPage realizaUploadPage;
11
12
13     public RealizaUploadFuncionalidades() {
14         this.realizaUploadPage = new RealizaUploadPage(navegador);
15     }
16
17
18     public void preencherCampoDePesquisa() {
19         File arquivo = new File("ParaUpload/LEIAME.txt");
20         this.realizaUploadPage.getCampoRealizaUpload().sendKeys(arquivo.getAbsolutePath());
21     }
22
23 }
24
25
```

O último passo realizado é o Upload do Arquivo “LEIAME.txt” que se encontra em “ParaUpload”.

```
RealizaUploadSteps.java X
1 package br.com.MarcusVinicius.steps;
2
3 import br.com.MarcusVinicius.funcionalidades.RealizaUploadFuncionalidades;
4
5
6
7 public class RealizaUploadSteps {
8     private RealizaUploadFuncionalidades realizaUploadFuncionalidades;
9
10     public RealizaUploadSteps() {
11         this.realizaUploadFuncionalidades = new RealizaUploadFuncionalidades();
12     }
13
14
15     @Entao("^devo fazer o Upload do arquivo \"([^\"]*)\" que esta na pasta \"([^\"]*)\"")
16     public void devoFazerOUploadDoArquivoQueEstaNaPasta(String arg1, String arg2) throws Throwable {
17         this.realizaUploadFuncionalidades.preencherCampoDePesquisa();
18     }
19
20 }
21
```

## 4. Resultados

Os resultados foram um sucesso e as 4 “steps” passaram.



## 5. Tendências de defeitos

Para estes testes serem feitos é recomendado que o Chromedriver do projeto seja da mesma versão que o seu navegador Chrome para que não ocorra erros na hora de rodar a automação.

## 6. Ações Sugeridas

Estar sempre atento a versão do seu navegador e baixar a versão mais atualizado do site

<https://chromedriver.chromium.org/downloads>.