ADS/GTI

SPRINT 3 – MISSÃO 11

PROJETO: "DEPLOYMENT QUALITY ASSURANCE"

ESTUDO DE CASO

Uma certa empresa decidiu estabelecer uma cultura QUALITY ASSURANCE em seu modelo de negócio, visando impactar positivamente processos de qualidade em suas as suas áreas de operação e tecnologia.

ESCOPO DO PROJETO

O projeto será composto por 3 Sprints que se complementam, onde os alunos deverão construir ações que validem a empresa a possuir uma cultura orientada a Q.A.

Em **duplas/trios** os alunos desenvolverão projeto 3 em Sprints:

- SPRINT 1: Vale 0,5 ponto na AC-1 e presenças nas aulas
- SPRINT 2: Vale 1 ponto na AC-2 e presenças nas aulas
- **SPRINT 3: Vale 1 ponto na AC-3 e presenças nas aulas**

OBJETIVO

Aprender as nuances e aplicabilidade do *QUALITY ASSURANCE* em uma organização. Construir um projeto de implementação de Gerenciamento de Qualidade Total e realizar atividades que valem nota.

SPRINT 3 (1 ponto)

Início: **23/10** – Término: **13/11**. Vale 1,0 ponto na AC-3 e presenças nas aulas.
Composto por 4 missões que se complementam para a entrega total do projeto:

- Missão 9: Testes de Segurança – 25% da AC-3
- Missão 10: Testes de Usabilidade – 25% da AC-3
- **Missão 11: QA em Mobile – 25% da AC-3**
- Missão 12: Integração Contínua (DevSecOps) e entrega final – 25% da AC-3

MISSÃO 11

VALE + 25% DA NOTA AC-3

BÔNUS: essa atividade tem bônus de + 0,25 para todos os alunos que conseguirem realizar a atividade completa sem ajuda do professor

Objetivo: Q.A. - Testes em Mobile

CONTEÚDO TEÓRICO:

Introdução aos Testes em Dispositivos Móveis

Testes em Aplicativos Móveis:

São processos de verificação e validação que asseguram que o aplicativo móvel funciona corretamente, ofereça boa usabilidade, e pronto para ser usado em dispositivos variados.

Abrangem desde funcionalidades básicas, como login e navegação, até aspectos avançados, como desempenho, segurança e compatibilidade entre diferentes sistemas operacionais.

Expectativas do Usuário:

- Usuários de dispositivos móveis esperam que o aplicativo seja rápido, intuitivo e responsivo. Problemas de desempenho, erros e travamentos podem levar a avaliações negativas e à perda de usuários.

Vantagens dos Testes em Aplicativos Móveis

- **Redução de Custos a Longo Prazo**:
  - Reduz significativamente os custos com correções de problemas detectados após o lançamento. Quanto mais cedo um erro é identificado e corrigido, menor o custo para corrigi-lo.

- **Melhoria na Experiência do Usuário (UX)**:
  - Testes de usabilidade garantem que o aplicativo seja fácil de usar e navegação intuitiva. Quando bem executados, esses testes aumentam a satisfação do usuário e as avaliações positivas, retendo e atraindo mais usuários.

- **Prevenção de Erros e Reputação da Marca**:
  - Aplicativos que falham constantemente ou apresentam problemas de segurança afetam a credibilidade e a imagem da marca.

- **Compatibilidade com Dispositivos e Plataformas**:
  - Com diversos dispositivos e versões de sistemas operacionais no mercado, os testes garantem que o aplicativo funcione bem em todos os cenários desejados, assegurando que a base de usuários do app seja o mais abrangente possível.

- **Desempenho em Diferentes Condições**:
  - Testes de performance verificam como o aplicativo reage em condições reais, como conexões de rede lentas, baixa bateria e multitarefa, garantindo que ele ofereça uma boa experiência mesmo em situações adversas.

Tipos de Testes em Aplicativos Móveis

- **Testes de Funcionalidade**:
  - Verifica se o aplicativo funciona como esperado. Exemplos incluem testes de login, navegação entre telas, interações com o usuário e uso de APIs.

- **Testes de Usabilidade**:
  - Avaliam a facilidade de uso e a experiência geral do usuário. Aqui, técnicas como análise heurística (ex.: heurísticas de Nielsen) são úteis para validar a interface.

- **Testes de Performance**:
  - Focados na resposta e na velocidade do aplicativo sob diferentes cargas de trabalho e condições de rede. É essencial para garantir que o aplicativo não trave ou apresente lentidão em momentos críticos.

- **Testes de Segurança**:
  - Avaliam a proteção de dados e identificam vulnerabilidades de segurança, como SQL injection, falhas de autenticação, etc.

- **Testes de Compatibilidade**:
  - Confere a operação do aplicativo em diferentes dispositivos, tamanhos de tela e versões de sistemas operacionais, assegurando uma experiência uniforme.

TAREFA 1 – PREPARAÇÃO:

1. Baixe o arquivo esse "**Missão11-Projeto QA - ADS-5.pdf**" disponível no AVA;

2. Abra o GitHub oficial da dupla/trio e o repositório que estão usando para o projeto;

3. Suba no seu repositório o arquivo "**Missão11-Projeto QA - ADS-5.pdf**";

4. Agora abra o projeto deste repositório e visualize o quadro Kanban que está gerenciando o projeto;

5. Criar e colocar o cartão MISSÃO 11 para a lista EM ANDAMENTO;

TAREFA 2

Para essa atividade prática, vamos configurar o ambiente para testes automatizados de um aplicativo móvel no Windows, utilizando **Appium** com **Android Studio**.

Pré-requisitos e Instalação

1.Verificar se está instalado o NODE..JS

Abra linha de comando no VSCODE e confirme as versões de instalação
node -v
npm -v

- Caso de sucesso no item acima, seguir para o item 2. Caso não, providenciar a instalação no NODE.JS no site oficial (buscar no Google)

- Cheque onde o NODE e o NPM estão instalados:

where node
where npm

2. Instale o Appium
- Após o Node.js, instale o Appium via npm:

npm install -g appium

- Após a instalação, confirme o comando para verificar a versão:

appium –v
where appium

3. Instalação do Android Studio e Configuração do Emulador Android
- Verifique se o CPU da máquina está com a virtualização ativada: Gerenciador de Tarefas – CPU - virtualização habilitada.
- Instalar o Android Studio
- Configure um emulador Android Studio:
    - Abra o Android Studio e crie um novo dispositivo virtual, selecionando um modelo de dispositivo e uma imagem de sistema Android.

4. Configuração das Variáveis de Ambiente
- **ANDROID_HOME**: Caminho para o SDK do Android.
- Normalmente, o SDK está localizado em uma pasta padrão como:
    - Windows: `C:\Users\SeuUsuario\AppData\Local\Android\Sdk`
    - macOS: `/Users/SeuUsuario/Library/Android/sdk`
    - Linux: `/home/SeuUsuario/Android/Sdk`

- **Path**: Adicione o caminho das seguintes pastas do SDK à variável `Path` para que o sistema possa acessar essas ferramentas:
- `<Caminho_do_SDK>/platform-tools`
- `<Caminho_do_SDK>/emulator`
- `<Caminho_do_SDK>/tools`

Como Configurar

No **Windows**:

1. Abra o **Painel de Controle** > **Sistema e Segurança** > **Sistema** > **Configurações avançadas do sistema**.
2. Na aba **Avançado**, clique em **Variáveis de Ambiente**.
3. Adicione uma nova variável chamada `ANDROID_HOME` com o caminho para o SDK.
4. Edite a variável `Path` e adicione os caminhos para `platform-tools`, `emulator` e `tools`.

5. Instalação de Bibliotecas Python

- No **comando do VSCODE** instale as bibliotecas **Appium-Python-Client** e **Unittest**:

pip install Appium-Python-Client
pip install unittest

6. Configuração do Appium Server

- No terminal, execute o servidor Appium com o comando:

Appium

- O servidor estará disponível em http://localhost:4723/wd/hub.

7. Criação do Script de Teste em Python

Para esse exercício, vamos criar um teste de automação que abre um aplicativo de exemplo, navega até a tela de login e realiza uma tentativa de login.

7.1 Estrutura do Projeto

- No **VS Code**, crie uma pasta chamada mobile_test_project.
- Dentro dela, crie o arquivo test_login.py.

7.2. Código do Teste

CRIAR APK.TESTE

Passo 1: Criar o Projeto no Android Studio

1. Abra o Android Studio e selecione **New Project**.
2. Escolha **Empty Activity** e clique em **Next**.
3. Nomeie o projeto como `AppTeste` e selecione **Java** como a linguagem.

4. Clique em **Finish** para criar o projeto.

Passo 2: Código do Aplicativo

Arquivo `MainActivity.java`

Este arquivo contém a lógica do aplicativo, incluindo o botão de cadastro e o campo de entrada.

```
java
Copiar código
package com.example.appteste;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
import android.view.View;

public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        EditText editTextUsername =
findViewById(R.id.editTextUsername);
        Button buttonSubmit = findViewById(R.id.buttonSubmit);

        buttonSubmit.setOnClickListener(new View.OnClickListener()
{
            @Override
            public void onClick(View v) {
                String username =
editTextUsername.getText().toString();
                if (!username.isEmpty()) {
                    Toast.makeText(MainActivity.this, "Cadastro de
" + username + " realizado!", Toast.LENGTH_SHORT).show();
                } else {
                    Toast.makeText(MainActivity.this, "Por favor,
insira um nome.", Toast.LENGTH_SHORT).show();
                }
            }
        });
    }
}
```

Layout `activity_main.xml`

O layout básico para a tela, contendo um campo de entrada e um botão para cadastro.

```
xml
Copiar código
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp">

    <EditText
        android:id="@+id/editTextUsername"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Digite seu nome"
        android:inputType="textPersonName" />

    <Button
        android:id="@+id/buttonSubmit"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/editTextUsername"
        android:layout_marginTop="16dp"
        android:text="Cadastrar" />
</RelativeLayout>
```

Passo 3: Compilar o APK

1. No Android Studio, vá até **Build** > **Build Bundle(s) / APK(s)** > **Build APK(s)**.
2. Após o build, o Android Studio exibirá o caminho onde o APK foi gerado, geralmente em `app/build/outputs/apk/debug/`.

Aqui está o passo a passo para configurar e rodar o teste:

Passo 1: Preparação do Ambiente

1. Certifique-se de que o Appium está instalado e em execução.
2. Abra o **Visual Studio Code** e crie uma pasta para organizar os arquivos de teste.
3. Nessa pasta, crie um arquivo chamado `test_login.py` para escrever o teste.

Passo 2: Escrevendo o Código de Teste

No arquivo `test_login.py`, escreva o código do teste. Ele usará a biblioteca `Appium-Python-Client` para interagir com o aplicativo. Aqui está um exemplo para o teste de login.

```
from appium import webdriver
from appium.webdriver.common.appiumby import AppiumBy
import time

# Configurações do Appium para o emulador Android
desired_caps = {
    "platformName": "Android",
    "deviceName": "emulator-5554",  # Nome padrão do emulador
Android
    "app": "<Caminho_do_APK>",  # Substitua com o caminho completo
do APK gerado
    "automationName": "UiAutomator2"
}

# Inicia uma sessão no Appium com as configurações fornecidas
driver = webdriver.Remote("http://localhost:4723/wd/hub",
desired_caps)

# Aguarda a inicialização do app
time.sleep(2)

try:
    # Encontra o campo de entrada do usuário e insere um nome
    username_field = driver.find_element(AppiumBy.ID,
"com.example.appteste:id/editTextUsername")
    username_field.send_keys("TesteUsuario")

    # Clica no botão "Cadastrar"
    submit_button = driver.find_element(AppiumBy.ID,
"com.example.appteste:id/buttonSubmit")
    submit_button.click()

    # Aguarda para verificar a resposta (toast message ou mensagem
de sucesso)
    time.sleep(2)

    # Verifica a existência da mensagem de sucesso (dependendo da
implementação do app)
    print("Teste de cadastro executado com sucesso!")

finally:
    # Fecha a sessão do driver
    driver.quit()
```

Nota: No campo `"app": "<Caminho_do_APK>"`, substitua `<Caminho_do_APK>` pelo caminho real do APK gerado.

Passo 3: Executando o Teste

1. Navegue até o diretório onde o arquivo `test_login.py` está salvo.
2. Execute o teste com o seguinte comando:

```
python test_login.py
```

O Appium então iniciará o emulador (ou dispositivo físico, se configurado) e executará o teste automatizado de acordo com as instruções no código.

8. Executando o Teste

Para executar o teste, siga os passos:

- **Inicie o Emulador Android**:
  - Abra o Android Studio e inicie o emulador configurado.
- **Execute o Appium Server**:
  - No terminal, execute:

appium

  - Certifique-se de que o servidor Appium está ativo em http://localhost:4723/wd/hub.

Execute o Script de Teste no VS Code:

- No terminal do **VS Code**, navegue até a pasta mobile_test_project e execute:

python test_login.py

- O teste irá iniciar o aplicativo no emulador, preencher o formulário de login com dados de teste e verificar se a mensagem de erro está correta.

Modifique o Teste para um Login Válido:

- Alterar o script para simular um login bem-sucedido e verificar se o usuário é redirecionado para a tela principal do aplicativo.

TAREFA 4 – FINALIZAÇÃO

6. Salve todos os elementos das aulas de hoje no seu cartão MISSÃO 11
7. Coloque no fim o nome e RA dos alunos presentes na atividade no cartão de hoje;

8. Coloque o cartão na lista EM VALIDAÇÃO;