Testes Integrados

Neste módulo, vamos introduzir os conceitos fundamentais dos testes de software, com foco nos testes integrados. Discutiremos a importância dos testes de software, os diferentes tipos de testes (unitários, de integração e de sistema) e como eles se aplicam ao desenvolvimento de aplicativos. Entender a importância e os tipos de testes é essencial para garantir a qualidade e a robustez do software.

Introdução aos Testes de Software

O que são Testes de Software?

Testes de software são processos que verificam se um aplicativo ou sistema funciona conforme o esperado. Eles ajudam a identificar bugs, garantir a qualidade e melhorar a confiança no software.

Importância dos Testes de Software

- **Qualidade:** Asseguram que o software atenda aos requisitos especificados.

- **Confiabilidade:** Aumentam a confiança na estabilidade do software.

- **Manutenibilidade:** Facilitam a manutenção e a evolução do software.

- **Custo-efetividade:** Identificam problemas cedo, reduzindo custos de correção.

Tipos de Testes de Software

1. Testes Unitários:

Testam componentes individuais isoladamente.

Exemplo: Testar uma função específica.

2. Testes de Integração:

Verificam a interação entre diferentes componentes.

Exemplo: Testar a comunicação entre dois módulos.

3. Testes de Sistema:

Avaliam o sistema como um todo.

Exemplo: Testar o aplicativo completo em um ambiente simulado.

Diferença entre Testes Unitários e Testes Integrados

Testes Unitários:

- Focados em componentes individuais.

- Rápidos e de fácil execução.

- Identificam problemas específicos.

Testes Integrados:

- Focados na interação entre componentes.

- Mais complexos e demorados.

- Verificam a funcionalidade global do sistema.

Nesta aula, aprenderemos como configurar o ambiente de testes para um projeto Flutter. Veremos como adicionar dependências de testes, configurar arquivos de teste e utilizar ferramentas de teste fornecidas pelo Flutter. A configuração adequada do ambiente de testes é crucial para realizar testes eficazes e eficientes.

Configurando o Ambiente de Testes no Flutter

Adicionando Dependências de Testes

1. Adicionar Dependências ao pubspec.yaml

```
dev_dependencies:
  flutter_test:
    sdk: flutter
  integration_test:
    sdk: flutter
```

2. Instalar Dependências

```
flutter pub get
```

Estrutura do Projeto de Teste

1. Criar Diretório de Testes

Crie um diretório chamado test na raiz do projeto.

2. Configurar Arquivo de Teste

Crie um arquivo de teste dentro do diretório test, por exemplo, test/integration_test.dart.

Escrevendo um Teste Simples

1. Código de Teste Simples

```
import 'package:flutter_test/flutter_test.dart';

void main() {

  test('Teste Simples', () {

    var resultado = 2 + 2;

    expect(resultado, 4);

  });

}
```

Explicação: Este teste verifica se a soma de 2 + 2 resulta em 4.

2. Executar o Teste

```
flutter test
```

Nesta aula, vamos nos concentrar em escrever testes integrados em Flutter. Aprenderemos a criar testes que verificam a interação entre diferentes componentes do aplicativo, como widgets e navegação. Os testes integrados são fundamentais para garantir que o aplicativo funcione corretamente em cenários do mundo real.

Escrevendo Testes Integrados

Criando Testes Integrados

1. Adicionar Dependência de Teste Integrado

```
dev_dependencies:
  integration_test:
    sdk: flutter
```

2. Criar Arquivo de Teste Integrado

Crie um arquivo chamado test_driver/app_test.dart.

Exemplo de Teste Integrado

1. Código de Teste Integrado

```
import 'package:flutter/material.dart';
import 'package:flutter_test/flutter_test.dart';
import 'package:integration_test/integration_test.dart';
import 'package:my_app/main.dart' as app;
void main() {
  IntegrationTestWidgetsFlutterBinding.ensureInitialized();
  testWidgets('Teste Integrado Simples', (WidgetTester tester) async {
    app.main();
    await tester.pumpAndSettle();
    // Verifica se o título do aplicativo está presente
    expect(find.text('Meu App'), findsOneWidget);
    // Interage com um botão
```

```
    await tester.tap(find.byIcon(Icons.add));

    await tester.pumpAndSettle();

    // Verifica se o contador aumentou

    expect(find.text('1'), findsOneWidget);

  });

}
```

Explicação: Este teste integrado inicializa o aplicativo, verifica a presença de um título, interage com um botão e verifica a alteração no contador.

2. Executar o Teste Integrado

```
flutter drive --target=test_driver/app_test.dart
```

Nesta aula, abordaremos a automatização e a manutenção de testes de software. Discutiremos como configurar pipelines de CI/CD (Integração Contínua/Entrega Contínua) para executar testes automaticamente, garantir que os testes sejam mantidos e atualizados regularmente e como analisar os resultados dos testes para melhorar continuamente a qualidade do software.

Automatização e Manutenção de Testes

Configuração de Pipelines de CI/CD

1. Escolher uma Ferramenta de CI/CD

Exemplos: GitHub Actions, GitLab CI, Travis CI.

2. Configuração do Workflow (Exemplo com GitHub Actions)

Crie um arquivo .github/workflows/flutter.yml.

```
name: Flutter CI

on: [push, pull_request]

jobs:

  build:

    runs-on: ubuntu-latest

    steps:

    - uses: actions/checkout@v2

    - uses: subosito/flutter-action@v1

      with:

        flutter-version: '2.2.3' # Coloque a versão do Flutter desejada

    - run: flutter pub get

    - run: flutter test

    - run: flutter drive --target=test_driver/app_test.dart
```

Explicação: Este workflow configura GitHub Actions para executar testes unitários e integrados automaticamente em cada push ou pull request.

Manutenção de Testes

1. Revisar Testes Regularmente

- Atualize os testes conforme novas funcionalidades são adicionadas.

- Remova ou modifique testes obsoletos.

2. Garantir Cobertura de Testes

- Utilize ferramentas para medir a cobertura de testes.

- Adicione testes para áreas críticas do código.

3. Analisar Resultados de Testes

- Revise os resultados dos testes após cada execução.

- Corrija falhas imediatamente para manter a qualidade do código.

Exemplo de Análise de Resultados de Testes

1. Código para Análise de Resultados

```
void main() {
  test('Exemplo de Análise de Resultados', () {
    var resultado = funcaoParaTestar();
    expect(resultado, isNotNull);
    expect(resultado, equals('Resultado Esperado'));
  });
}
String funcaoParaTestar() {
  return 'Resultado Esperado';
}
```

Explicação: Este teste verifica se a função retorna um resultado não nulo e igual ao esperado.

Esses exemplos e explicações fornecem uma base sólida para iniciantes em Flutter aprenderem a realizar testes integrados. Para mais detalhes, consulte a documentação oficial do Flutter: Flutter Documentation.

Materiais Extras

Você pode realizar o download do arquivo contendo os materiais extras utilizados ao longo das aulas por meio do seguinte link: https://drive.google.com/file/d/1mg7lqMl8Pt2zl0rHIsFS0Qew00YN-sEX/view?usp=sharing.

Conteúdo Bônus

Documentação Oficial do Android Developers: O site oficial do Android Developers oferece uma seção dedicada a treinamentos, incluindo tópicos sobre testes e qualidade, que abrangem desde noções básicas até práticas avançadas no desenvolvimento para Android.

Esse recurso fornecerá uma base sólida para a implementação de testes integrados em projetos de desenvolvimento mobile, essenciais para garantir a qualidade e a confiabilidade das aplicações.

Referências Bibliográficas

BOYLESTAD, R. L.; NASHELSKY, L. Dispositivos Eletrônicos e Teoria de Circuitos. 11. ed. Pearson, 2013.

DEITEL, P. J.; DEITEL, H. M. Ajax, Rich Internet Applications e Desenvolvimento Web para Programadores. Pearson, 2008.

DUARTE, W. Delphi para Android e iOS: Desenvolvendo Aplicativos Móveis. Brasport, 2015.

FELIX, R.; SILVA, E. L. da. Arquitetura para Computação Móvel. 2. ed. Pearson, 2019.

LEE, V.; SCHNEIDER, H.; SCHELL, R. Aplicações Móveis: Arquitetura, Projeto e Desenvolvimento. Pearson, 2005.

MARINHO, A. L.; CRUZ, J. L. da. Desenvolvimento de Aplicações para Internet. 2. ed. Pearson, 2019.

MOLETTA, A. Você na Tela: Criação Audiovisual para a Internet. Summus, 2019.

SILVA, D. (Org.) Desenvolvimento para dispositivos móveis. Pearson, 2017.

Ir para exercício