FECAP

SKL – Aplicativo de gestão e divulgação da Fecap Social

Requisitos da disciplina Modelagem de Software e Arquitetura de Sistemas

São Paulo

2024

INTEGRANTES DO PROJETO e RA’S

Grupo 2

Ana Flavia Lorêdo	-	23025092
Fernanda Mayumi Kuba Kato	-	23024484
Kevin Makoto Shiroma	-	20020925
Renato Riichi Kato	-	23024516

Contents

1. INTRODUÇÃO

Tendo.....

2. Teste de Software

2.1. Apresentar 2 testes unitários.

2.1.1 Teste: Os get, set e construtor das classes

Nas figuras abaixo os testes unitários consistem na verificação das funcionalidades da classeUsuario.

2.1.1.1 Teste unitário 1: Este teste verifica a funcionalidade do construtor completo da classe `ClasseUsuario`. Primeiro, criamos um objeto `ClasseUsuario` com todos os atributos preenchidos: nome, email e senha.

Em seguida, verificamos se o objeto criado não é nulo usando o método `assertNotNull()`. Isso garante que o construtor esteja funcionando corretamente e que um objeto válido tenha sido criado.

Depois disso, usamos o método `assertEquals()` para verificar se os atributos do objeto `ClasseUsuario` foram definidos corretamente. Comparamos o nome, o email e a senha do objeto com os valores esperados ("Nome Teste", "email@teste.com" e "senha123", respectivamente).

Essa abordagem de teste garante que o construtor da classe `ClasseUsuario` esteja configurando corretamente os atributos do objeto com os valores fornecidos durante a criação. Isso ajuda a garantir a integridade dos dados e o correto funcionamento do código que depende desses objetos `ClasseUsuario`.

Figura 01: Teste unitário 1

```
@Test
public void testConstrutorCompleto() {
    // Criar um objeto ClasseUsuario com todos os atributos
    ClasseUsuario usuario = new ClasseUsuario( nome: "Nome Teste",  email: "email@teste.com",  senha: "senha123");

    // Verificar se o objeto não é nulo
    assertNotNull(usuario);

    // Verificar se os atributos foram definidos corretamente
    assertEquals( expected: "Nome Teste", usuario.getNome());
    assertEquals( expected: "email@teste.com", usuario.getEmail());
    assertEquals( expected: "senha123", usuario.getSenha());
}
```

2.1.1.2 Teste unitário 2: Este teste verifica a funcionalidade dos métodos setters e getters da classe `ClasseUsuario`.

Primeiro, criamos um objeto `ClasseUsuario` vazio, passando apenas o email como parâmetro no construtor.

Em seguida, usamos os métodos setters para definir os valores dos atributos `nome` e `senha`. Definimos o nome como "Novo Nome" e a senha como "novaSenha123".

Por fim, usamos os métodos getters para verificar se os valores dos atributos foram definidos corretamente. Utilizamos o método `assertEquals()` para comparar os valores retornados pelos getters com os valores esperados.

Essa abordagem de teste garante que os métodos setters e getters da classe `ClasseUsuario` estejam funcionando corretamente, permitindo a manipulação e obtenção dos valores dos atributos do objeto `ClasseUsuario`.

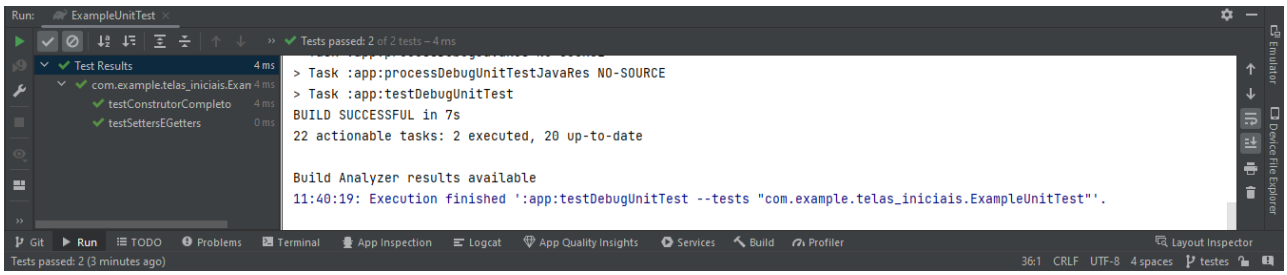
Figura 02: Teste unitário 2

```
@Test
public void testSettersEGetters() {
    // Criar um objeto ClasseUsuario vazio
    ClasseUsuario usuario = new ClasseUsuario( email: "email@teste.com");

    // Definir os valores dos atributos usando os setters
    usuario.setNome("Novo Nome");
    usuario.setSenha("novaSenha123");

    // Verificar se os getters retornam os valores corretos
    assertEquals( expected: "Novo Nome", usuario.getNome());
    assertEquals( expected: "email@teste.com", usuario.getEmail());
    assertEquals( expected: "novaSenha123", usuario.getSenha());

}
```

2.1.2 Resultado dos testes unitários

A figura abaixo, mostra o sucesso dos testes realizados:

Figura 03: Resultado dos testes unitários



2.2. Apresentar 2 testes de componentes

Nos testes de componentes foi feito os códigos usando o junit e também em formato de vídeo, que acompanha a documentação no Github com o nome:

Testequaldevops_Teste_componente

2.2.1 Teste: Login do usuário

Nas figuras abaixo os testes consistem em o usuário fazer o login com o email e a senha, e é retornado se o acesso teve sucesso ou erro.

2.2.1.1 Teste componente 1: Este teste verifica se o login é bem-sucedido no aplicativo. Primeiro, simulamos a inserção de um email digitando "skl.projetopi2@gmail.com" no campo de email e, em seguida, digitamos a senha "valid6" no campo de senha. Depois disso, clicamos no botão de login.

O objetivo é garantir que o usuário consiga fazer login com sucesso fornecendo um email e uma senha válidos. Este teste nos ajuda a verificar se o fluxo de login está funcionando corretamente e se os campos de email e senha estão aceitando os dados corretamente.

Figura 01: teste componente 1

```
@Rule
public ActivityScenarioRule<LoginActivity> activityScenarioRule = new ActivityScenarioRule<>(LoginActivity.class);

@Test
public void testLoginSucesso() {
    // Simula um input
    Espresso.onView(ViewMatchers.withId(R.id.inputEmailLogin)).perform(ViewActions.typeText( stringToBeTyped: "skl.projetopi2@gmail.com"));
    Espresso.onView(ViewMatchers.withId(R.id.inputSenhaLogin)).perform(ViewActions.typeText( stringToBeTyped: "validó"), ViewActions.closeSoftKeyboard());

    // Clica no botão de login
    Espresso.onView(ViewMatchers.withId(R.id.btnEntrar)).perform(ViewActions.click());

}
```

2.2.1.2 Teste componente 2: Este teste verifica o comportamento do aplicativo quando o login é inválido. Primeiro, simulamos a inserção de um email válido ("skl.projetopi2@gmail.com") no campo de email e uma senha inválida ("invalid") no campo de senha. Em seguida, fechamos o teclado virtual após a inserção da senha.

Depois disso, clicamos no botão de login para tentar efetuar o login com as credenciais fornecidas.

O objetivo deste teste é garantir que, quando as credenciais de login são inválidas, um alerta seja exibido ao usuário com o título "Erro de Login" e a mensagem "Email ou senha incorretos!!!". Além disso, o alerta deve conter um botão com o texto "Tentar novamente" para que o usuário possa tentar fazer login novamente.

Figura 02: teste componente 2

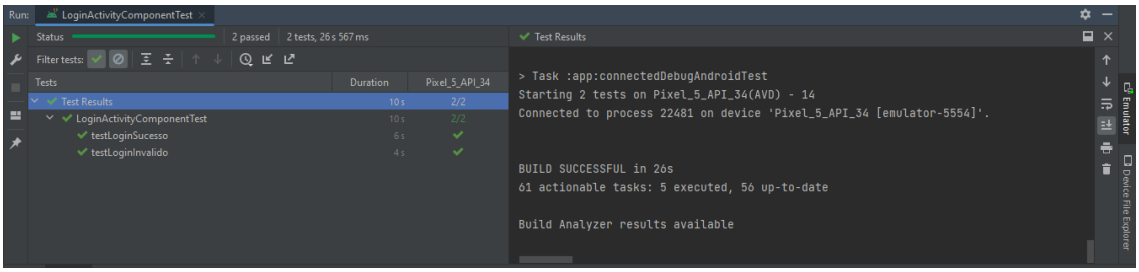
```
@Test
public void testLoginInvalido() {
    // Simula um input
    Espresso.onView(ViewMatchers.withId(R.id.inputEmailLogin)).perform(ViewActions.typeText( stringToBeTyped: "skl.projetopi2@gmail.com"));
    Espresso.onView(ViewMatchers.withId(R.id.inputSenhaLogin)).perform(ViewActions.typeText( stringToBeTyped: "invalid"), ViewActions.closeSoftKeyboard());

    // Clica no botão de login
    Espresso.onView(ViewMatchers.withId(R.id.btnEntrar)).perform(ViewActions.click());

    // Verifica se o alerta é exibido com o titulo correto
    Espresso.onView(ViewMatchers.withText("Erro de Login")).inRoot(RootMatchers.isDialog()).check(ViewAssertions.matches(ViewMatchers.isDisplayed()));

    // Verifica se o alerta contém a mensagem correta
    Espresso.onView(ViewMatchers.withText("Email ou senha incorretos!!!")).inRoot(RootMatchers.isDialog()).check(ViewAssertions.matches(ViewMatchers.isDi

    // Verifica se o botão no alerta contém o texto correto
    Espresso.onView(ViewMatchers.withText("Tentar novamente")).inRoot(RootMatchers.isDialog()).check(ViewAssertions.matches(ViewMatchers.isDisplayed()));
}
```

Essas verificações são feitas usando as funções do Espresso para verificar se o alerta é exibido corretamente com o título, mensagem e botão esperados. Isso ajuda a garantir que o aplicativo esteja fornecendo feedback apropriado ao usuário quando as credenciais de login são inválidas.

2.2.2 Resultado dos testes de componente

A figura abaixo, mostra o sucesso dos testes realizados:

Figura 3: Resultado dos testes de componente



2.3. Apresentar um teste de sistema.

O teste de sistema foi realizado em formato de vídeo e acompanha a documentação no Github com o nome:

Testequaldevops_Teste_sistema

3. Qualidade de Software

3.1. Indicar 4 atributos de qualidade de software e informar como foi aplicado no projeto integrador (PI)

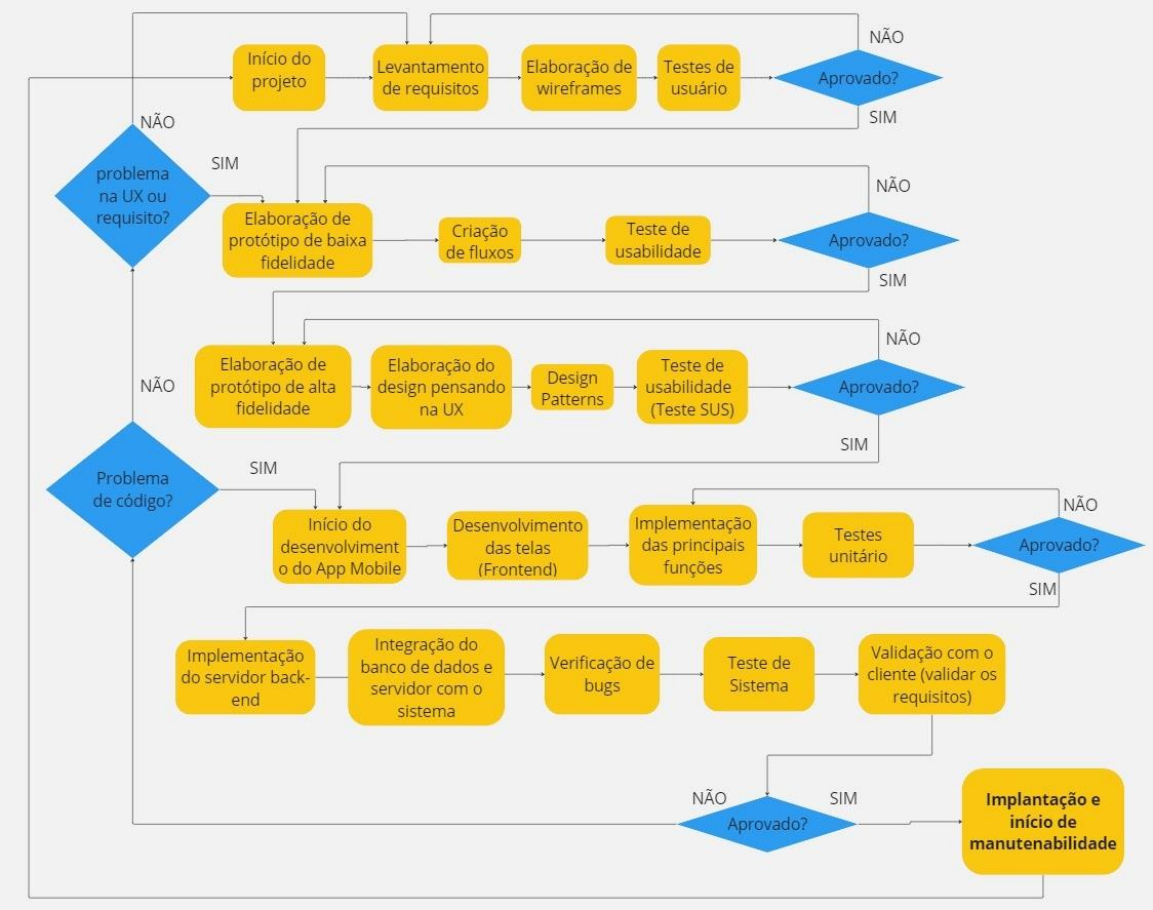
3.2. Apresentar um Modelo que qualidade de software

Este modelo segue o ISO/IEC 25010 de qualidade de software e as seguintes características foram destacadas:

Qualidade de Software		
Adequação Funcional	**Capacidade de Interação**	**Confiabilidade**
Completude Funcional: O APP não possui funções complexas, possui caráter informativo acerca dos aventos feito pela FECAP Social. O APP tem como função abordar essa pretensão.	**Reconhecibilidade de adequação:** Mesmo ao instalar ao APP o usuário rapidamente já sabe se ele irá satisfazer suas necessidades ou não, pois o único objetivo do app é: informar ao usuario sobre a FECAP Social e seus eventos futuros.	**Disponibilidade:** O APP precisa estar disponível sempre que um usuário quiser verificar os eventos próximos da FECAP Social.

3.3. Apresentar um Processo (plano) de gerenciamento de qualidade de software

O processo de avaliação é baseado na ISO 25010 e baseada no modelo acima:



4. REFERÊNCIAS BIBLIOGRÁFICAS

SOMMERVILLE, I. **Engenharia de Software.** 11ª Edição. São Paulo: Pearson Addison-Wesley, 2017.