



Flutter Avançado - Aula 2

Exercícios 4 a 6



- 10 melhores extensões para VS Code e Flutter

[Ver mais](#)



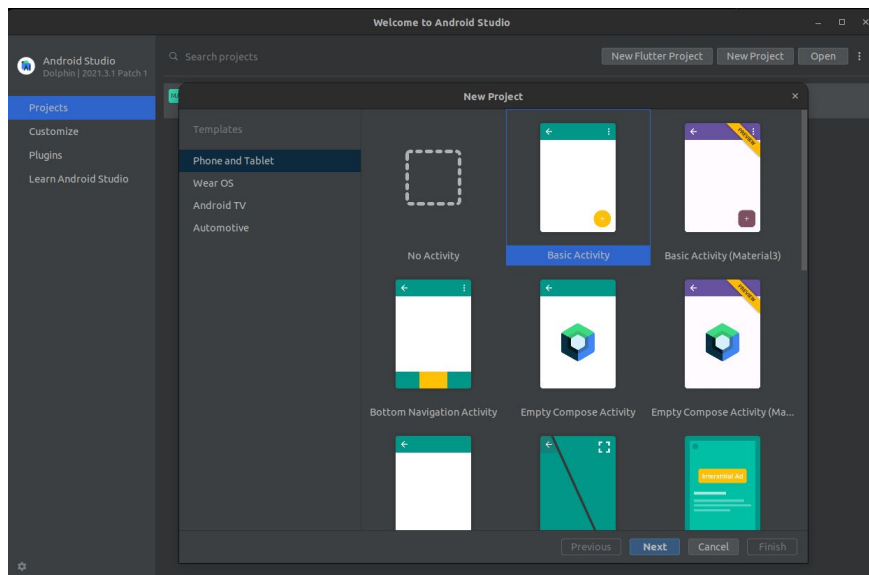


- É possível adicionar código Flutter em aplicações Android ou iOS nativas
- Isso é interessante para projetos já existentes, onde esteja sendo feita a migração para Flutter
- Nos focaremos aqui na criação de um módulo em uma aplicação Android





- Primeiro será necessário criar um projeto Android, pelo Android Studio





- Após criar o projeto Android, adicionaremos um módulo Flutter ao projeto
- 1ª forma: Pelo Menu: File / New / New Module
- 2ª forma: Via linha de comando:

```
flutter create -t module --org com.example my_flutter
```





- O próximo passo será setar o nosso projeto Android para que seja compilado com a versão 11 do Java
- No arquivo build.gradle (./app/build.gradle), altere os seguintes dados:

```
android {  
    //...  
    compileOptions {  
        sourceCompatibility 11  
        targetCompatibility 11  
    }  
}
```





- O próximo passo será necessário adicionar o módulo Flutter como uma dependência do projeto Android
- Isso pode ser feito de duas formas
- 1ª forma: Usando AAR (Android Archives)
- 2ª forma: Adicionando dependência a partir do código fonte





- Usando AAR (Android Archives): Dessa forma, o módulo Flutter será empacotado como um pacote Maven
- A vantagem é que outros desenvolvedores não precisarão ter o Flutter instalado em suas máquinas
- Para compilar o módulo como um pacote AAR, execute o comando abaixo

```
flutter build aar
```



- Adicionando a dependência no arquivo app/build.gradle

```
android {  
    // ...  
}  
  
repositories {  
    maven {  
        url 'some/path/my_flutter/build/host/outputs/repo'  
        // This is relative to the location of the build.gradle file  
        // if using a relative path.  
    }  
    maven {  
        url 'https://storage.googleapis.com/download.flutter.io'  
    }  
}  
  
dependencies {  
    // ...  
    debugImplementation 'com.example.flutter_module:flutter_debug:1.0'  
    profileImplementation 'com.example.flutter_module:flutter_profile:1.0'  
    releaseImplementation 'com.example.flutter_module:flutter_release:1.0'  
}
```





- Desta segunda forma, sempre que o projeto Android for compilado, nosso módulo também será
- Essa forma é interessante para quando você estiver constantemente alterando seu módulo Flutter e precisar testá-lo
- Porém todos os desenvolvedores que trabalham no projeto precisarão do Flutter instalado



- Vamos incluir o módulo flutter como um subprojeto do projeto Android, em settings.gradle

```
include ':app' // Linha existente
setBinding(new Binding([gradle: this])) // nova linha
evaluate(new File( // nova linha
    settingsDir.parentFile, // nova linha
    'test_flutter/.android/include_flutter.groovy' // nova linha
)) // nova linha
```

- No arquivo app/build.gradle, adicione o seguinte trecho de código

```
dependencies {
    implementation project(':flutter')
}
```





- Registrando a atividade Flutter, no AndroidManifest.xml

```
<activity
  android:name="io.flutter.embedding.android.FlutterActivity"
  android:theme="@style/LaunchTheme"

  android:configChanges="orientation|keyboardHidden|keyboard|screenSize|locale|layoutDirection|fontScale|screenLayout|density|uiMode"
  android:hardwareAccelerated="true"
  android:windowSoftInputMode="adjustResize"
/>
```





- No fonte da atividade que chamará o Flutter

```
import io.flutter.embedding.android.FlutterActivity;

binding.fltn.setOnClickListener {
    startActivity(
        FlutterActivity.createDefaultIntent(view.context)
    )
}
```





- Podemos adicionar views e telas nativas dentro de nossas aplicações Flutter
- Isso possibilita adicionarmos funcionalidades nativas ao nosso app, como Google Maps, por exemplo
- Para Android, podemos utilizar duas metodologias diferentes:
- Composição híbrida
- Display virtual





- Utilizando a composição híbrida, a view será adicionada à árvore do Flutter
- Em versões anteriores a Android 10 podem ocorrer uma redução da taxa de FPS na execução da aplicação



```
import 'package:flutter/foundation.dart';
import 'package:flutter/gestures.dart';
import 'package:flutter/material.dart';
import 'package:flutter/rendering.dart';
import 'package:flutter/services.dart';

Widget build(BuildContext context) {
  // This is used in the platform side to register the view.
  const String viewType = '<platform-view-type>';
  // Pass parameters to the platform side.
  const Map<String, dynamic> creationParams = <String, dynamic>{};

  return PlatformViewLink(
    viewType: viewType,
    surfaceFactory:
      (context, controller) {
        return AndroidViewSurface(
          controller: controller as AndroidViewController,
          gestureRecognizers: const <Factory<OneSequenceGestureRecognizer>>{},
          hitTestBehavior: PlatformViewHitTestBehavior.opaque,
        );
      },
    onCreatePlatformView: (params) {
      return PlatformViewsService.initSurfaceAndroidView(
        id: params.id,
        viewType: viewType,
        layoutDirection: TextDirection.ltr,
        creationParams: creationParams,
        creationParamsCodec: const StandardMessageCodec(),
        onFocus: () {
          params.onFocusChanged(true);
        },
      )
      ..addOnPlatformViewCreatedListener(params.onPlatformViewCreated)
      ..create();
    },
  );
}
```





- Renderizará a view Android dentro de uma textura
- Algumas funcionalidades de teclado ou de acessibilidade podem não funcionar



```
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';

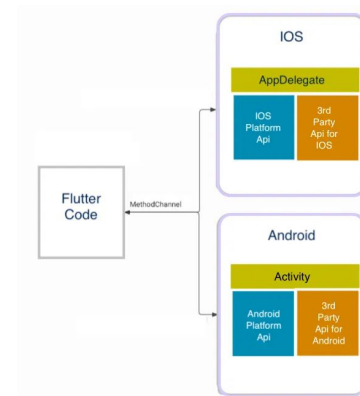
Widget build(BuildContext context) {
  // This is used in the platform side to register
  the view.
  const String viewType = '<platform-view-type>';
  // Pass parameters to the platform side.
  final Map<String, dynamic> creationParams =
  <String, dynamic>{};

  return AndroidView(
    viewType: viewType,
    layoutDirection: TextDirection.ltr,
    creationParams: creationParams,
    creationParamsCodec: const
    StandardMessageCodec(),
  );
}
```





- Podemos executar código nativo a partir do nosso código Flutter
- Para que isso seja possível, precisamos utilizar MethodChannel
- O MethodChannel possibilita o envio e recebimento de mensagens entre o código nativo e o Flutter





- Criando um MethodChannel e enviando mensagens

```
static const platform = MethodChannel('test_flutter_proway');

ElevatedButton(onPressed: () async {
  int vrLevel = await platform.invokeMethod("battery_level");


  setState(() {
    this._batteryLevel = vrLevel;
  });
}, child: Text("Nível da bateria")),
```

- No código nativo, vamos ficar ouvindo o MethodChannel

```
override fun configureFlutterEngine(@NonNull flutterEngine: FlutterEngine) {
    super.configureFlutterEngine(flutterEngine)
    MethodChannel(flutterEngine.dartExecutor.binaryMessenger,
CHANNEL).setMethodCallHandler {
        call, result ->
        if (call.method == "helloWorld")
            result.success("Teste 123");

        if (call.method == "battery_level")
            result.success(this.getBatteryLevel());
    }
}
```

- Recuperando o nível atual da bateria



```
private fun getBatteryLevel(): Int {
    val batteryLevel: Int
    if (VERSION.SDK_INT >= VERSION_CODES.LOLLIPOP) {
        val batteryManager = getSystemService(Context.BATTERY_SERVICE) as
        BatteryManager
        batteryLevel =
        batteryManager.getIntProperty(BatteryManager.BATTERY_PROPERTY_CAPACITY)
    } else {
        val intent =
        ContextWrapper(applicationContext).registerReceiver(null,
        IntentFilter(Intent.ACTION_BATTERY_CHANGED))
        batteryLevel = intent!!.getIntExtra(BatteryManager.EXTRA_LEVEL, -1)
        * 100 / intent.getIntExtra(BatteryManager.EXTRA_SCALE, -1)
    }

    return batteryLevel
}
```

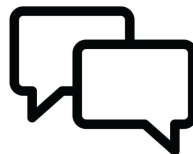
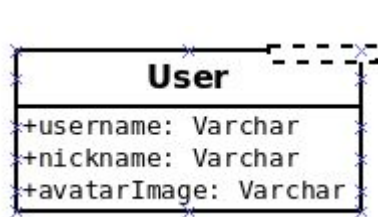
- Importando os arquivos necessários no código nativo

```
import androidx.annotation.NonNull
import io.flutter.embedding.android.FlutterActivity
import io.flutter.embedding.engine.FlutterEngine
import io.flutter.plugin.common.MethodChannel

import android.content.Context
import android.content.ContextWrapper
import android.content.Intent
import android.content.IntentFilter
import android.os.BatteryManager
import android.os.Build.VERSION
import android.os.Build.VERSION_CODES
```



- Vamos criar um chat!
- No nosso chat, teremos dois cadastros diferentes: Usuários e Mensagens
- Neste exercício, vamos criar um projeto no Firebase
- Você deve criar também os modelos e repositórios





- O próximo passo agora é listar todos os usuários, ao iniciar o app
- Devem ser listados os usuários e a imagem deles

Página inicial



Duwe
fernando_duwe



Jacklane e Manda Brasa
jacklane.mandabrasa





- O próximo passo é criar a tela de chat, listando as mensagens trocadas entre o seu usuário e o usuário selecionado na lista

