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

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

ATENÇÃO: vocês podem criar script para executar o teste do código de vocês

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.Instalação do Node.js

- Baixe e instale o **Node.js** (obrigatório para o Appium) a partir do site oficial.
- Confirme a instalação, abrindo o terminal do Windows (cmd) e digitando:

node -v
npm -v

2. Instalação do 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

3. Instalação do Android Studio e Configuração do Emulador Android

- Configure um emulador Android Studio:
  - Abra o Android Studio e vá em **Tools > AVD Manager**.
  - Crie um novo dispositivo virtual, selecionando um modelo de dispositivo e uma imagem de sistema Android.
  - Configure o emulador com Android 10 ou superior e inicie-o.

4. Configuração das Variáveis de Ambiente

- Adicione o caminho do SDK Android às variáveis de ambiente para garantir o funcionamento do ADB (Android Debug Bridge):
  - Encontre o caminho do SDK Android no Android Studio em **File > Settings > Appearance & Behavior > System Settings > Android SDK**.
  - Adicione as seguintes variáveis no **Painel de Controle > Sistema > Configurações Avançadas do Sistema > Variáveis de Ambiente**:
    - ANDROID_HOME: Caminho do SDK Android.
    - PATH: Inclua os caminhos para \platform-tools e \tools.

5. Instalação de Bibliotecas Python

- No terminal, 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

Use o seguinte código para criar um teste básico de login automatizado:

```
from appium import webdriver
import unittest
from time import sleep

class LoginTest(unittest.TestCase):

    def setUp(self):
        # Configurações para conexão com o emulador Android e Appium Server
        desired_caps = {
            "platformName": "Android",
            "platformVersion": "10",  # ajuste conforme a versão do emulador
            "deviceName": "Android Emulator",
            "automationName": "UiAutomator2",
            "app":  "<Caminho_do_APK_do_aplicativo_de_teste>"    #  Ex.:  "C:\\Apps\\app-teste.apk"
        }
        self.driver = webdriver.Remote("http://localhost:4723/wd/hub", desired_caps)
        sleep(5)  # Aguarda a inicialização do aplicativo

    def test_login(self):
        # Localize e interaja com elementos de login
        username_field = self.driver.find_element_by_id("com.example:id/username")
        password_field = self.driver.find_element_by_id("com.example:id/password")
        login_button = self.driver.find_element_by_id("com.example:id/login")
```

```
        # Preenche os campos de login e clica no botão de login
        username_field.send_keys("usuario_teste")
        password_field.send_keys("senha_incorreta")
        login_button.click()
        sleep(3)  # Aguarda a resposta

        # Verifica se o login falhou (exemplo de mensagem de erro)
        error_message = self.driver.find_element_by_id("com.example:id/error")
        self.assertEqual(error_message.text, "Usuário ou senha incorretos")

    def tearDown(self):
        # Fecha o aplicativo
        self.driver.quit()

if __name__ == "__main__":
    unittest.main()
```

8. Executando o Teste

Para executar o teste, siga os passos:

- **Inicie o Emulador Android**:
  - Abra o Android Studio, vá em **Tools > AVD Manager**, 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.

SEGUNDA ATIVIDADE DO DIA:
SE COMPLETAR S/ AJUDA GANHA MAIS 0,25

Envolve a configuração do **Appium** com o emulador **Android Studio** e a criação de um script em **Python** para testar a funcionalidade de cadastro de um usuário no aplicativo.

1. Pré-requisitos
Antes de iniciar, certifique-se de que:
- O **Node.js** e o **Appium** estão instalados e configurados (passos detalhados anteriormente).
- O **Android Studio** está configurado com um emulador Android ativo.
- O APK do aplicativo de teste está salvo no computador e seu caminho está anotado.

2. Configuração do Ambiente e Variáveis de Ambiente
- Siga as instruções de configuração do Android Studio, Appium e variáveis de ambiente detalhadas na atividade anterior.

3. Código do Teste de Cadastro
O código abaixo realiza a automação do cadastro de um novo usuário:

Estrutura do Projeto
- No **VS Code**, crie uma pasta chamada user_registration_test.
- Dentro dela, crie o arquivo test_registration.py.

Código do Teste
O código a seguir preenche os campos do formulário de cadastro e verifica se o usuário é direcionado para uma tela de confirmação.

```
from appium import webdriver
import unittest
from time import sleep

class RegistrationTest(unittest.TestCase):

    def setUp(self):
        # Configurações para o emulador e Appium
        desired_caps = {
            "platformName": "Android",
            "platformVersion": "10",  # ajuste conforme a versão do seu emulador
            "deviceName": "Android Emulator",
            "automationName": "UiAutomator2",
            "app": "<Caminho_do_APK_do_aplicativo_de_teste>"  # ex.: "C:\\Apps\\app-
teste.apk"
        }
        self.driver = webdriver.Remote("http://localhost:4723/wd/hub", desired_caps)
        sleep(5)  # Aguarda o aplicativo iniciar

    def test_user_registration(self):
        # Campos do formulário de cadastro
        first_name_field = self.driver.find_element_by_id("com.example:id/first_name")
        last_name_field = self.driver.find_element_by_id("com.example:id/last_name")
```

```
        email_field = self.driver.find_element_by_id("com.example:id/email")
        password_field = self.driver.find_element_by_id("com.example:id/password")
        confirm_password_field =
self.driver.find_element_by_id("com.example:id/confirm_password")
        register_button = self.driver.find_element_by_id("com.example:id/register")

        # Preenchimento dos campos do formulário
        first_name_field.send_keys("João")
        last_name_field.send_keys("Silva")
        email_field.send_keys("joao.silva@example.com")
        password_field.send_keys("senhaSegura123")
        confirm_password_field.send_keys("senhaSegura123")

        # Clica no botão de cadastro
        register_button.click()
        sleep(3)  # Aguarda a resposta

        # Verificação de sucesso do cadastro
        success_message =
self.driver.find_element_by_id("com.example:id/success_message")
        self.assertEqual(success_message.text, "Cadastro realizado com sucesso")

    def tearDown(self):
        # Fecha o aplicativo
        self.driver.quit()

if __name__ == "__main__":
    unittest.main()
```

4. Executando o Teste

Para executar o teste, siga estes passos:

1. **Inicie o Emulador Android**:
    - Abra o **Android Studio**, vá em **Tools > AVD Manager**, e inicie o emulador configurado.
2. **Execute o Servidor Appium**:
    - No terminal, execute:

```
Appium
```

    - Verifique que o Appium está disponível em http://localhost:4723/wd/hub.

3. **Execute o Script de Teste no VS Code**:
    - No terminal do **VS Code**, navegue até a pasta user_registration_test e execute:

```
python test_registration.py
```

Esse teste iniciará o aplicativo no emulador, preencherá o formulário de cadastro e verificará se uma mensagem de sucesso é exibida.

Desafios para Alunos

1. **Validação de E-mail**:
    - Modifique o teste para usar um e-mail inválido e verifique se o aplicativo exibe uma mensagem de erro, garantindo que a validação de e-mail está funcionando corretamente.
2. **Verificação de Senhas**:
    - Alterar o código para que as senhas não coincidam e verificar se o sistema impede o cadastro e exibe uma mensagem de erro de confirmação de senha.

TAREFA 4 – FINALIZAÇÃO

6. Salve todos os elementos das aula de hoje no seu
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;