**COMO CRIAR NOVO PROJETO NO INTELLIJ IDEA**

MAVEN (Gerenciador de Dependencias)

Nome do Projeto: webdriverJava

Location: ~\IdeaProjects\webdriver-java

Artifact Coords

GroupId: br.com.caioandrian.webdriverJava

ArtifactId: webdriveJava

Version:1.0-SNAPSHOT

POM.XML (arquivo de dependencias)

adicionar as dependencias JUNIT e SELENIUM JAVA

ENABLE AUTO-IMPORT

DENTRO DE SRC

MAIN = para quem está criando uma aplicação

TEST = para quem vai criar um script de teste

DENTRO DA PASTA JAVA (poderia ser outra linguagem)

CRIAR UM PACOTE (package) tests

CRIAR CLASSE InformacoesUsuarioTest

INSERIR ANOTACAO @Teste (do pacote JUNIT) - isso deixará claro que sua classe é

para testes.

*\* TODO MÉTODO DE TESTE PRECISA SER PUBLICO!*

*\* TODO MÉTODO DE TESTE NÃO PODE RETORNAR UM VALOR, OU SEJA SEMPRE SERÁ VOID!*

DENTRO DA CLASSE DEVEMOS DIZER QUANDO O TESTE PASSARÁ OU IRÁ FALHAR.

PARA ISSO USAMOS A CLASSE ASSERT (DO JUNIT)

Assert.assertEquals(expect: 1, actual: 1);

*\* PARA USAR TODOS OS MÉTODOS DA CLASSE ASSERT, PODEMOS ALTERAR O IMPORT PARA SER ESTÁTICO,* ***IMPORT STATIC ORG.JUNIT.ASSERT.\*****;*

***\* STATIC*** *SERVE PARA REFERENCIAR TODOS AQUELES ATRIBUTOS/MÉTODOS DE CLASSE, OU SEJA, QUE PODEM SER ACESSADOS DIRETAMENTE DA DEFINIÇÃO DA CLASSE, SEM PRECISAR INSTANCIAR NENHUM OBJETO.*

*\* DESSA FORMA NÃO PRECISAMOS MAIS COLOCAR ASSERT ANTES DE TODO MÉTODO*

assertEquals(expect: 1, actual: 1);

OUTRO MÉTODO MUITO USADO PARA VALIDAÇÃO

assertTrue(boolean condicaoAtual, String mensagemErro)

**ADICIONANDO O BROWSER / WEBDRIVER NO PROJETO**

System.setProperty(chave do parametro, onde esta?)

System.setProperty("webdriver.chrome.driver", "C:\\drivers.chromedriver.exe");

**CRIANDO UMA VARIÁVEL PARA RECEBER UMA INSTÂNCIA DO CHROME DRIVER.**

WebDriver navegador = new ChromeDriver();

**NAVEGANDO PARA A PAGINA DE TASKIT**

navegador.get("http://www.juliodelima.com.br/taskit");

**O PADRÃO AAA DOS SCRIPTS DE TESTE**

**ARRANGE - PREPARAÇÃO DO TESTE**

// Clicar no link com texto "Sign in" poderia ser pelo ID, NAME, CLASSE, ETC.

// Clicar no campo com name "login" que está dentro do formulário de id "signinbox"

// Digitar no campo login o texto "julio0001"

// Clicar no campo com name "password" que está dentro do formulário de id "signinbox"

// Digitar no campo password o texto "123456"

// Clicar no link com texto "SIGN IN"

// O texto a ser verificado sempre será o que o usuário vê na tela

// Validar que dentro do elemento com a class "me" está o texto "Hi, Julio"

// Fechar o navegador

**ACTION - CODIFICANDO O QUE A AÇÃO QUE SERÁ FEITA NO TESTE**

// Clicar no link com texto "Sign in" poderia ser pelo ID, NAME, CLASSE, ETC.

navegador.findElement(By.linkText("Sign in")).click();

// Identificando o formulário de id "signinbox"

WebElement formularioSignInbox = navegador.findElement(By.id("signinbox"));

// Digitar no campo com name "login" que está dentro do formulário signinbox o texto "julio0001"

formularioSignInbox.findElement(By.name("login")).sendKeys("julio0001");

// Digitar no campo com name "password" que está dentro do formulário signinbox o texto "123456"

formularioSignInbox.findElement(By.name("password")).sendKeys("123456");

// Clicar no link com texto "SIGN IN"

// O texto a ser verificado sempre será o que o usuário vê na tela, no HTML está Sign in

formularioSignInbox.findElement(By.name("password")).click();

formularioSignInbox.findElement(By.linkText("SIGN IN")).click();

// Validar que dentro do elemento com a class "me" está o texto "Hi, Julio"

WebElement me = navegador.findElement(By.className("me"));

String textoNoElementoMe = me.getText();

**ASSERT**

// Validação

assertEquals("Hi, Julio", textoNoElementoMe);

// Fechar o navegador

navegador.quit();

*\* A VALIDAÇÃO DEVERÁ SER SEMPRE NO FINAL DE CADA TESTE E AS BOAS PRÁTICAS RECOMENDAM QUE SEJA APENAS UMA VALIDAÇÃO POR TESTE.*

**COMO EXECUTAR O MESMO MÉTODO SEMPRE QUE INICIAR UM NOVO TESTE?**

private WebDriver navegador;

**@BEFORE (ANTES DO TESTE INICIAR)**

public void setUp(){

// Abrindo um navegador pela variável instanciada do chrome driver

System.setProperty("webdriver.chrome.driver", "C:\\drivers\\chromedriver79.exe");

navegador = new ChromeDriver();

// Adicionando tempo de espera para tentar achar o elemento na página

navegador.manage().timeouts().implicitlyWait(60, TimeUnit.SECONDS);

// Abrir a instância em tela cheia.

navegador.manage().window().maximize();

// Navegando para a página de taskit

navegador.get("http://www.juliodelima.com.br/taskit");

}

**@AFTER (DEPOIS QUE FINALIZAR O TESTE)**

public void tearDown() {

// Fechar o navegador

navegador.quit();

}

**COMO SELECIONAR A OPÇÃO DE UM SELECT NA PÁGINA?**

// Identificar o popup onde está o formulário com id "addmoredata"

WebElement popUpAddmoredata = navegador.findElement(By.id("addmoredata"));

// Na combo de name "type" a opção "Phone"

WebElement campoType = popUpAddmoredata.findElement(By.name("type"));

new Select(campoType).selectByVisibleText("Phone");

**TRABALHANDO COM XPATH UM POUCO MAIS COMPLEXO**

// Clicar pelo xpath no elemento //span[text()="+5149212121"]/following-sibling::a

// following = o primeiro após encontrar a classe span com o número desejado

// preceding = o primeiro antes de encontrar a classe span com o número desejado

navegador.findElement(By.xpath("//span[text()=\"+5149212121\"]/following-sibling::a")).click();

**TRABALHANDO COM JAVASCRIPT NO TESTE**

// Confirmar a janela javascript

navegador.switchTo().alert().accept();

**COMO FAZER PARA QUE UM MÉTODO DE TESTE NÃO SEJA MAIS EXECUTADO DURANTE O TESTE?**

//@Test

**A CLASSE EXPECTEDCONDITIONS**

Ela nos ajudam a criar formas de *aguardar que certos eventos aconteçam*, como por exemplos que um determinado seja removido do DOM (caso que vimos no vídeo relacionado ao staleness). Abaixo está um link com todas as estratégias de espera, ou seja, métodos da classe ExpectedConditions:

As estratégias mais utilizadas são:

alertIsPresent()

Aguarda até que uma mensagem Javascript seja apresentada.

elementToBeClickable(By localizador)

Aguarda até que um elemento esteja pronto para receber um clique, por exemplo, ele poderia ficar desabilitado durante algum período e então teríamos de esperar até que ficasse habilitado novamente.

visibilityOfElementLocated(By localizador)

Aguarda até que as propriedades CSS display e hidden possuam valores que possibilitam a visualização do elemento pelo usuário.

**TIRANDO SCREENSHOTS NO TESTE**

Criar um **pacote chamado suporte**, onde teremos duas classes.

**Classe Generator**

public class Generator {

public static String dataHoraParaArquivo(){

Timestamp ts = new Timestamp(System.currentTimeMillis());

return new SimpleDateFormat("yyyyMMddHHmmss").format(ts);

//exemplo: 20200124194200

}

}

*\* QUANDO TRABALHAMOS COM TIMESTAMP, USA-SE A BIBLIOTECA DO JAVA.SQL.TIMESTAMP;*

**Classe Screenshot**

public static void tirar(WebDriver navegador, String arquivo){

File screenshot = ((TakesScreenshot) navegador).getScreenshotAs(OutputType.FILE);

try {

FileUtils.copyFile(screenshot, new File(arquivo));

} catch (Exception e){

System.out.println("Houveram problemas ao copiar o arquivo para a pasta " + e.getMessage());

}

}

\* *QUANDO TRABALHAMOS COM ARQUIVOS USA-SE A BIBLIOTECA DO JAVA.IO.FILE;*

**Classe do Teste**

// Rule faz parte da biblioteca do JUnit, irá pegar o nome do método que está sendo executado.

**@Rule**

public TestName test = new TestName();

// Tirando screenshot do teste

String screenshotArquivo = "C:\\Users\\caio\\IdeaProjects\\test-report\\taskit\\"

+ Generator.dataHoraParaArquivo() + test.getMethodName() + ".png";

Screenshot.tirar(navegador, screenshotArquivo);

**TRABALHANDO COM TESTES DIRIGIDOS A DADOS (DATA DRIVEN TESTING)**

Data-Driven Testing é uma estratégia de testes que visa impedir a duplicação de scripts quando

apenas os dados de entrada ou saída são diferentes.

// Adicionar a dependência do Easy Test no arquivo pom.xml

<!-- https://mvnrepository.com/artifact/org.easetech/easytest-core -->

<dependency>

<groupId>org.easetech</groupId>

<artifactId>easytest-core</artifactId>

<version>1.4.0</version>

</dependency>

De volta para a Classe do Teste

// Adicionar uma anotação com a classe do Easy Test que será usada para rodar o Teste Dirigido a Dados.

**@RunWith**(DataDrivenTestRunner.class)

// Arquivo que será usado para receber os dados externos que serão usados via Easy Test

// Informar no data louder o arquivo criado na pasta resources do projeto

**@DataLoader**(filePaths = "informacoesUsuarioTestData.csv")

Criar a pasta resources dentro da pasta test, caso a ferramenta já não tenha criado.

**Criar o arquivo informacoesUsuarioTestData.csv** (o nome da classe + Data + .csv)

No arquivo .csv, adicionar os dados que serão inseridos no sistema, podendo ser mais de um para o mesmo teste.

Esquema:

qual o método que vai receber os dados, nome da coluna, nome da coluna, nome da coluna

Exemplo:

testAdicionarUmaInformacaoAdicionalDoUsuario,tipo,contato,mensagem

,Phone,+1155112232,Your contact has been added!

,Phone,+1155112233,Your contact has been added!

,Phone,+1155112234,Your contact has been added!

,E-mail,meuteste@gmail.com,Your contact has been added!

,E-mail,meuteste2@gmail.com,Your contact has been added!

No método da classe de teste

// Incluir os parâmetros que serão usados pelo método.

public void testAdicionarUmaInformacaoAdicionalDoUsuario(

**@Param(name = "tipo")String tipo**,

@Param(name = "contato")String contato,

@Param(name = "mensagem")String mensagemEsperada)

{ ... }

Anotação **@Param**, diz que o primeiro parâmetro com nome tipo refere-se a coluna tipo, o segundo ao contato, e o terceiro a mensagem.

\* OS PARÂMETROS E COLUNAS, DEVEM ESTAR NA MESMA ORDEM E EXATAMENTE COM O MESMO NOME.

Substituir os valores que estavam fixos pelos parâmetros dentro do método.

**ORGANIZANDO A CRIAÇÃO DO WEBDRIVER (REFACTOR)**

Separamos a criação do WebDriver

Dentro do pacote chamado suporte, criamos a **Classe Web.**

public class Web {

public static WebDriver createChrome(){

// Abrindo um navegador pela variável instanciada do chrome driver

System.setProperty("webdriver.chrome.driver", "C:\\drivers\\chromedriver79.exe");

WebDriver navegador = new ChromeDriver();

// Adicionando tempo de espera para tentar achar o elemento na página

navegador.manage().timeouts().implicitlyWait(60, TimeUnit.SECONDS);

// Abrir a instância em tela cheia.

navegador.manage().window().maximize();

// Navegando para a página de taskit

navegador.get("http://www.juliodelima.com.br/taskit");

return navegador;

}

}

Iremos criar uma instância do chrome na **Classe de teste**

navegador = Web.createChrome();

**TRABALHANDO COM PAGE OBJECTS**

\* TODA PÁGINA (PAGE OBJECT) PRECISA TER UM ATRIBUTO PRIVADO WEBDRIVER

\* E UM CONSTRUTOR QUE ESTÁ AGUARDANDO UM WEBDRIVER

\* DESSA FORMA É POSSÍVEL NAVEGAR ENTRE AS PÁGINAS

Criamos um **pacote chamado pages,** dentro dela vamos criar as classes de todas as páginas que serão usadas nos testes.

**Classe LoginPage**

private WebDriver navegador;

// Construtor

public LoginPage(WebDriver navegador){

this.navegador = navegador;

}

\* TODO MÉTODO DA PÁGINA OBJECT PRECISA RETORNAR PARA ONDE ELA VAI APÓS SER FINALIZADA SUA AÇÃO

// Após clicar no Sign In iremos para a próxima página "LoginFormPage", mesmo que seja um pop up.

public LoginFormPage clickSignIn(){

navegador.findElement(By.linkText("Sign in")).click();

return new LoginFormPage(navegador);

}

\* UM POUCO SOBRE **ABORDAGEM ESTRUTURAL E ABORDAGEM FUNCIONAL**

Estrutural = é criado um método para cada ação ou campo que serão executados na mesma página.

Funcional = um método recebe todas as ações que serão realizadas nessa página.

Podemos ainda, trabalhar usando um pouco de cada abordagem como nesse modelo abaixo.

**Classe LoginFormPage**

private WebDriver navegador;

public LoginFormPage(WebDriver navegador){

this.navegador = navegador;

}

// Como a próxima ação irá para a mesma página, retornamos a mesma classe.

public LoginFormPage typeLogin(String login){

navegador.findElement(By.id("signinbox")).findElement(By.name("login")).sendKeys(login);

return this;

}

public LoginFormPage digitarPassword(String password){

navegador.findElement(By.id("signinbox")).findElement(By.name("password")).sendKeys(password);

return this;

}

// Ao finalizar essa ação vamos para a página Secreta.

public SecretaPage clicarSignIn(){

navegador.findElement(By.id("signinbox")).findElement(By.name("password")).click();

navegador.findElement(By.linkText("SIGN IN")).click();

return new SecretaPage(navegador);

}

// ABORDAGEM ESTRUTURAL + ABORDAGEM FUNCIONAL

public SecretaPage fazerLogin(String login, String password){

digitarLogin(login);

digitarPassword(password);

clicarSignIn();

return new SecretaPage(navegador);

}

**Classe SecretaPage**

public class SecretaPage extends BasePage{

public SecretaPage(WebDriver navegador) {

super(navegador);

}

public MePage clicarMe(){

navegador.findElement(By.className("me")).click();

return new MePage(navegador);

}

}

**Classe MePage**

public class MePage extends BasePage{

public MePage(WebDriver navegador) {

//como o atributo da BasePage é protect, a Classe Filha pode usar esse atributo da classe Pai.

super(navegador);

}

public MePage clicarNaAbaMoreDataAboutYou(){

navegador.findElement(By.linkText("MORE DATA ABOUT YOU")).click();

return this;

}

// o popup que será aberto para cadastrar um novo contato é considerado uma nova página

// logo iremos para AddContactPage

public AddContactPage clicarNoBotaoAddMoreDataAbout(){

navegador.findElement(By.xpath("//button[@data-target=\"addmoredata\"]")).click();

return new AddContactPage(navegador);

}

}

**Classe AddContactPage**

public class AddContactPage extends BasePage{

public AddContactPage(WebDriver navegador) {

super(navegador);

}

public AddContactPage escolherTipoDeContato(String tipo){

// Selecionar opção do Select Box

WebElement campoType = navegador.findElement(By.id("addmoredata")).findElement(By.name("type"));

new Select(campoType).selectByVisibleText(tipo);

return this;

}

public AddContactPage digitarContato(String contato){

navegador.findElement(By.id("addmoredata")).findElement(By.name("contact")).sendKeys(contato);

return this;

}

public MePage clicarSalvar(){

// Identificar e clicar no link que tem o texto "SAVE" que está na div "addmoredata"

navegador.findElement(By.linkText("SAVE")).click();

return new MePage(navegador);

}

// Método funcional com a abordagem estrutural

public MePage adicionarContato(String tipo, String contato){

escolherTipoDeContato(tipo);

digitarContato(contato);

clicarSalvar();

return new MePage(navegador);

}

}

Criamos uma nova classe de teste chamada, dentro do **pacote tests**

**Classe InformacoesUsuarioPageObjectsTest**

// Classe do Easy Test que será usada para rodar o *Teste Dirigido a Dados*

**@RunWith**(DataDrivenTestRunner.class)

// Arquivo que será usado para receber os dados externos que serão usados via Easy Test

// Informar no data louder o arquivo criado na pasta resources do projeto

**@DataLoader**(filePaths = "InformacoesUsuarioPageObjectsTest.csv")

\* CRIAR ARQUIVO CSV DENTRO DE RESOURCES COM: nome\_da\_classe\_teste,login,senha,tipo,contato,mensagem

private WebDriver navegador;

@Before

public void setUp(){

navegador = Web.createChrome();

}

\* ADICIONANDO O PAGE OBJECTS COM TESTES DIRIGIDOS A DADOS

@Test

public void testAdicionarUmaInformacaoAdicionalDoUsuario(

@Param(name = "login")String login,

@Param(name = "senha")String senha,

@Param(name = "tipo")String tipo,

@Param(name = "contato")String contato,

@Param(name = "mensagem")String mensagemEsperada){

String textoToast = new LoginPage(navegador) // Classe LoginPage

.fazerLogin(login,senha) // Classe LoginFormPage

.clicarMe() // Classe SecretaPage

.clicarNaAbaMoreDataAboutYou() // Classe MePage

.clicarNoBotaoAddMoreDataAbout() // Classe MePage

.adicionarContato(tipo, contato) // Classe AddContactPage

.capturarTextoToast(); // Classe BasePage

// As boas práticas recomendam que apenas no final de um teste tenha uma validação

assertEquals(mensagemEsperada, textoToast);

}

@After

public void tearDown(){

navegador.quit();

}

**OTIMIZANDO A ESTRUTURA DE PAGE OBJECTS**

Como toda página precisa ter um construtor que recebe um Webdriver para continuar navegando. Criamos uma nova classe base para ser estendida nas demais classes do pacote pages.

\* *PROTECTED PERMITE QUE UMA CLASSE FILHA QUANDO ESTENDIDA DA BASEPAGE, POSSA USAR SEU ATRIBUTO DIRETAMENTE.*

**Classe BasePage**

public class BasePage {

protected WebDriver navegador;

public BasePage(WebDriver navegador){

this.navegador = navegador;

}

}

\* *COMO CAPTURAR A MENSAGEM DO ID TOAST É ALGO BASTANTE* ***COMUM EM TODOS OS TESTES****, COLOCAMOS O MÉTODO DE CAPTURAR A MENSAGEM NA CLASSE BASE.*

// Notificacao que aparece no sistema após adicionar, excluir, editar, etc.

public String capturarTextoToast(){

// Na mensagem de id "toast-container" validar que o texto é "Your contact has been added!"

return navegador.findElement(By.id("toast-container")).getText();

}

Exemplo como ficará em outras classes:

**Classe LoginPage**

public class LoginPage **extends BasePage**{

public LoginPage(WebDriver navegador) {

// Refere-se ao atributo da classe Pai (BasePage). Ou seja, o navegador da classe Base está recebendo o navegador criado pelo Web.createChrome() dentro da classe de Teste.

super(navegador);

}

}

**Classe LoginFormPage**

public class LoginFormPage **extends BasePage**{

public LoginFormPage(WebDriver navegador) {

super(navegador);

}

….

}

**TRABALHANDO COM TESTES NA NUVEM ATRAVÉS DO BROWSER STACK**

Cadastrar-se no site com uma conta free-trial.

[https://www.browserstack.com/](https://www.browserstack.com/automate/java)

Escolher quais configurações (device, sistema operacional, versão do navegador, resolução, etc) serão usadas no teste

<https://www.browserstack.com/automate/java>

**Na Classe BasePage**

// Variáveis de uso para acessar o STACK BROWSER

// https://www.browserstack.com/automate/java

public static final String USERNAME = "xxxxxxxx";

public static final String AUTOMATE\_KEY = "xxxxxxxxxxxxxxxx";

public static final String URL = "https://" + USERNAME + ":" + AUTOMATE\_KEY + "@hub-cloud.browserstack.com/wd/hub";

// Carregar o driver que está na NUVEM

// https://www.browserstack.com/automate/java

public static WebDriver createBrowserStack(){

DesiredCapabilities caps = new DesiredCapabilities();

caps.setCapability("browser", "IE");

caps.setCapability("browser\_version", "11.0");

caps.setCapability("os", "Windows");

caps.setCapability("os\_version", "7");

caps.setCapability("resolution", "1024x768");

caps.setCapability("name", "Bstack-[Java] Sample Test");

caps.setCapability("browserstack.debug", "true");

//caps.setCapability("resolution", "1024x768");

//caps.setCapability("browserstack.video", "false");

WebDriver navegador = null;

// Substituir o primeiro URL pela URL do java.net

try {

navegador = new RemoteWebDriver(new URL(URL), caps);

// Adicionando tempo de espera para tentar achar o elemento na página

navegador.manage().timeouts().implicitlyWait(60, TimeUnit.SECONDS);

// Navegando para a página de taskit

navegador.get("http://www.juliodelima.com.br/taskit");

} catch (MalformedURLException e) {

System.out.println("Houveram problemas com a URL:" + e.getMessage());

}

return navegador;

}

**Na Classe InformacoesUsuarioPageObjectsTest**

// Usado para acessar um driver local em sua máquina

//navegador = Web.createChrome();

// Usado para acessar um driver na Nuvel pelo Browser Stack

navegador = Web.createBrowserStack();

Para verificar os resultados obtidos no Browser Stack, acessar: <https://automate.browserstack.com/dashboard/v2/>

\* É POSSÍVEL VISUALIZAR OS VÍDEOS DE CADA TESTE EXECUTADO.

Concorrente da Browser Stack - <https://saucelabs.com/>

**TRABALHANDO COM LINHAS DE COMANDO AO INVÉS DE RODAR OS TESTES ATRAVÉS DA IDE.**

Bastante útil quando você quer colocar seus testes em um processo de integração contínua.

No site <https://maven.apache.org/download.cgi> precisamos baixar o binário do maven, [apache-maven-3.6.3-bin.zip](http://mirror.nbtelecom.com.br/apache/maven/maven-3/3.6.3/binaries/apache-maven-3.6.3-bin.zip)

Vamos executar o arquivo mvn.exe que está dentro da pasta apache-maven-3.63.3/bin

Para isso precisamos antes, editar as variáveis do sistema operacional.

No Windows:

Em Sistemas/Avançado/Variáveis Ambiente/Path -> editar o valor, colocando no final do texto:

***;local onde extraiu a pasta\apache-maven-3.6.3\bin***

No Mac:

Verificar a Aula do professor Julio, seção 5-30.

Navegar pelo command prompt até a pasta que está seu projeto.

Exemplo: C:\Users\caio\IdeaProjects\webdriver-java

Executar o comando ***mvn clean test -Dtest=tests.informacoesUsuarioPageObjectsTest***

tests = qual o pacote que possui o teste

E qual o nome da classe do teste

Caso esteja acontecendo erro de não encontrar a JDK ao executar a linha de comando.

Adicionar no pom.xml do projeto

<build>

<plugins>

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.1</version>

<configuration>

<fork>true</fork>

<executable>C:\Program Files\Java\jdk1.8.0\_241\bin\javac.exe</executable>

</configuration>

</plugin>

</plugins>

</build>

Configurar variáveis ambiente

<http://www.matera.com/blog/post/tutorial-instalacao-do-java-jdk-e-eclipse-no-windows>

Variável **JAVA\_HOME**

Valor: C:\Program Files\Java\jdk1.8.0\_45

Variável **Path**

Editar valor: ;%JAVA\_HOME%\bin

Variável **ClassPath**

Valor %JAVA\_HOME%\lib\tools.jar

\* SE O TESTE ESTIVER RODANDO NA NUVEM (BROWSER STACK), VOCÊ PODERÁ VISUALIZAR O VÍDEO E RESULTADO DO TESTE PELO SITE.