Tópicos especiais I:
testes e automação para dispositivos móveis

Pós graduação em Testes Ágeis
22 e 23 de julho de 2022

Prof. Maria Clara Bezerra

www.github.com/clarabez

O que vimos na última aula

- Definimos o que são testes para mobile;
- Tipos de redes;
- Dispositivos reais x emulados;
- Contexto sobre mobile;
- Recursos e sensores;
- Tipos de testes
- Fragmentação do SO;
- Gestos na tela;
- Atividade #1

Ferramentas que usamos na última aula

- Setup de ambiente;
- Comandos ADB;
- Monkey testing;
- Android Studio;
- Dispositivo emulado (AVD e/ou Genymotion);
- Espresso;
- UIAutomator.

O que vamos ver nesta aula

Vamos focar no **Appium** e realizar alguns **exercícios** em classe.

Dispositivo emulado (AVD do Android Studio e/ou Genymotion);
Appium Inspector
Appium Desktop (Server)
IDE para desenvolvimento JAVA ou Python (eu vou usar VSCode com Python)

Appium desktop (server): https://github.com/appium/appium-desktop/releases
Appium inspector: https://github.com/appium/appium-inspector/releases/

O que vamos ver nesta aula

Vamos focar no **Appium** e realizar alguns **exercícios** em classe.

Atividade #2
Instalar a aplicação
Conectar com o Appium server
Definição de cenário
Mapeamento de elementos
Realização dos fluxos iniciais

Atividade #3
Instalar a aplicação
Utilizar métodos do Appium
Waits

Appium

- Open source;
- Multiplataforma;
- Tem como base o webdriver;
- *Framework* mais popular para mobile;
- Atende dispositivos emulados e reais;
- Todos os tipos de aplicações;
- Também usado para *desktop*.

Appium

- Open source;
- Multiplataforma;
- Tem como base o webdriver;
- *Framework* mais popular para mobile;
- Atende dispositivos emulados e reais;
- Todos os tipos de aplicações;
- Também usado para *desktop*.

- Open source;
- *Multibrowsers*;
- Tem como base o *webdriver*;
- *Framework* mais popular para web;

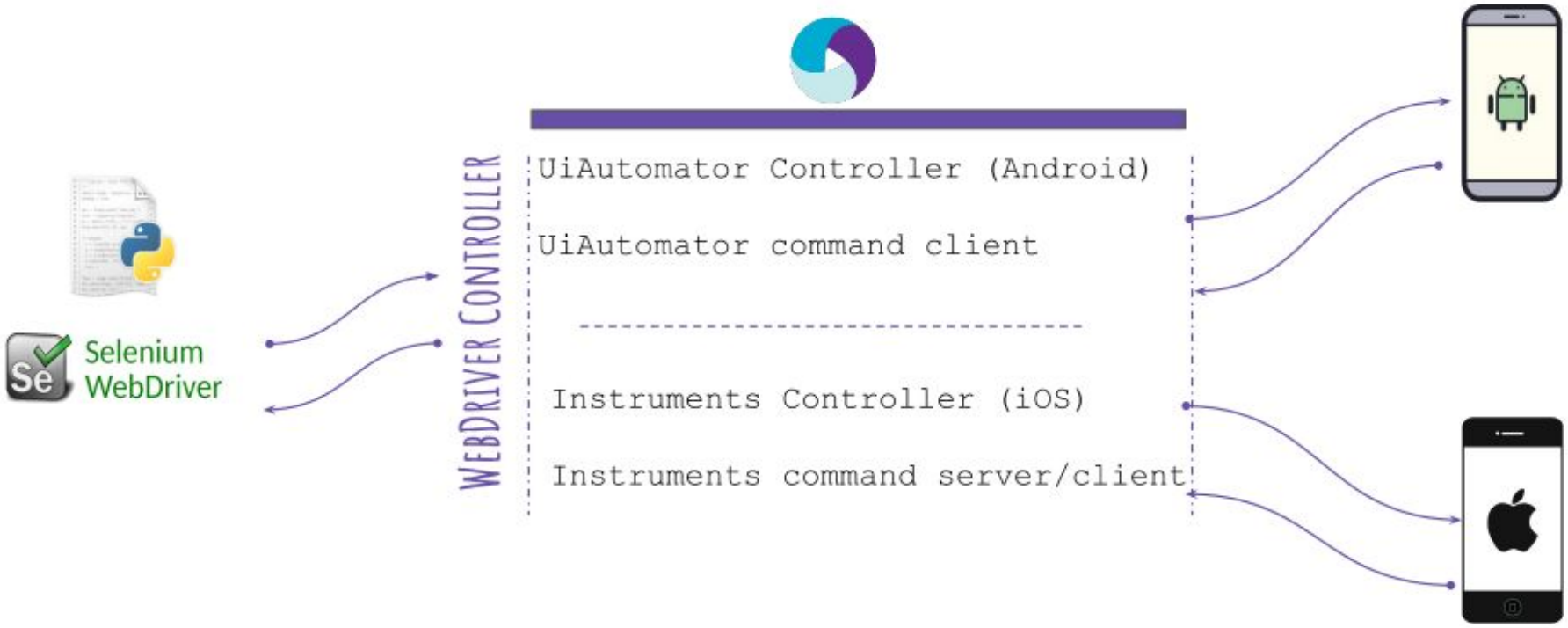
Appium

- Open source;
- Multiplataforma;
- Tem como base o webdriver;
- *Framework* mais popular para mobile;
- Atende dispositivos emulados e reais;
- Todos os tipos de aplicações;
- Também usado para *desktop*.

- Open source;
- *Multibrowsers*;
- Tem como base o *webdriver*;
- *Framework* mais popular para web;

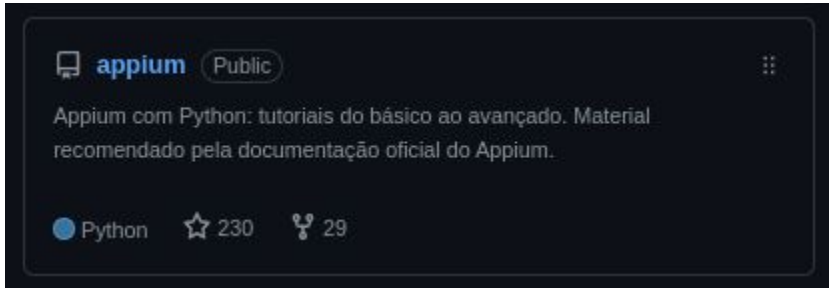
Appium

C.e.S.A.R school



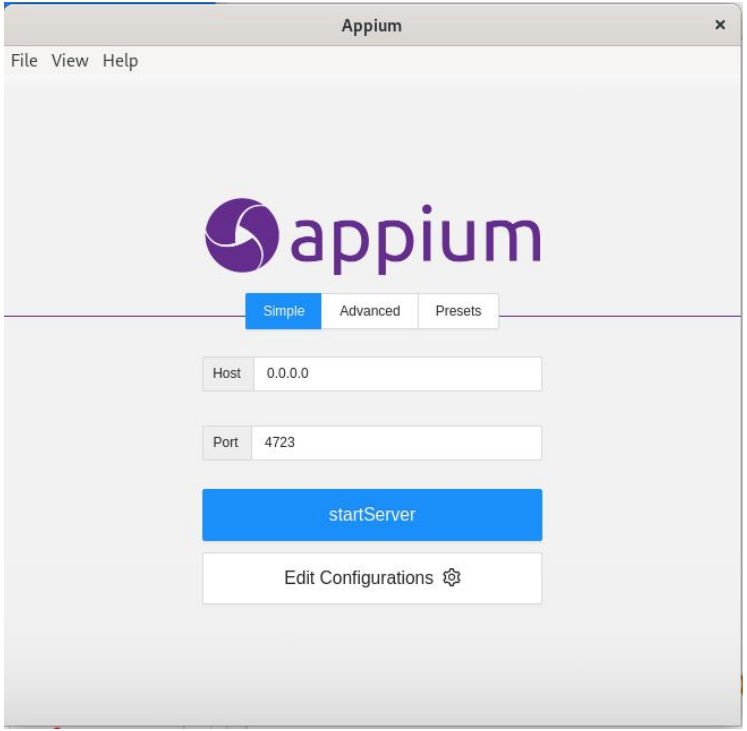
Appium

Vamos utilizar o Curso de **Appium** como referência:



⭐ https://github.com/clarabez/appium ⭐

Appium

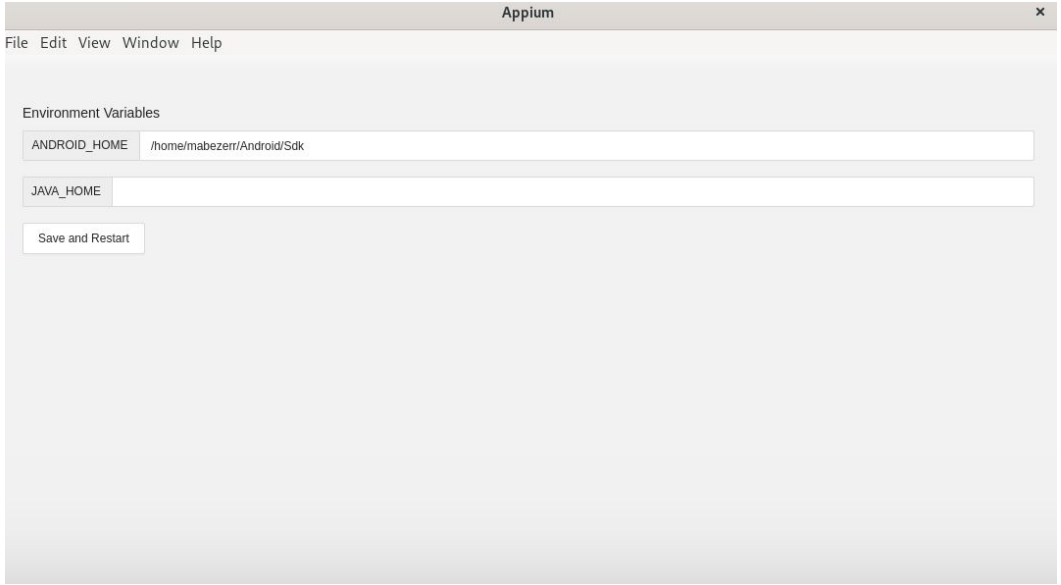


Appium desktop (server):
Releases · appium/appium-desktop · GitHub

Appium inspector:
https://github.com/appium/appium-inspector/releases/

Vamos praticar!

Appium - configuração

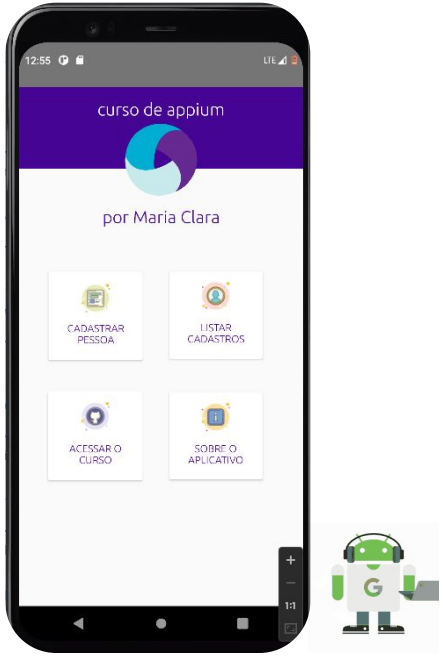


A mesma variável de ambiente que definimos na aula anterior, temos que adicionar aqui no campo **ANDROID_HOME**, cujo valor é: **/<localização>/<para>/Android/Sdk**

Essa é uma forma rápida e prática de configurarmos o ambiente.

Vamos praticar!

Dia de *release* do APK do curso de Appium!



- Aplicativo desenvolvido para o curso de Appium;
- Link do repositório: https://github.com/clarabez/appium-curso-apk

- Tem algumas *issues*, vcs podem registrar aqui: https://github.com/clarabez/appium-curso-apk/issues

- O time de desenvolvimento (eu) agradece 😅

Entregáveis da aula deste final de semana

Aplicativo do curso de Appium:
- Instalação do aplicativo;
- Conexão com o Appium;
- Mapeamento de elementos;
- Fluxo inicial;
- Criação de novos casos.

Aplicativo Resultados:
- Instalação do aplicativo;
- Conexão com o Appium;
- Mapeamento de elementos;
- Fluxo inicial;
- Criação de novos casos.

Prazo final de envio da atividade: 6 de agosto.
Como enviar a atividade: email (clarinhab@gmail.com) com link do repositório no GitHub

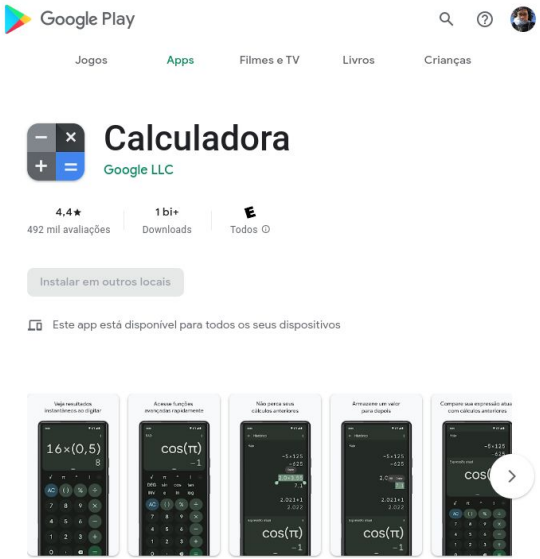
C.e.S.A.R school

QUIZ TIME

Baixando um aplicativo (.apk) da Play Store

Acessar a **“Google Play”**:
https://play.google.com/store/games?hl=pt_BR&gl=US

Vamos procurar o aplicativo **“Calculadora”** do Google:
https://play.google.com/store/apps/details?id=com.google.android.calculator&hl=pt_BR&gl=US

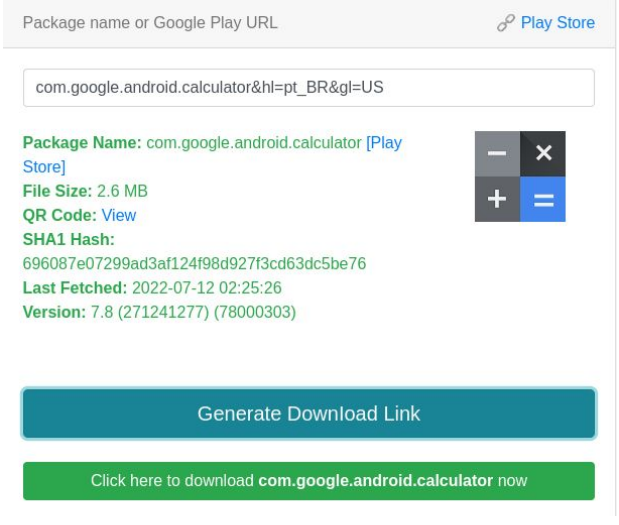


Baixando um aplicativo (.apk) da Play Store

Vamos usar a ferramenta **"Evozi"** para fazer o download do aplicativo:
https://apps.evozi.com/apk-downloader/

Vamos copiar o "id" da aplicação "Calculadora" (fica na URL) e colar na barra de busca do Evozi:

Baixe o aplicativo em: "Click here to download ... now"



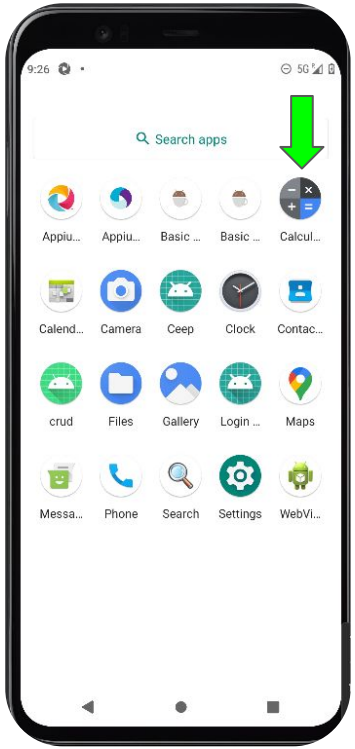
Instalando um aplicativo no dispositivo

Podemos instalar por 2 caminhos:

- Clicando e arrastando o aplicativo no dispositivo (vou mostrar como);
- Utilizando comandos adb:

```
adb install -r nome.do.aplicativo.apk
```

```
→ apks adb install -r com.google.android.calculator.apk
Performing Streamed Install
Success
→ apks
```



Quais testes devem ser automatizados?

Testes de Regressão: repetitivos, exaustivos, tendem a crescer em número;
Funcionalidades complexas: evita o erro humano;
Smoke: Análise rápida das funcionalidades principais, recorrentes;
Performance: Quase impossível de fazer manualmente
Funcionais: tendem a ser exaustivos
Sem dependência de interação humana: porém podem ter semi-automatizados

Conectando o Appium Server e nosso DUT

C.E.S.A.R school

1. Instanciar um dispositivo emulado;
2. Startar o nosso **Appium server**;
3. Usar a biblioteca [Appium-Python-Client]();
4. Definir as mínimas *capabilities* para conectar ao dispositivo:

```
from appium import webdriver

desired_cap = {
    'platformName': 'Android',
    "deviceName": "emulator-5554",
    'autoGrantPermissions': True,
    'newCommandTimeout': 300

}

driver = webdriver.Remote('http://localhost:4723/wd/hub', desired_cap)
```

5. Iniciar o código de forma interativa:

```
> python -i NomeDoArquivo.py
```

Conectando o Appium Server e nosso DUT

```
public class CursoAppium {

    private AndroidDriver driver;
    public WebDriverWait wait;

    By btnCadastrarPessoa   = By.id("button cadastrar");
    By btnCadastrar         = By.id("BotaoCadastrar");


    @BeforeMethod
    public void setUp() throws MalformedURLException {
        DesiredCapabilities desiredCapabilities = new DesiredCapabilities();
        desiredCapabilities.setCapability("appium:platformName", "Android");
        desiredCapabilities.setCapability("appium:deviceName", "emulator-5554");
        desiredCapabilities.setCapability("appium:app",
"/home/mabezerr/Documents/pessoal/aulas/appium-java/src/test/java/resources/app-curso.apk");
        desiredCapabilities.setCapability("appium:appPackage", "com.example.cursoappium");
        desiredCapabilities.setCapability("appium:newCommandTimeout", 3600);

        URL remoteUrl = new URL("http://localhost:4723/wd/hub");

        driver = new AndroidDriver(remoteUrl, desiredCapabilities);
        wait = new WebDriverWait(driver, Duration.ofSeconds(10));
    }

    @Test
    public void sampleTest() {
        wait.until(ExpectedConditions.visibilityOfElementLocated(btnCadastrarPessoa)).click();
        driver.findElement(btnCadastrar).click();
        driver.findElement(radioButtonMulher).click();


    }

    @AfterMethod
    public void tearDown() {
        driver.quit();
    }
}
```

Biblioteca Appium-client JAVA - https://github.com/appium/java-client

Appium com JAVA - https://github.com/clarabez/appium-curso-java

Utilizando métodos nativos do Appium

Realizando chamadas:

Método: make_gsm_call.
Parâmetros: Número (str), action (str)
Ações: call, accept, cancel, hold.

```
> driver.make_gsm_call('101010', 'call')
> driver.make_gsm_call('101010', 'accept')
> driver.make_gsm_call('101010', 'cancel')
> driver.make_gsm_call('101010', 'hold')
```

Recebendo SMS:

Método: send_sms
Parâmetros: Número (str), Conteúdo (str)

```
> driver.send_sms('101010', 'hello, world!')
```

Utilizando métodos nativos do Appium

Controlando nível de bateria:

Método: set_power_capacity.
set_power_ac

Ações: int, “on”, “off”.

```
> driver.set_power_ac()
> driver.set_power_capacity()

> driver.AC_ON
> driver.AC_OFF

> driver.battery_info
```

Utilizando métodos nativos do Appium

Orientação de tela:
Método: orientation.
Ações: portrait, landscape.

```
> driver.orientation('value')
```

Utilizando métodos nativos do Appium

Screenshot e gravação de vídeo da tela:
Método: set_power_capacity.
set_power_ac
Ações: int, “on”, “off”.

```
> driver.start_recording_screen('caminho/para/folder')
> driver.stop_recording_screen('caminho/para/folder')

> driver.save_screenshot('caminho/para/folder')
```

Utilizando métodos nativos do Appium

Gerenciando aplicações:
Método: set_power_capacity.
set_power_ac
Ações: int, “on”, “off”.

```
> driver.close app('PacoteDoApp')
> driver.install_app('/caminho/do/apk')
> driver.is app installed('PacoteDoApp')
> driver.launch app('PacoteDoApp')
> driver.remove app('PacoteDoApp')
> driver.terminate_app('PacoteDoApp')
```

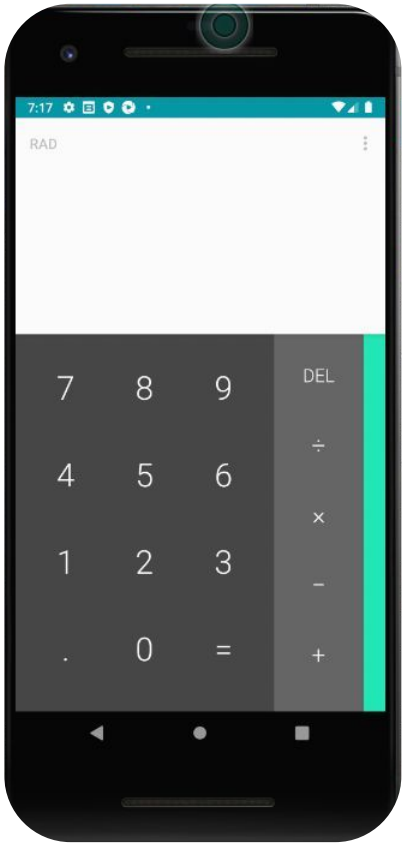
Utilizando métodos nativos do Appium

Coletando informações gerais

Método: set_power_capacity.
set_power_ac
Ações: int, “on”, “off”.

```
> driver.current_activity
> driver.current_context
> driver.current_package
> driver.current_url # esse precisa de capability para web
> driver.battery_info
> driver.get_settings() # testar esse aqui
```

Exercício calculadora



> Aplicativo da Calculadora nativa do Android.

Vamos mapear alguns elementos da Calculadora e realizar uma operação de soma?

Para isso, vamos:

- Identificar os valores para o desired capabilites;
  - platformName
  - deviceName
  - appPackage (comando ADB)
  - appActivity (comando ADB)
- Fazer a chamada da aplicação via Appium;
- Mapear os elementos necessários para a operação;
- Realizar a soma de 2 elementos.
- Exibir o resultado da soma na IDE.

Waits em automação

Implícito: sleep 😴

```
import time
time.sleep(10)
```

```
from selenium import webdriver

driver = webdriver.Firefox()
driver.implicity_wait(10)
driver.get("www.url.com.br")
driver.find_element_by_id("valor_do_id")
```

Class Expected Conditions - https://selenium-python.readthedocs.io/waits.html

Waits em automação

C.e.S.A.R school

Implícito: sleep 😴

```
import time
time.sleep(10)
```

```
from selenium import webdriver

driver = webdriver.Firefox()
driver.implicity_wait(10)
driver.get("www.url.com.br")
driver.find_element_by_id("valor_do_id")
```

Explícito: esperar por algum evento 🧐

```
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.support.wait import WebDriverWait
from appium.webdriver.common.mobileby import MobileBy

self.num1 = WebDriverWait(self.driver.instance, 10).until(EC.presence_of_element_located((
            MobileBy.ID, 'com.android.calculator2:id/digit_1'
        )))
```

(valores em JAVA no próximo slide)

Class Expected Conditions - https://selenium-python.readthedocs.io/waits.html

Waits em automação

Implícito: sleep 😴

```
Thread.sleep(5000);
```

Explícito: esperar por algum evento 🧐

```
Webdriver wait = new WebDriverWait(Driver, 10);
wait.until(ExpectedConditions.visibilityOfElementLocated(By.id(
'com.android.calculator2:id/digit_1')));
```

Class Expected Conditions:
https://www.selenium.dev/selenium/docs/api/java/org/openqa/selenium/support/ui/ExpectedConditions.html

Exercício Curso de Appium

> Aplicativo do Curso de Appium.

Vamos tentar realizar o mesmo teste que fizemos com o Espresso?

Para isso, vamos:

- Identificar os valores para o desired capabilites;
  - platformName
  - deviceName
  - appPackage (comando ADB)
  - appActivity (comando ADB)

- Fazer a chamada da aplicação via Appium;
- Realizar um cadastro válido (ou qualquer outro cenário de sua escolha).

Para isso precisar utilizar de ***Waits***. Vamos usar a abordagem explícita?

Exercício Curso de Appium


```
from appium import webdriver
from selenium.webdriver.support.wait import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.common.by import By
from appium.webdriver.common.appiumby import AppiumBy

desired_cap = {
    'platformName': 'Android',
    [...]
}
driver = webdriver.Remote('http://localhost:4723/wd/hub', desired_cap)

btnCadastrarPessoa = "button_cadastrar"
textInputNome = "TextInputNome"
btnCadastrar = "BotaoCadastrar"

cadastrar = WebDriverWait(driver, 10).until(EC.presence_of_element_located((
By.ID, btnCadastrarPessoa)))
cadastrar.click()
WebDriverWait(driver, 10).until(EC.presence_of_element_located((By.ID, textInputNome)))
driver.find_element(by=AppiumBy.ID, value=textInputNome).send_keys("Maria4")
driver.find_element(by=AppiumBy.ID, value=btnCadastrar).click()
```

Abstraindo o Appium com Robot

C.E.S.A.R school

Robot Framework - https://robotframework.org/
Appium Library - https://github.com/serhatbolsu/robotframework-appiumlibrary

Abstraindo o Appium com Robot

Robot Framework é baseado em *keywords* (palavras-chave) em alto nível;
É orientado à teste de aceitação (ATDD);
É estendido por bibliotecas Python e JAVA, abstraindo a codificação;
É utilizado para *mobile, web, desktop*.

AppiumLibrary é a biblioteca de testes do ***Appium*** para ***Robot***. É compatível tanto para Android quanto para iOS.

Robot Framework - https://robotframework.org/
Appium Library - https://github.com/serhatbolsu/robotframework-appiumlibrary

Abstraindo o Appium com Robot

C.E.S.A.R school

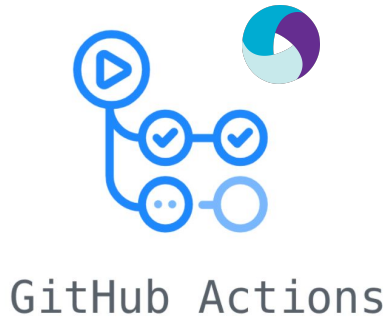
Robot Framework é baseado em *keywords* (palavras-chave) em alto nível;
É orientado à teste de aceitação (ATDD);
É estendido por bibliotecas Python e JAVA, abstraindo a codificação;
É utilizado para *mobile, web, desktop*.

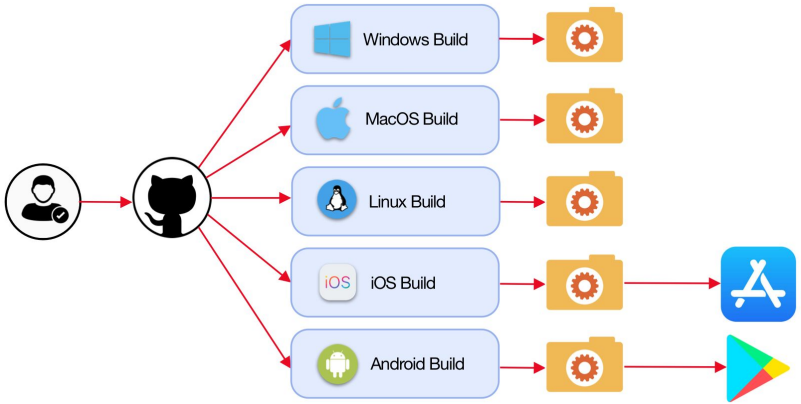
AppiumLibrary é a biblioteca de testes do ***Appium*** para ***Robot***. É compatível tanto para Android quanto para iOS.

Vamos ver como funciona?
https://github.com/clarabez/appium-robot

Robot Framework - https://robotframework.org/
Appium Library - https://github.com/serhatbolsu/robotframework-appiumlibrary

Pipelines com Appium - Github Actions





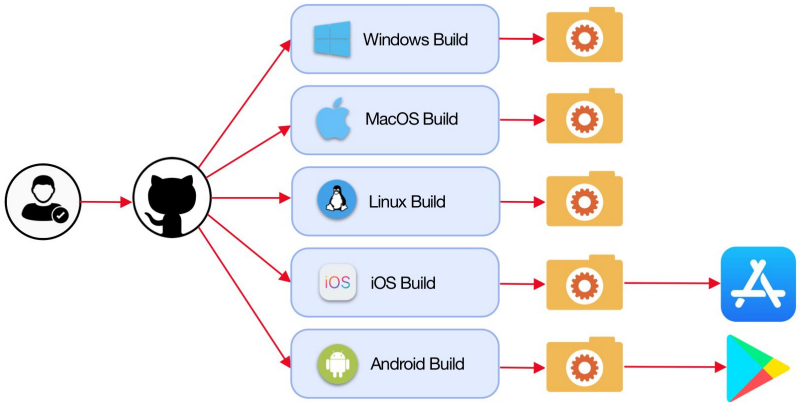
GitHub Actions - https://docs.github.com/pt/actions/learn-github-actions/understanding-github-actions
Github Actions Marketplace - https://github.com/marketplace?type=actions
Github Actions Pricing - https://github.com/pricing

Pipelines com Appium - Github Actions

C.e.S.A.R school

GitHub Actions é uma plataforma de CI/CD que nos permite testar nossa *pipeline* de implantação.

Podemos programar execuções a depender de eventos em nossos repositórios.



GitHub Actions - https://docs.github.com/pt/actions/learn-github-actions/understanding-github-actions
Github Actions Marketplace - https://github.com/marketplace?type=actions
Github Actions Pricing - https://github.com/pricing

BrowserStack com Appium

BrowserStack

O *BrowserStack* é um serviço de provimento de dispositivos reais na nuvem, com diversos de modelos disponíveis.

Tem opção de conta grátis por 15 dias, com 100 minutos de automação mobile.

É possível conectar o *Appium Inspector* ou *GitHub Actions* a uma *farm* do *BrowserStack*.

BrowserStack - https://www.browserstack.com/
BrowserStack Capabilities - https://www.browserstack.com/app-automate/capabilities?tag=w3c

Projeto em sala

Escolher um aplicativo:
Pode ser os que vimos em sala ou qualquer um da Google Play. Você pode também dar um fork num projeto existente em meu GitHub (qualquer coisa, te ajudo com isso!).

python™ Java

Escolher a linguagem:
Pode ser qualquer uma, inclusive usando o *recorder* do ***Appium***.

GitHub Actions

Pipeline:
Vamos hospedar o projeto no Github e criar uma *pipeline* através do **GitHub Actions**.

JUnit pytest unittest

Plus:
Que tal adicionarmos um *framework* de testes ao projeto?

Reconhecimento de image com o Appium

C.e.S.A.R school

Exercício Resultados

> Aplicativos Resultados - TSE

- Procurar pela aplicação na PlayStore
- Baixar a aplicação usando o Evozi
- Instalar a aplicação no dispositivo
- Iniciar uma sessão com o Appium
  - Desired capabilities (platformName, deviceName, app)
- Mapear botão da tela inicial
- Passar o código para uma linguagem de programação preferida
- Realizar um fluxo simples de maneira corrida
- Podemos fazer uso de algum framework de testes (unittest ou pytest, junit, etc.)

Para fecharmos a aula:

- Automação mobile pode começar de forma "simples" com atividades de rotina;
- Para ambiente de desenvolvimento Android: ***Espresso*** e ***Ulautomator***;
- ***Appium server*** estabelece conexão com o dispositivo, diferente do ***Ulautomator***;
- ***Appium*** vai muito além de mapear elementos;
- Associado a uma linguagem de programação, o ***Appium*** ganha muitos poderes;
- A documentação do ***Appium*** traz muitas dicas para irmos além.

Tópicos especiais I:
testes e automação para dispositivos móveis

Pós graduação em Testes Ágeis
22 e 23 de julho de 2022

Prof. Maria Clara Bezerra

www.github.com/clarabez

IntelliJ - instalar SDK