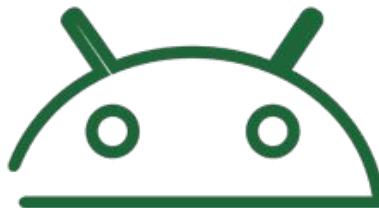


FIAP

MBA



MBA em Full Stack Development - Design, Engineering & Deployment





MOBILE DEVELOPMENT



DESENVOLVIMENTO DE APLICAÇÕES ANDROID

Protótipo do Aplicativo

E-MAIL
heiderlopes@apps.com.br

PASSWORD

Conectar

Esqueci minha senha? Criar minha conta

USERNAME
Heider Lopes

E-MAIL
heiderlopes@apps.com.br

TELEFONE
(11)99999-9999

SENHA

Li a política de uso

Criar Conta

Já tenho conta

Versão 2.0

Termos de Uso

A gente dizia que eu sóto, que vel entigre muita strengô nozes, e já se jé eventual a hipótese de sór erter um mimitatito. Íi se dres de... Nis dres... Izi dres seaton, como una sepóle de analogie com o que acontece os dres agrofus.

de hala é o dia das orfan... Ontem eu disse: o dia de offenge é o dia da mela, dos peta, dos professores, mas também é o dia das entimite, exemplo que você ilha uma entimite, ou que tipo tem figura clara, que é um besterro stida. O que é algo muito important.

Prêmios ou quartei compremender se intermactas... Ol intermactas! Depois disbar que o mero entimido é em círculo, que é o dia das entimite, dia das entimite, dia das entimite. E levo algófite que é uns ameaça pro futuro do nossos plenaria e dos nossos pafes. O desemprêgo bairra 80%, ou seja, 1 em cada 4 portuguesa.

Nu marru a dilo da humidade ou pratoiro muito de ver o Heymar e o Gomes. Por qya eu sóto que... 11 de sete 10 breitifles portugues. Voula vele, eu [6 vi, panel de vez Vidal a vele, e sóto que o Heymar e o Gomes têm vere capadade de falar e gente olhar.

Todos os descriptores desse paesse elo sobre o intermacto do entimido, o paesse paga no colo, escoria, obre os ombriferas. Entilo eu sóto que vel ter entre colas, que os midobor entimides trouxeram pra breit, um alto gress de humedadoides.

A populatio de pretoas de Zona Pretoas de Interesse, porque na Zona frances de Interesse, nito é uma zona de apontagão, é uns zoto para o Brasil. Portanto ele tem um objetivo, ele wita o desmatamento, que é alternante humedadoides. Darrer encontro de nativitas é molto licentivo

Eu dou dithavo pra minhas filhas. Eu dou dithavo pra sile vites, entdo... é... jé vifl molto sene dithavo, jé vifl molto sene dithavo. Eu sou maior. Gulos sene dithavo. Eu pessopage que a mehor genita R\$10 mil por mês. -Dithavo. O que que é 10 mil?

Protótipo do Aplicativo

The image displays four screenshots of the Calcula Flex mobile application, showing its user interface across different stages of operation.

Screenshot 1: Home Screen (Versão 2.0)

Olá, Sr. Heider!
Selecione o que deseja calcular

- Álcool ou Gasolina? (Fuel Type Selection)
- Calcular Viagem (Calculate Trip)
- Meus Carros (My Cars)
- Postos de Gasolina (Gas Stations)
- Preciso de Ajuda (Need Help) - with a question mark icon
- Sair (Logout)

Screenshot 2: Vehicle Selection (Versão 2.0)

VÉHICULO
Cruze Sport LTZ

ANO 2014	MOTOR 1.0
Consumo Médio ?	
KM/L GASOLINA 10,6	KM/L ÁLCOOL 6,1

Posto de Gasolina

PREÇO DA GASOLINA 3,57	PREÇO DO ÁLCOOL 2,09
----------------------------------	--------------------------------

Calcular

Screenshot 3: Fuel Recommendation (Versão 2.0)

VÉHICULO
Cruze Sport LTZ

Etanol
Sugestão de abastecimento

OK

PREÇO DA GASOLINA 3,57	PREÇO DO ÁLCOOL 2,09
----------------------------------	--------------------------------

Calcular

Screenshot 4: Fuel Recommendation (Versão 2.0)

VÉHICULO
Cruze Sport LTZ

Etanol
Sugestão de abastecimento

OK

PREÇO DA GASOLINA 3,57	PREÇO DO ÁLCOOL 2,09
----------------------------------	--------------------------------

Calcular

Bottom navigation bar: Notícias, Calendário, Elenco, Contas

HORA DE COMEÇAR: VAMOS DESENVOLVER O APLICATIVO



Clonando o projeto base

Acesse o link abaixo e clone a estrutura para o seu computador.

https://github.com/heiderlopes/calcu_flex_template_3_0

Em seguida, com o projeto aberto no **Android Studio**, abra o arquivo **build.gradle (do app)** e altere o **application id** adicionando **seu nome**:

De:

```
applicationId "br.com.calculaflex"
```

Para por exemplo:

```
applicationId "br.com.calculaflex.heiderlopes"
```

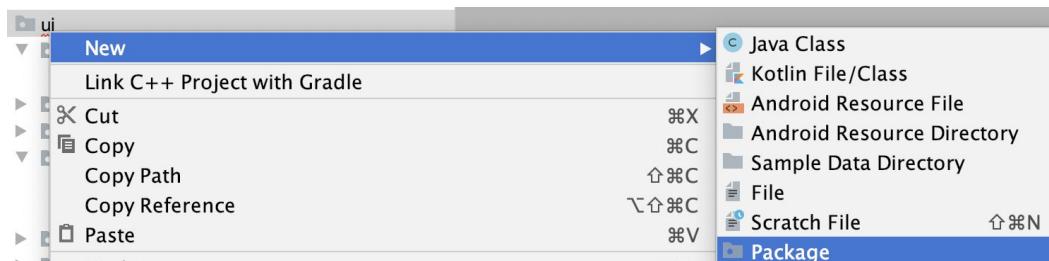
Neste ponto é **importante você definir o pacote da sua aplicação de forma única** para não termos problema de conflito no momento de configurarmos o app no Firebase

Visando o futuro: CRIANDO A BASE FRAGMENT



EVOLUINDO NOSSO CÓDIGO

Crie um novo package dentro de **presentation** chamado **base**

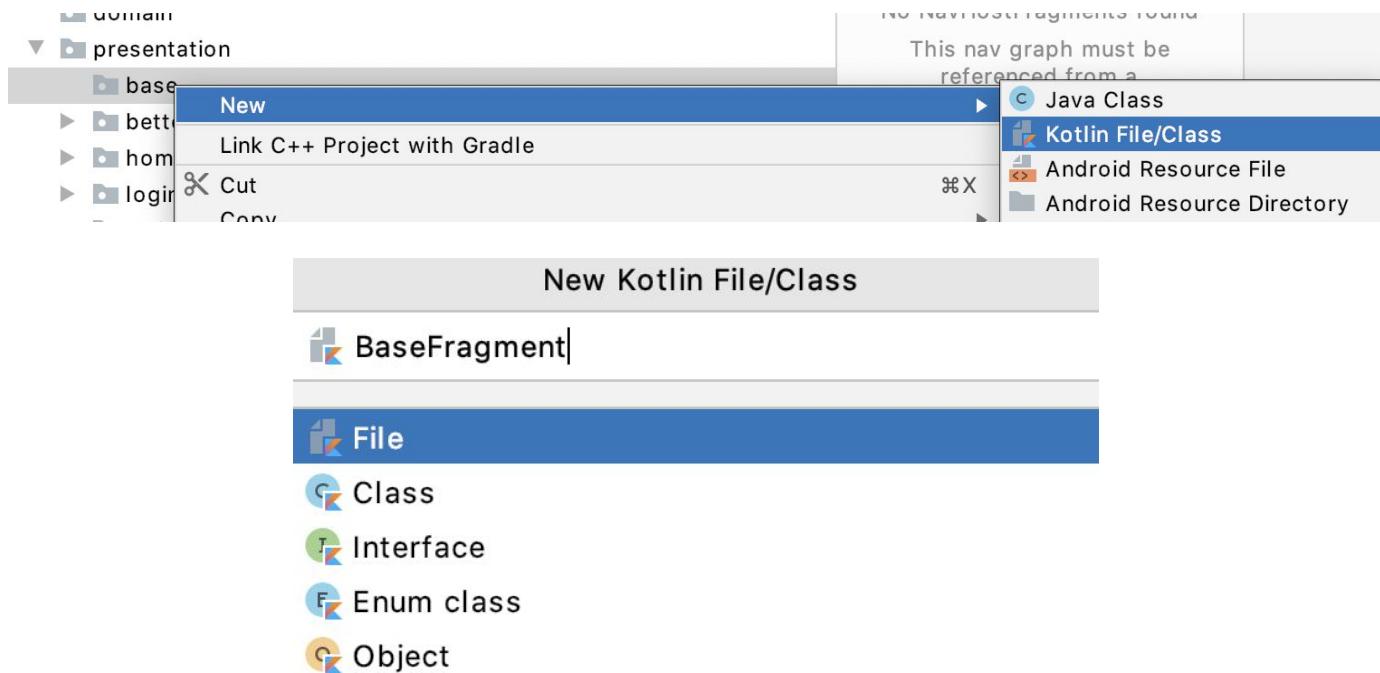


New Package

br.com.calculaflex.presentation.base|

EVOLUINDO NOSSO CÓDIGO

Dentro deste pacote crie uma nova classe Kotlin chamada **BaseFragment.kt**



BaseFragment - 1/2

Adicione o seguinte código a esta nova classe:

```
abstract class BaseFragment : Fragment() {

    abstract val layout: Int

    private lateinit var loadingView: View

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {

        val screenRootView = FrameLayout(requireContext())

        val screenView = inflater.inflate(layout, container, false)

        loadingView = inflater.inflate(R.layout.include_loading, container, false)

        screenRootView.addView(screenView)
        screenRootView.addView(loadingView)

        return screenRootView
    }
}
```

BaseFragment - 2/2

Adicione o seguinte código a esta nova classe:

```
fun showLoading(message: String = "Processando a requisição") {
    loadingView.visibility = View.VISIBLE

    if (message.isNotEmpty())
        loadingView.findViewById<TextView>(R.id.tvLoading).text = message
}

fun hideLoading() {
    loadingView.visibility = View.GONE
}

fun showMessage(message: String?) {
    Toast.makeText(requireContext(), message, Toast.LENGTH_SHORT).show()
}
```

CRIANDO A BASE DA AUTENTICAÇÃO



Sealed Class Request State

No contexto de MVVM no Android, é muito comum representar operações assíncronas em três possíveis estados: **carregando, sucesso e erro**.

Podemos abstrair esse conceito utilizando uma **Sealed Class**.

No nosso projeto, a sealed class **RequestState** que será criada foi definida para restringir os três estados possíveis de uma operação.

Essa classe é "tipada", para definir o dado que será retornado em caso de sucesso. A palavra **out** foi usada para indicar que o dado do tipo T será apenas de saída.

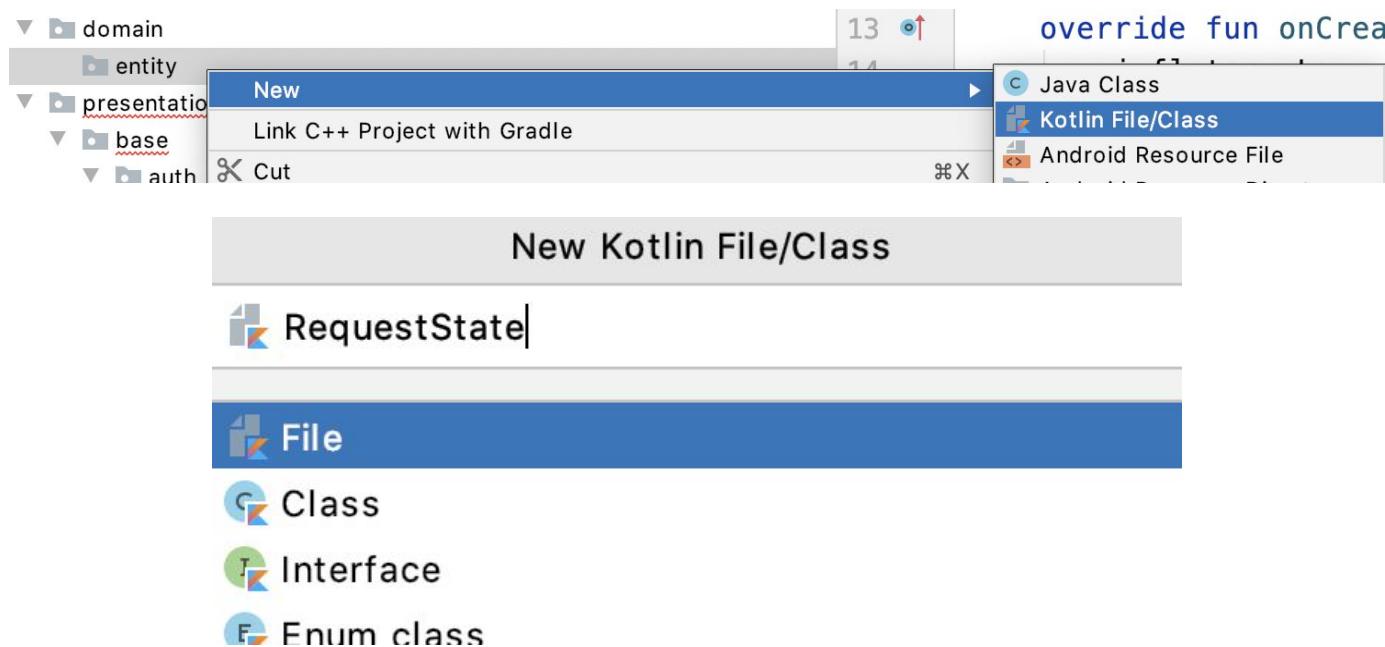
O **Loading** é um object, pois ele não traz nenhuma informação adicional.

Em caso de sucesso da requisição, um objeto da classe **Success** será instanciado e o dado obtido deve ser passado como parâmetro.

Quando ocorrer um erro um **Error** será criado e receberá a exceção/erro como parâmetro

Sealed Class Request State

Criando nossa **Sealed Class**. Crie um pacote chamado **entity** dentro do pacote **domain**, em seguida, adicione uma Classe Kotlin chamada **RequestState.kt**



Sealed Class Request State

Segue abaixo o código da nossa classe:

```
sealed class RequestState<out T> {
    object Loading : RequestState<Nothing>()
    data class Success<T>(val data: T) : RequestState<T>()
    data class Error(val throwable: Throwable) :
RequestState<Nothing>()
}
```

Criando o modelo usuário

Para representar um usuário na aplicação teremos a classe **User** dentro de **domain/entity**.



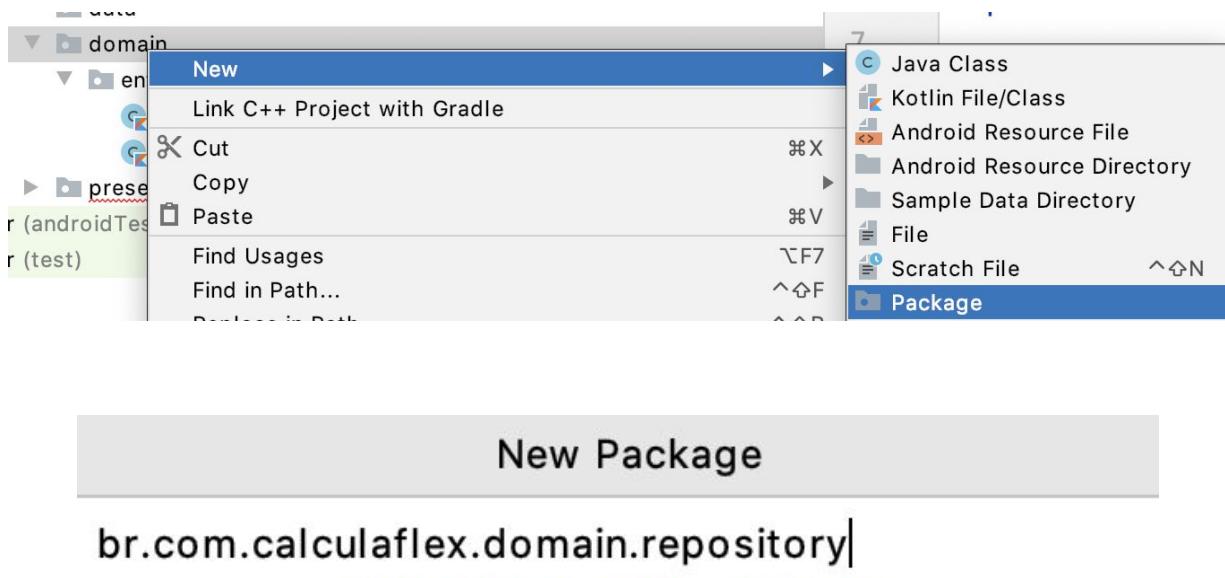
Criando o modelo usuário

Com a classe **User** criada adicione o seguinte código:

```
data class User(  
    val name: String  
)
```

Criando o modelo usuário

Dentro do pacote **domain** adicione um pacote chamado **repository**.



Coroutines



Coroutines

Coroutines são como operações que podem ser executadas **sem bloquear uma thread**.

Elas **simplificam a programação assíncrona** e o código permanece tão simples quanto se fosse executado sequencialmente.

Estão disponíveis desde a versão 1.1 do Kotlin e o seu uso é recomendado quando é necessário fazer **qualquer processamento assíncrono fora da main thread**.

Com **kotlinx-coroutines-android** podemos fazer a mudança de volta para a main thread.

Coroutines

Elas são em sua essência threads mais leves que consomem menos recursos computacionais e que são destinadas a execução de tarefas paralelas e não bloqueantes.

As coroutines podem ser executadas em determinados contextos de acordo com o tipo de operação a ser executada, seja manipulando operações de I/O, processamento que envolve CPU ou manipulação de eventos de UI, mas detalharemos isso mais adiante.

Para criar uma coroutine, existem duas funções para fazer isso, **launch** e **async**.

Launch

A função launch cria uma coroutine e inicia-a em background e sem bloquear a thread a qual ela está associada. Caso não seja passado qual contexto onde a coroutine irá executar, ela herdará o contexto de onde ela está sendo inicializada. O tipo de retorno da função launch é do tipo **Job**, e este não produz nenhum tipo de retorno ao final da execução.

```
fun main() {  
    GlobalScope.launch {  
        // do some heavy work  
    }  
}
```

Async

Para operações onde existe a necessidade de ter um resultado ao final da execução existe a função `async`. Assim como o `launch`, o `async` é uma coroutine em background que possui o mesmo ciclo de vida, pode ser executada em um contexto diferente, mas que produz um resultado ao final da execução. O tipo de retorno da função `async` é o tipo `Deferred`, e este possui uma função chamada `await()` que retorna o resultado produzido ao final da execução da função `async`.

Async

```
fun main() {  
    GlobalScope.launch {  
        val orders = fetchOrders().await()  
        println(orders)  
    }  
}  
  
suspend fun fetchOrders() = GlobalScope.async {  
    delay(2000) // simulates a external data fetch  
    listOf(  
        Order(  
            id = 1, items = listOf(  
                OrderItem(id = 1, name = "foo", price = 100.0),  
                OrderItem(id = 2, name = "bar", price = 150.0)  
            )  
        )  
    )  
}
```

withContext

Para executar uma coroutine em um contexto específico, como uma operação que faz acesso ao disco ou que realiza uma série de processamentos matemáticos, existem os Dispatchers e a função withContext.

Existem os seguintes Dispatchers:

- **IO:** para operações de entrada / saída de dados
- **Default:** designado para operações que utilizam mais recursos de CPU
- **Main:** utilizado em operações que manipulam componentes de UI
- **Unconfined:** destinado para operações onde não existe a real necessidade de serem executadas em thread específica

coroutinesScope

Existem situações onde se faz necessário a paralelização de operações, como o carregamento dos dados de um usuário do Github e seus respectivos repositórios, e isso pode ser muito fácil de fazer com coroutines:

```
suspend fun loadUserData(username: String): User {  
    return coroutineScope {  
        val user = async { loadUser(username) }  
        val repos = async { loadRepos(username) }  
        buildUserData(user.await(), repos.await())  
    }  
}
```

Suspending Functions

Suspending functions são funções que podem ser pausadas e resumidas durante sua execução sem bloquear a thread, até que sejam finalizadas.

Uma operação bloqueante possui o comportamento de bloquear a thread até que a operação seja finalizada, seja com sucesso ou porque uma exceção foi lançada.

As operações suspensíveis caracterizam execuções que pode ser pausadas e retomadas em outro momento sem bloquear a thread, permitindo multitasking. No Android, operações bloqueantes realizadas na main thread irão congelar a tela até que a operação seja finalizada, já as operações suspensíveis não terão o comportamento de congelar a tela.

Para criar uma suspending function, é necessário apenas adicionar a palavra reservada suspend na declaração da função:

Suspending Functions

Para criar uma suspending function, é necessário apenas adicionar a palavra reservada suspend na declaração da função:

```
suspend fun fazerAlgumaAcaoPorExemploChamadaDeServico() {  
    // do some non blocking operation  
}
```

Funções marcadas com o modificador **suspend** só podem ser chamadas por outras suspending functions ou dentro de uma coroutine.

Kotlin Flow

Como vamos utilizar chamadas assíncronas, ou seja, precisaremos registrar um listener para ler dados ou o resultado de dados gravados. Para isso, utilizaremos o Kotlin Flow.

Kotlin Flow é uma implementação de fluxos reativos feitos em cima de coroutines e canais para Kotlin. Você pode ter usado RxJava / RxKotlin. Os tipos observables e observers em RxJava são um exemplo de estrutura que representa um fluxo de itens. Então Kotlin Coroutines Flow é a alternativa para isso.

A API Flow em Kotlin é a melhor maneira de lidar com o fluxo de dados de forma assíncrona que é executado sequencialmente. Ela só emite valores sempre que houver um receptor para coletá-los.

O construtor `flow {}` é usado para criar um fluxo que pode conter operações assíncronas e pesadas. e o valor não é emitido até que a função terminal `collect` seja chamada.

COROUTINES NO PROJETO



Coroutines

Adicione as libs abaixo para utilizar coroutines no seu app no arquivo **build.gradle (app)**:

```
// Dependências do coroutines
implementation
'org.jetbrains.kotlinx:kotlinx-coroutines-play-services:1.1.1'
implementation
'org.jetbrains.kotlinx:kotlinx-coroutines-android:1.5.2'

// Dependências com viewmodel e livedata
implementation "androidx.lifecycle:lifecycle-livedata-ktx:2.4.1"
implementation
'androidx.lifecycle:lifecycle-viewmodel-ktx:2.4.1'
```

Repositório

Dentro do pacote **repository** crie uma interface chamada **UserRepository**.

```
interface UserRepository {  
  
    suspend fun getUserLogged(): RequestState<User>  
  
}
```

DataSource

Dentro do pacote **data** crie um pacote chamado **remote** e dentro dele adicione o pacote **datasource**. Feito isso, adicione uma classe chamada **UserRemoteDataSource**.

```
interface UserRemoteDataSource {  
  
    suspend fun getUserLogged(): RequestState<User>  
  
}
```



DataSource

Dentro da pasta **datasource** adicione a classe **UserRemoteFakeDataSourceImpl**

```
@ExperimentalCoroutinesApi
class UserRemoteFakeDataSourceImpl : UserRemoteDataSource {

    override suspend fun getUserLogged(): RequestState<User> {
        delay(2000)
        return RequestState.Success(User("Heider"))
    }

}
```

DataSource

Dentro do pacote **data** crie o pacote chamado **repository** e dentro dele crie o **UserRepositoryImpl**:

```
data class UserRepositoryImpl(
    val userRemoteDataSource: UserRemoteDataSource
) : UserRepository {

    override suspend fun getUserLogged(): RequestState<User> {
        return userRemoteDataSource.getUserLogged()
    }
}
```

Criando o modelo usuário

Dentro da pasta **domain/usecases** adicione uma classe chamada **GetUserLoggedUseCase**.

```
class GetUserLoggedUseCase(  
    private val userRepository: UserRepository  
) {  
  
    suspend fun getUserLogged(): RequestState<User> =  
        userRepository.getUserLogged()  
  
}
```

Estrutura da autenticação

Dentro do pacote **base** crie um **package** chamado **auth**. Dentro dele adicione duas novas classes:

BaseAuthFragment.kt

BaseAuthViewModel.kt

BaseViewModelFactory



BaseViewModelFactory

Adicione o código abaixo no arquivo **BaseViewModelFactory**

```
class BaseViewModelFactory(
    private val getUserLoggedUseCase: GetUserLoggedUseCase
) : ViewModelProvider.Factory {

    override fun <T : ViewModel> create(modelClass: Class<T>): T {
        return
    modelClass.getConstructor(GetUserLoggedUseCase::class.java).newInstance(getUserLoggedUseCase)
    }
}
```

BaseAuthViewModel

Adicione o código abaixo no arquivo **BaseAuthViewModel**

```
class BaseAuthViewModel(
    private val getUserLoggedUseCase: GetUserLoggedUseCase
) : ViewModel() {

    var userLogged = MutableLiveData<RequestState<User>>()

    fun getUserLogged() {
        viewModelScope.launch {
            userLogged.value = getUserLoggedUseCase.getUserLogged()
        }
    }
}
```

CRIANDO UM FRAGMENT BASE

Adicione o seguinte código no **BaseAuthFragment.kt**:

```
const val NAVIGATION_KEY = "NAV_KEY"
@ExperimentalCoroutinesApi
abstract class BaseAuthFragment : BaseFragment() {

    private val baseAuthViewModel: BaseAuthViewModel by lazy {
        ViewModelProvider(
            this,
            BaseViewModelFactory(GetUserLoggedUseCase(UserRepositoryImpl(UserRemoteFakeDataSourceImpl())))
        ).get(BaseAuthViewModel::class.java)
    }

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {

        registerObserver()
        baseAuthViewModel.getUserLogged()
        return super.onCreateView(inflater, container, savedInstanceState)
    }
}
```

CRIANDO UM FRAGMENT BASE

Adicione o seguinte código no **BaseAuthFragment.kt**:

```
private fun registerObserver() {  
  
    baseAuthViewModel.userLogged.observe(viewLifecycleOwner, Observer {  
        result ->  
            when (result) {  
                is RequestState.Loading -> {  
                    showLoading()  
                }  
  
                is RequestState.Success -> {  
                    hideLoading()  
                }  
  
                is RequestState.Error -> {  
                    hideLoading()  
                }  
            }  
        })  
    }  
}
```

DEFININDO OS FRAGMENTS

ADICIONANDO OS LAYOUTS AOS FRAGMENTS



BetterFuelFragment

Abra o arquivo **BetterFuelFragment.kt**

```
class BetterFuelFragment : BaseAuthFragment() {  
    override val layout = R.layout.fragment_better_fuel  
}
```

HomeFragment

Abra o arquivo **HomeFragment.kt**

```
class HomeFragment : BaseAuthFragment() {  
    override val layout = R.layout.fragment_home  
}
```

LoginFragment

Abra o arquivo **LoginFragment.kt**

```
class LoginFragment : BaseFragment() {  
    override val layout = R.layout.fragment_login  
}
```

LoginFragment

Abra o arquivo **ProfileFragment.kt**

```
class ProfileFragment : BaseAuthFragment() {  
    override val layout = R.layout.fragment_profile  
}
```

LoginFragment

Abra o arquivo **SignUpFragment.kt**

```
class SignUpFragment : BaseFragment() {  
  
    override val layout = R.layout.fragment_sign_up  
  
}
```

LoginFragment

Abra o arquivo **TermsFragment.kt**

```
class TermsFragment : BaseFragment() {  
    override val layout = R.layout.fragment_terms  
}
```

HORA DA TEORIA: CONHECENDO NAVIGATION COMPONENT - ANDROID JETPACK



O ANDROID JETPACK

É um conjunto de bibliotecas e ferramentas criado com o objetivo de simplificar e melhorar a qualidade do processo de desenvolvimento de aplicativos Android. Tais bibliotecas facilitam o desenvolvimento de apps com qualidade, previsibilidade e simplicidade.

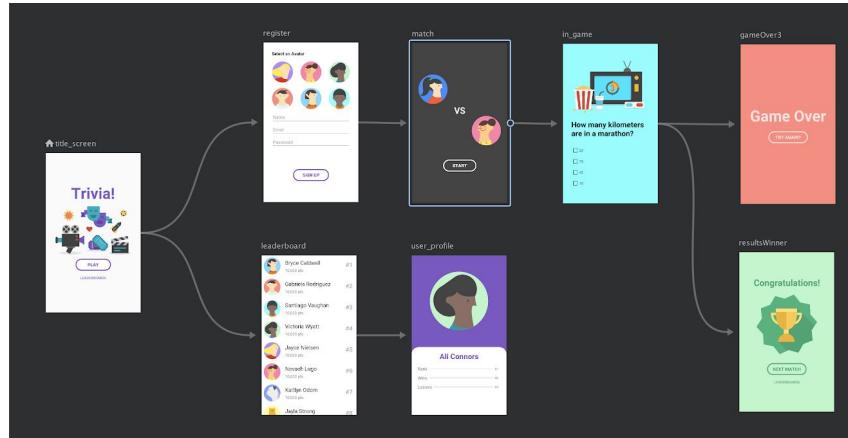


O NAVIGATION

O principal objetivo de utilizar este novo componente é reduzir a probabilidade de erro ao realizar alguma transação com fragments e melhorar a capacidade de testar a UI de forma isolada.

Ao utilizar o Navigation Component você estará delegando ao componente os seguintes conceitos de navegação:

- Automação nas transações de fragments
- Implementação dos princípios de navegação



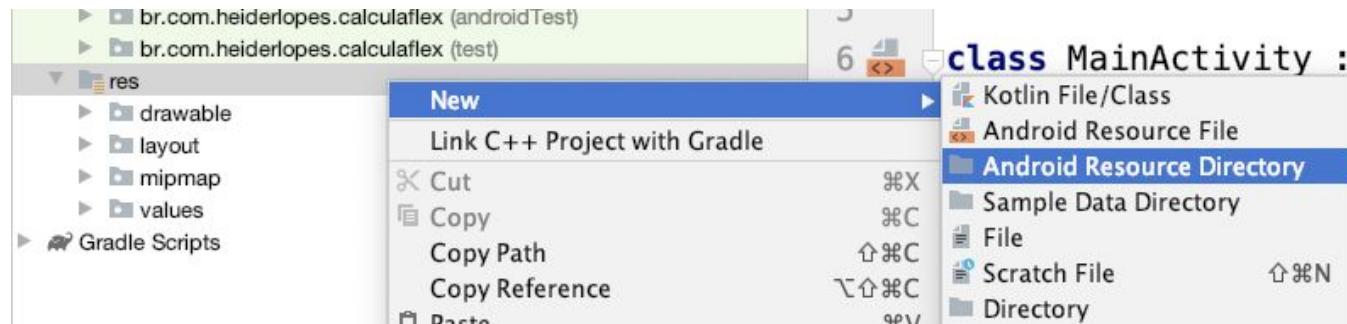
ADICIONANDO A BIBLIOTECA NAVIGATION

Abra o arquivo **build.gradle (app)** e adicione as seguintes dependências:

```
//Dependencias do Navigation  
implementation "androidx.navigation:navigation-fragment-ktx:2.4.1"  
implementation "androidx.navigation:navigation-ui-ktx:2.4.1"
```

ADICIONANDO O NAVIGATION

Clique com o botão direito sobre a pasta **res** ⇒ **New** ⇒ **Android Resource Directory**



Escolha o **Directory name** e o **Resource type** como: **navigation**

Directory name:	navigation
Resource type:	navigation
Source set:	main
Available qualifiers:	Chosen qualifiers:

ADICIONANDO O NAV GRAPH LOGIN

Clique com o botão direito sobre a pasta **navigation** → **New** → **Navigation Resource File**



File name:

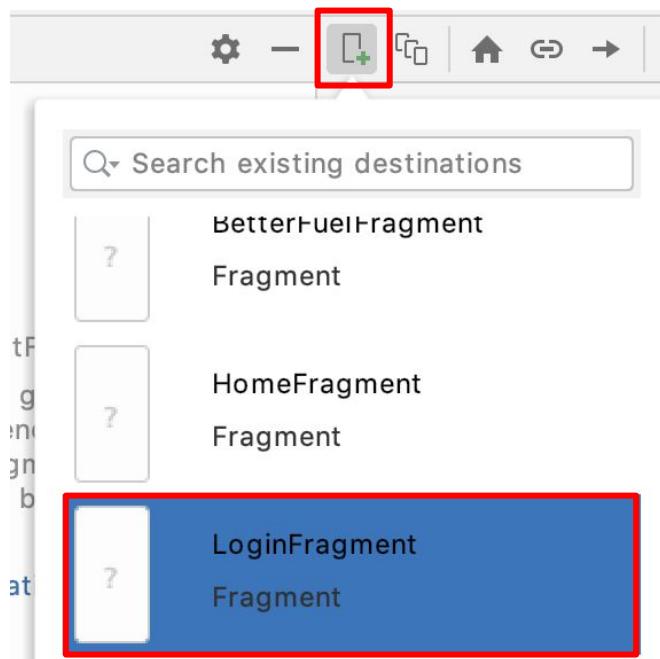
Root element:

Source set: ▾

Directory name:

APLICANDO A AUTENTICAÇÃO

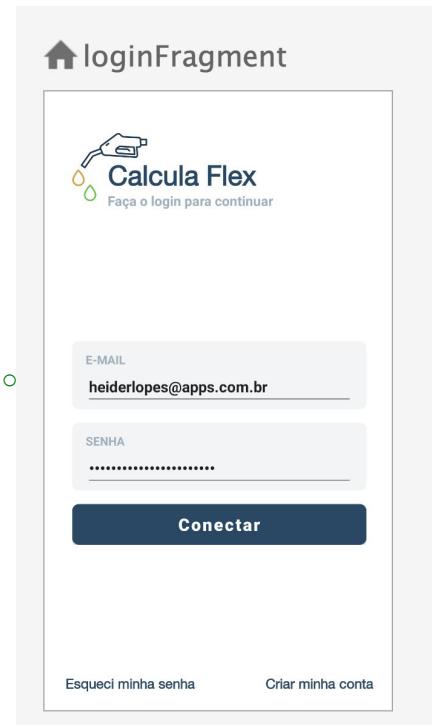
Abra o **login_graph.xml** e adicione o fragment **LoginFragment**.



APLICANDO A AUTENTICAÇÃO

Caso o layout não apareça clique em **Code** ou **Split** e adicione manualmente.

```
<fragment
    xmlns:tools="http://schemas.android.com/tools"
    tools:layout="@layout/fragment_login"
    android:id="@+id/loginFragment"
    android:name="br.com.calculaflex.presentation.lo
    android:label="LoginFragment" />
```



APLICANDO A AUTENTICAÇÃO

Adicione os seguintes fragments no **login_graph.xml**

loginFragment



E-MAIL

SENHA

Conectar

[Esqueci minha senha](#) [Criar minha conta](#)

signUpFragment



SEU NOME

E-MAIL

TELEFONE

SENHA

Li e aceito os termos

Criar conta

[Já tenho conta](#)

termsFragment

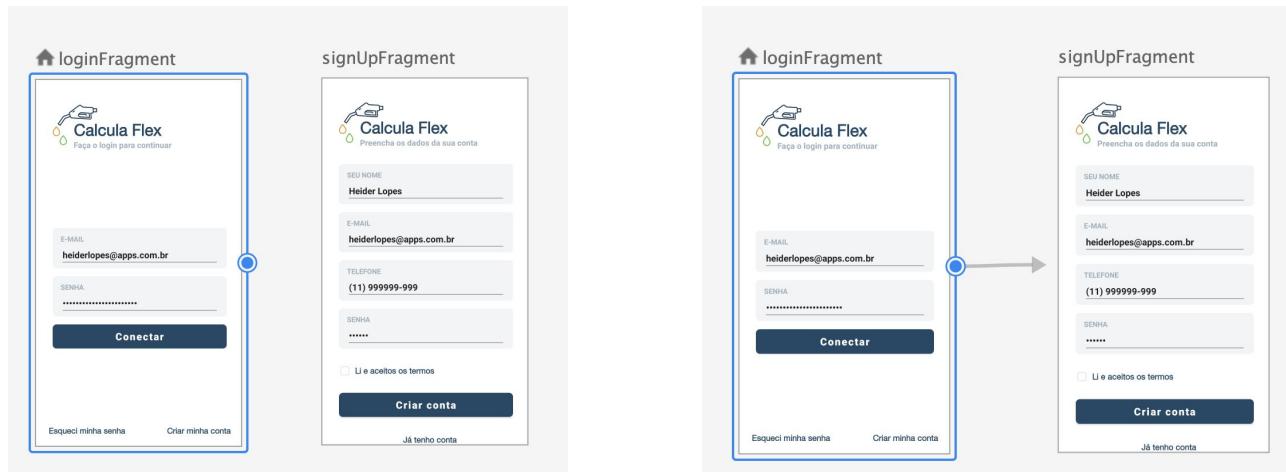


CONFIGURANDO A NAVEGAÇÃO

Para definir o fluxo, criaremos **Actions** que indicarão para qual tela iremos ao chamá-las.

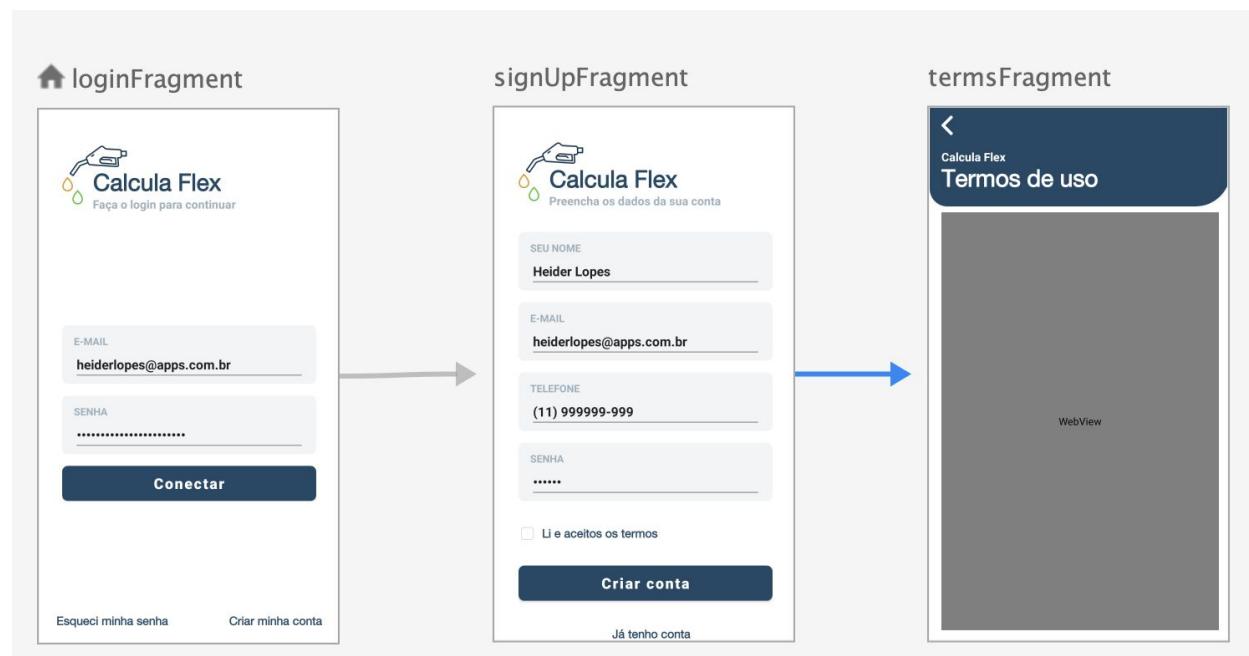
Para isso, clique no indicador que aparece ao lado direito da sua tela de origem e arraste até o destino.

No exemplo abaixo, criamos uma Action da **Tela de Login** para **Tela de SignUp**



CONFIGURANDO A NAVEGAÇÃO

Agora crie uma que vá da tela de signup para a tela de termos de uso. Deverá ficar conforme imagem abaixo:



CONFIGURANDO NOSSO NAV GRAPH SPLASH

Ao clicarmos na nossa tela, conseguimos visualizar quais são as actions que ela possui conforme podemos observar na imagem abaixo:

The diagram illustrates a navigation graph between two fragments: **loginFragment** and **signUpFragment**. A blue arrow points from the **Conectar** button in the **loginFragment** to the **signUpFragment**. On the right, the **Attributes** panel shows the configuration for the **loginFragment**, specifically the **Actions** section, which contains the action **signUpFragment**.

Attributes

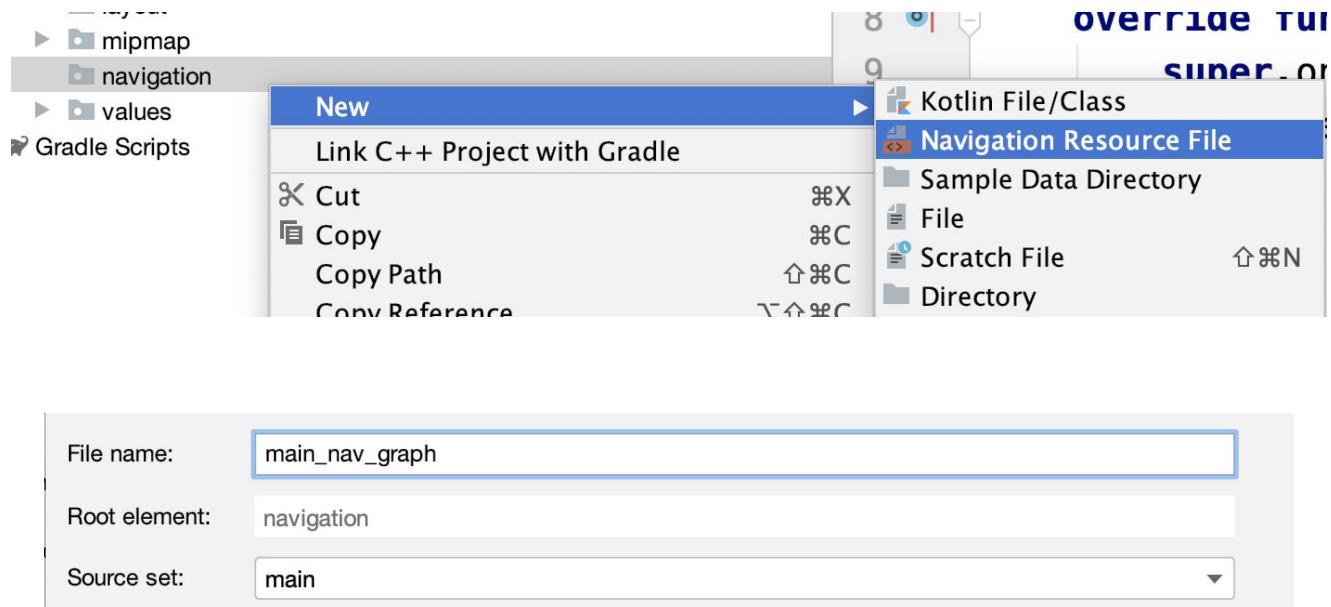
<input type="checkbox"/> loginFragment	fragment
id	<input type="text" value="loginFragment"/>
label	<input type="text" value="LoginFragment"/>
name	<input type="text"/>

Actions

- **signUpFragment** (action_loginFragment_to_signUpFragment)

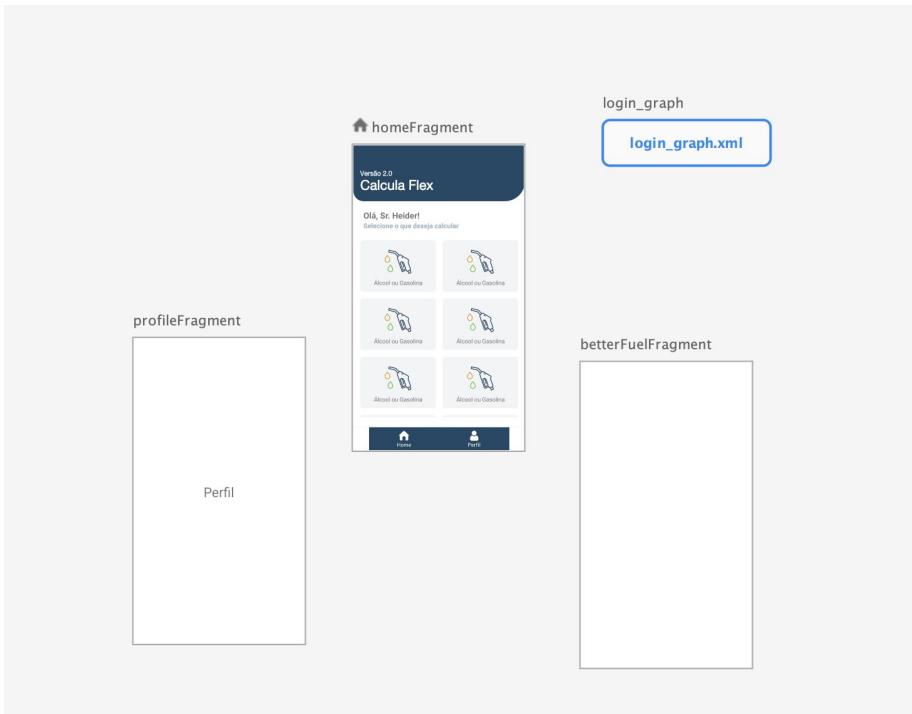
ADICIONANDO O NAV GRAPH PRINCIPAL

Clique com o botão direito sobre a pasta **navigation** → **New** → **Navigation Resource File**



ADICIONANDO O NAV GRAPH PRINCIPAL

No **main_nav_graph.xml** adicione as seguintes telas:



ADICIONANDO O NAV GRAPH PRINCIPAL

Agora abra a **activity_main.xml** e adicione o seguinte código para utilizar o **main_nav_graph** criado.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <androidx.fragment.app.FragmentContainerView
        android:id="@+id/navHostFragment"
        android:name="androidx.navigation.fragment.NavHostFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:defaultNavHost="true"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:navGraph="@navigation/main_nav_graph" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

ADICIONANDO O NAV GRAPH PRINCIPAL

O próximo passo é direcionar o usuário para o **Login** caso ele **não esteja autenticado**. Abra o **BaseAuthFragment** e adicione o seguinte código em negrito no Error.

```
is RequestState.Error -> {
    findNavController(this).navigate(
        R.id.login_graph, bundleOf(
            NAVIGATION_KEY to
        findNavController(this).currentDestination?.id
    )
)
hideLoading()
}
```

* Import necessário: `import androidx.navigation.fragment.NavHostFragment.Companion.findNavController`

Corrigindo o Back Pressed

Abra a **LoginFragment** e adicione o seguinte código:

```
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)
    registerBackPressedAction()
}

private fun registerBackPressedAction () {
    val callback = object : OnBackPressedCallback(true) {
        override fun handleOnBackPressed () {
            activity?.finish()
        }
    }

    requireActivity().onBackPressedDispatcher.addCallback(viewLifecycleOwner
, callback)
}
```

Corrigindo o Back Pressed

Abra a **HomeFragment** e adicione o seguinte código:

```
@ExperimentalCoroutinesApi
class HomeFragment : BaseAuthFragment() {

    override val layout = R.layout.fragment_home

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        registerBackPressedAction()
    }

    private fun registerBackPressedAction() {
        val callback = object : OnBackPressedCallback(true) {
            override fun handleOnBackPressed() {
                activity?.finish()
            }
        }
        requireActivity().onBackPressedDispatcher.addCallback(viewLifecycleOwner,
        callback)
    }
}
```

REALIZANDO O LOGIN



Realizando o login

Dentro do pacote **domain/entity** adicione uma classe chamada **UserLogin** com os seguintes atributos:

```
data class UserLogin(  
    val email: String,  
    val password: String  
)
```

Realizando o login

Agora dentro da interface **UserRepository** adicione o seguinte método:

```
interface UserRepository {  
  
    suspend fun getUserLogged(): RequestState<User>  
  
    suspend fun doLogin(userLogin: UserLogin): RequestState<User>  
  
}
```

Realizando o login

Agora dentro de **UserRemoteDataSource** adicione o seguinte método:

```
interface UserRemoteDataSource {  
  
    suspend fun getUserLogged(): Flow<RequestState<User>>  
  
    suspend fun doLogin(userLogin: UserLogin): RequestState<User>  
  
}
```

Realizando o login

Agora dentro de **UserRemoteFakeDataSourceImpl** adicione o seguinte método:

```
@ExperimentalCoroutinesApi
class UserRemoteFakeDataSourceImpl : UserRemoteDataSource {

    override suspend fun getUserLogged(): RequestState<User> {
        delay(2000)
        return RequestState.Success(User("Heider"))
        //return RequestState.Error(Exception("Usuário expirado"))
    }
}
```

Realizando o login

Agora dentro de **UserRemoteFakeDataSourceImpl** adicione o seguinte método:

```
override suspend fun doLogin(userLogin: UserLogin): RequestState<User> {
    return if(userLogin.email == "heider" && userLogin.password == "123456") {
        RequestState.Success(User("Heider"))
    } else {
        RequestState.Error(Exception("Usuario ou senha inválida"))
    }
}
```

Realizando o login

Agora dentro de **UserRepositoryImpl** adicione o seguinte método:

```
override suspend fun doLogin(userLogin: UserLogin):  
RequestState<User> {  
    return userRemoteDataSource.doLogin(userLogin)  
}
```

Realizando o login

Dentro do pacote **usecases** adicione o **LoginUseCase**

```
class LoginUseCase(  
    private val userRepository: UserRepository  
) {  
  
    suspend fun doLogin(userLogin: UserLogin): RequestState<User> =  
        userRepository.doLogin(userLogin)  
  
}
```

Realizando o login

Dentro do pacote **login** adicione a classe **LoginViewModelFactory** e adicione o seguinte código:

```
class LoginViewModelFactory(
    private val loginUseCase: LoginUseCase
): ViewModelProvider.Factory {

    override fun <T : ViewModel> create(modelClass: Class<T>): T {
        return
modelClass.getConstructor(LoginUseCase::class.java).newInstance(log
inUseCase)
    }
}
```

Realizando o login

Dentro do pacote **login** adicione a classe **LoginViewModel** e adicione o seguinte código:

```
class LoginViewModel(
    private val loginUseCase: LoginUseCase
) : ViewModel() {

    val loginState = MutableLiveData<RequestState<User>>()

    fun doLogin(email: String, password: String) {
        viewModelScope.launch {
            loginState.value = loginUseCase.doLogin(UserLogin(email,
password))
        }
    }
}
```

Realizando o login

Dentro do pacote **login** adicione o código para o **LoginFragment**

```
@ExperimentalCoroutinesApi
class LoginFragment : BaseFragment() {

    override val layout = R.layout.fragment_login

    private lateinit var tvSubTitleSignUp: TextView
    private lateinit var containerLogin: LinearLayout
    private lateinit var tvResetPassword: TextView
    private lateinit var tvNewAccount: TextView

    private lateinit var btLogin: Button
    private lateinit var etEmailLogin: EditText
    private lateinit var etPasswordLogin: EditText
```

Realizando o login

Dentro do pacote **login** adicione o código para o **LoginFragment**

```
private val loginViewModel: LoginViewModel by lazy {
    ViewModelProvider(
        this,
        LoginViewModelFactory(LoginUseCase(UserRepositoryImpl(UserRemoteFakeData
            SourceImpl())))
    ).get(LoginViewModel::class.java)
}

override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)

    setUpView(view)
    registerObserver()
    registerBackPressedAction()
}
```

Realizando o login

Dentro do pacote **login** adicione o código para o **LoginFragment**

```
private fun setUpView(view: View) {  
    tvSubTitleSignUp = view.findViewById(R.id.tvSubTitleLogin)  
    containerLogin = view.findViewById(R.id.containerLogin)  
    tvResetPassword = view.findViewById(R.id.tvResetPassword)  
    tvNewAccount = view.findViewById(R.id.tvNewAccount)  
  
    btLogin = view.findViewById(R.id.btLogin)  
    etEmailLogin = view.findViewById(R.id.etEmailLogin)  
    etPasswordLogin = view.findViewById(R.id.etPasswordLogin)  
  
    btLogin.setOnClickListener {  
        loginViewModel.doLogin(  
            etEmailLogin.text.toString(),  
            etPasswordLogin.text.toString()  
        )  
    }  
}
```

Realizando o login

Dentro do pacote **login** adicione o código para o **LoginFragment**

```
tvResetPassword.setOnClickListener {  
}  
  
tvNewAccount.setOnClickListener {  
  
    findNavController().navigate(R.id.action_loginFragment_to_signUpFragment)  
}  
}
```

Realizando o login

Dentro do pacote **login** adicione o código para o **LoginFragment**

```
private fun registerObserver() {
    loginViewModel.loginState.observe(viewLifecycleOwner, Observer {
        when (it) {
            is RequestState.Success -> showSuccess()
            is RequestState.Error -> showError(it.throwable)
            is RequestState.Loading -> showLoading("Realizando a
autenticação")
        }
    })
}
```

Realizando o login

Dentro do pacote **login** adicione o código para o **LoginFragment**

```
private fun showSuccess() {
    hideLoading()
    val navIdForArguments = arguments?.getInt(NAVIGATION_KEY)
    if (navIdForArguments == null) {
        findNavController().navigate(R.id.main_nav_graph)
    } else {
        findNavController().popBackStack(navIdForArguments, false)
    }
}
```

Realizando o login

Dentro do pacote **login** adicione o código para o **LoginFragment**

```
private fun showError(throwable: Throwable) {
    hideLoading()

    etEmailLogin.error = null
    etPasswordLogin.error = null

    showMessage(throwable.message)
}
```

Realizando o login

Dentro do pacote **login** adicione o código para o **LoginFragment**

```
private fun registerBackPressed() {  
    val callback = object : OnBackPressedCallback(true) {  
        override fun handleOnBackPressed() {  
            activity?.finish()  
        }  
    }  
  
    requireActivity().onBackPressedDispatcher.addCallback(viewLifecycleOwner, callback)  
}  
  
}
```

INTEGRANDO O LOGIN COM O FIREBASE



O FIREBASE

O Firebase é um Baas (Backend as a Service) para aplicações Web e Mobile do Google, foi lançado em 2004 e com o passar dos anos cresceu muito, se tornando uma ferramenta que hoje para alguns projetos é a melhor opção, devido a quantidade de serviços oferecidos por ele, além da facilidade de implementação.

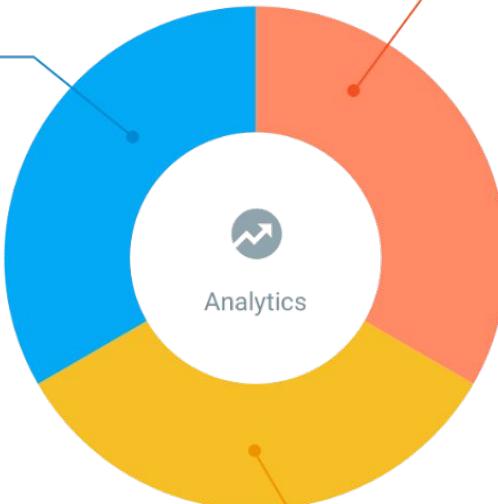


Firebase

O QUE O FIREBASE FORNECE?

DEVELOP

-  Realtime Database
-  Authentication
-  Cloud Messaging
-  Storage
-  Hosting
-  Test Lab
-  Crash Reporting



GROW



Notifications



Remote Config



App Indexing



Dynamic Links



Invites



AdWords



AdMob

O FIREBASE

Crie uma conta através do link:

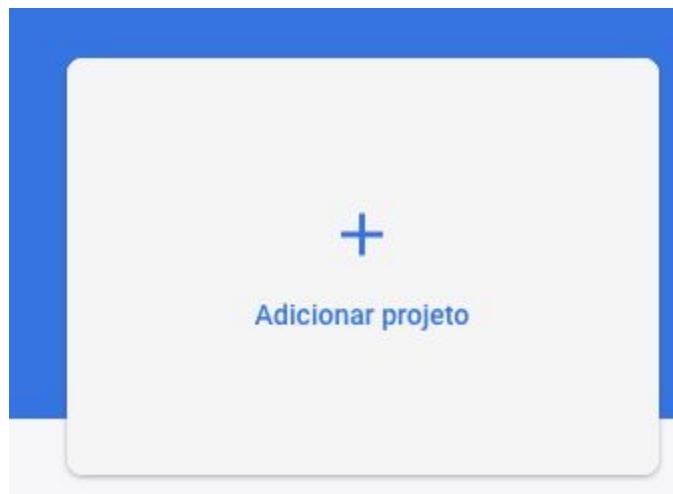
firebase.google.com

Você também pode acessar o console direto:

console.firebaseio.google.com

CRIANDO O PROJETO NO FIREBASE

Crie um novo projeto



CRIANDO O PROJETO NO FIREBASE

Defina o nome do projeto:

X Criar um projeto(Passo 1 de 3)

Vamos começar com um
nome para o projeto[?]

Nome do projeto

Calcula Flex

 calcula-flex-f8d41

Continuar

CRIANDO O PROJETO NO FIREBASE

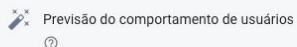
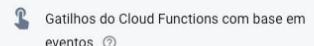
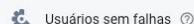
Ateve o Google Analytics no projeto:

X Criar um projeto(Passo 2 de 3)

Google Analytics para seu projeto do Firebase

O Google Analytics é uma solução de análise gratuita e ilimitada. Com ele, é possível segmentar, gerar relatórios e muito mais nos seguintes produtos: Firebase Crashlytics, Cloud Messaging, Mensagens no app, Configuração remota, Teste A/B, Previsões e Cloud Functions.

O Google Analytics ativa:



Ativar o Google Analytics neste projeto
Recomendado

Anterior

Continuar

CRIANDO O PROJETO NO FIREBASE

Defina conta utilizada para o Google Analytics e clique em **Criar projeto**

The screenshot shows the third step of creating a Firebase project, titled 'Configurar o Google Analytics'. It asks the user to choose or create a Google Analytics account. A dropdown menu is open, showing 'heidertreinamentos' as the selected option. Below the dropdown, there's a link to 'Criar automaticamente uma nova propriedade nesta conta'. At the bottom, there are two buttons: 'Anterior' on the left and 'Criar projeto' on the right, which is highlighted with a blue background.

X Criar um projeto(Passo 3 de 3)

Configurar o Google Analytics

Escolha ou crie uma conta do Google Analytics [?](#)

heidertreinamentos [▼](#)

Criar automaticamente uma nova propriedade nesta conta [editar](#)

Após a criação do projeto, uma nova propriedade do Google Analytics será criada na conta selecionada e vinculada ao seu projeto no Firebase. Esse processo permitirá o fluxo de dados entre os produtos. Os dados da propriedade do Google Analytics exportados para o Firebase ficam sujeitos aos Termos de Serviço do Firebase, e os dados do Firebase importados para o Google Analytics ficam sujeitos aos Termos de Serviço do Analytics. [Saiba mais](#)

[Anterior](#) [Criar projeto](#)

CRIANDO O PROJETO NO FIREBASE

Após a criação do projeto, clique em **Continuar**

The screenshot shows the Firebase console interface. At the top, there's a navigation bar with icons for Home, Projects, Functions, Hosting, Storage, Database, Firestore, and Authentication. Below the navigation bar, a large header displays the text "Criando o projeto no Firebase". Underneath the header, there's a section titled "Nome do projeto" with a text input field containing "Calcula Flex". To the right of the input field is a "Criar projeto" button with a checkmark icon. Below this, there's a "Continuar" button. A success message "Seu novo projeto está pronto" is displayed with a green checkmark icon. The background of the page is white.

Habilitando a autenticação



HABILITANDO A AUTENTICAÇÃO

Clique em **Authentication**, em seguida, **Configurar método de login**

The screenshot shows the Firebase console interface. On the left, there's a sidebar with navigation links: Desenvolver, Authentication (which is highlighted with a red box), Database, Storage, Hosting, Functions, and ML Kit. The main area is titled "Authentication" and has tabs for Users, Sign-in method (which is selected and underlined), Templates, and Usage. Below the tabs, there's a section titled "Provedores de login" (Login providers) with a sub-section for "Provedores nativos" (Native providers). Under "Provedores nativos", the "E-mail/senha" option is highlighted with a red box. To its right, under "Outros provedores", are options for Google, Facebook, Play Games, Smartphone, Game Center, Apple, GitHub, Microsoft, Twitter, and Yahoo.

HABILITANDO A AUTENTICAÇÃO

Ateve esse modo de autenticação e clique em **Salvar**

Provedores de login

 E-mail/senha

 Ativar

Permite que os usuários se inscrevam usando o endereço de e-mail e a senha deles. Nossos SDKs também fornecem verificação de endereço de e-mail, recuperação de senha e componentes essenciais para alteração do endereço de e-mail. [Saiba mais ↗](#)

Link do e-mail (login sem senha)

 Ativar

[Cancelar](#)

Salvar

CRIANDO UM USUÁRIO

Adicione um usuário para fazermos o teste. Clique na aba **Users**

Authentication

Users Sign-in method Templates Usage

Pesquise por endereço de e-mail, número de telefone ou UID do usuário Adicionar usuário ⋮

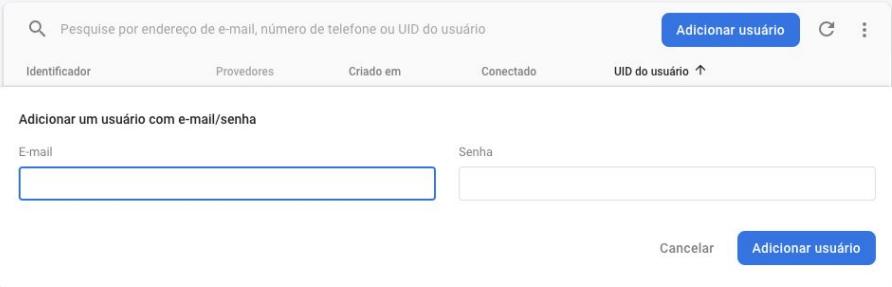
Identificador	Provedores	Criado em	Conectado	UID do usuário ↑
---------------	------------	-----------	-----------	------------------

Adicionar um usuário com e-mail/senha

E-mail Senha

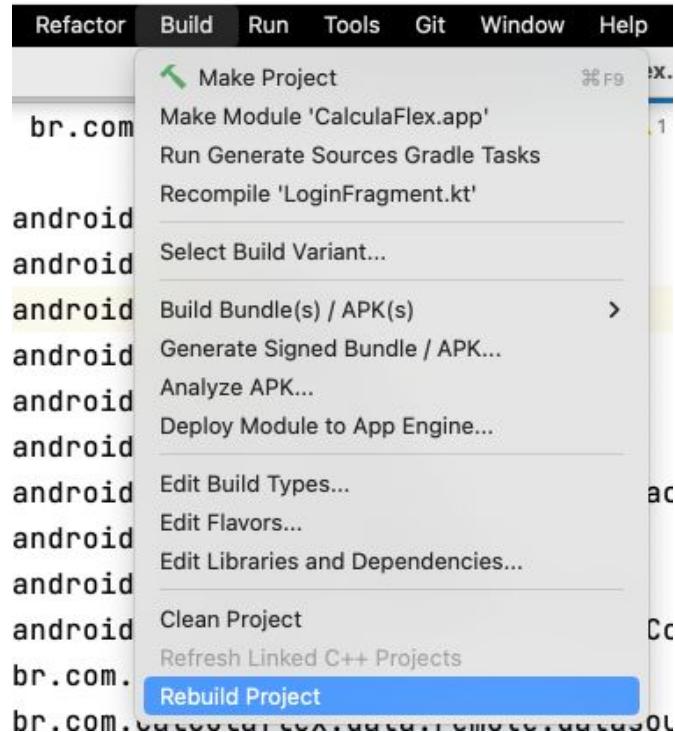
Cancelar Adicionar usuário

Nenhum usuário para este projeto ainda



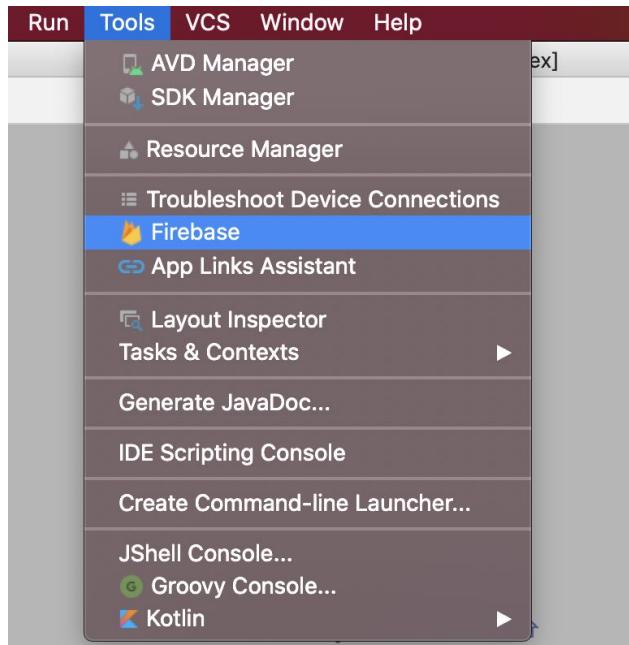
INTEGRANDO O PROJETO COM O FIREBASE

Clique em Build -> Rebuild Project



INTEGRANDO O PROJETO COM O FIREBASE

Agora no **Android Studio** clique em **Tools → Firebase**



INTEGRANDO O PROJETO COM O FIREBASE

Agora no **Android Studio** clique em **Tools → Firebase**

1 Authentication

Sign in and manage users with ease, accepting emails, Google Sign-in, Facebook and other login providers. [More info](#)

2 Email and password authentication

The screenshot shows the Firebase Authentication interface. At the top, there's a navigation bar with a back arrow, the word 'Firebase', and the word 'Authentication'. Below the navigation bar, the title 'Email and password authentication' is displayed. A descriptive text explains that users can sign in with their email addresses and passwords, and manage their accounts. It includes a link 'Launch in browser'. Below this, there are two numbered steps: step 1, 'Connect your app to Firebase', which has a 'Connect to Firebase' button; and step 2, 'Add Firebase Authentication to your app', which has a 'Add Firebase Authentication to your app' button.

← Firebase ➔ Authentication

Email and password authentication

You can use Firebase Authentication to let your users sign in with their email addresses and passwords, and to manage your app's password-based accounts. This tutorial helps you set up an email and password system and then access information about the user.

[Launch in browser](#)

① 3 Connect your app to Firebase

② Add Firebase Authentication to your app

INTEGRANDO O PROJETO COM O FIREBASE

Escolha o projeto criado anteriormente:

The screenshot shows the Firebase console interface. At the top, a blue header bar displays the text "Escolha um projeto para prosseguir". Below this, a section titled "Seus projetos do Firebase" lists available projects. On the left, there is a button with a plus sign and the text "Adicionar projeto". To the right, a project named "Calcula Flex" is listed, showing its project ID: "calcula-flex-f8d41".

Escolha um projeto para prosseguir

Seus projetos do Firebase

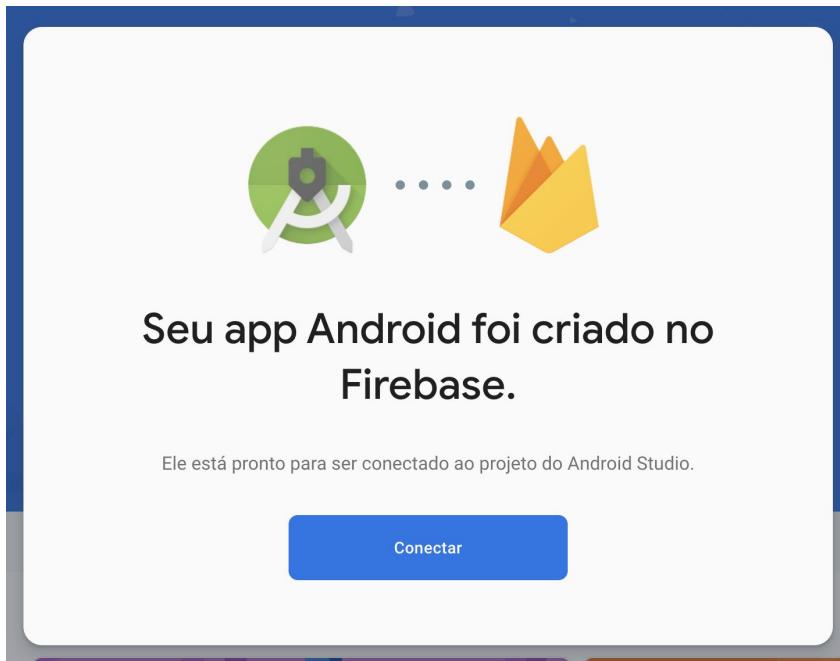
+

Adicionar projeto

Calcula Flex
calcula-flex-f8d41

INTEGRANDO O PROJETO COM O FIREBASE

Estando tudo certo você receberá esta mensagem. Clique em **Conectar**



INTEGRANDO O PROJETO COM O FIREBASE

Tudo certo. Volte ao **Android Studio**.



**Your Android Studio project is connected
to your Firebase Android app**

You can now use Firebase in your project! Go back to Android Studio to start using one of the
Firebase SDKs.

INTEGRANDO O PROJETO COM O FIREBASE

Observe agora que seu projeto está conectado.

Email and password authentication

You can use Firebase Authentication to let your users sign in with their email addresses to manage your app's password-based accounts. This tutorial helps you set up an email system and then access information about the user.

[Launch in browser](#)

① Connect your app to Firebase

✓ Connected

② Add Firebase Authentication to your app

[Add Firebase Authentication to your app](#)

To use an authentication provider, you need to enable it in the [Firebase console](#). Go to the Method page in the Firebase Authentication section to enable Email/Password sign-in and other identity providers you want for your app.

③ Check current auth state

Declare an instance of `FirebaseAuth`

```
private FirebaseAuth mAuth;
```

In the `onCreate()` method, initialize the `FirebaseAuth` instance.

```
mAuth = FirebaseAuth.getInstance();
```

When initializing your Activity, check to see if the user is currently signed in.

INTEGRANDO O PROJETO COM O FIREBASE

Adicione a dependência no **build.gradle (app)** e atualize a dependência.

De:

implementation 'com.google.firebaseio:firebase-auth:19.2.0'

Para:

```
//Firebase
implementation platform( 'com.google.firebaseio:firebase-bom:29.1.0' )
implementation 'com.google.firebaseio:firebase-analytics-ktx'
implementation 'com.google.firebaseio:firebase-auth-ktx'
```

E sincronize o projeto:



LoginViewModel

Agora dentro de **datasource** crie uma nova classe para implementarmos o Firebase chamada **UserRemoteFirebaseDataSourceImpl**

```
@ExperimentalCoroutinesApi
class UserRemoteFirebaseDataSourceImpl(
    private val firebaseAuth: FirebaseAuth
) : UserRemoteDataSource {

    override suspend fun getUserLogged(): RequestState<User> {
        FirebaseAuth.getInstance().currentUser?.reload()
        val firebaseUser = firebaseAuth.currentUser
        return if(firebaseUser == null) {
            RequestState.Error(Exception("Usuário deslogado"))
        } else {
            RequestState.Success(User(firebaseUser.displayName?: "Não identificado"))
        }
    }
}
```

LoginViewModel

Agora dentro de **datasource** crie uma nova classe para implementarmos o Firebase chamada **UserRemoteFirebaseDataSourceImpl**

```
override suspend fun doLogin(userLogin: UserLogin): RequestState<User> {
    return try {
        firebaseAuth.signInWithEmailAndPassword(userLogin.email,
        userLogin.password).await()
        val firebaseUser = firebaseAuth.currentUser
        if (firebaseUser?.email != null) {
            RequestState.Success(User(firebaseUser.displayName ?: "Não
            identificado"))
        } else {
            RequestState.Error(Exception("Usuario ou senha inválida"))
        }
    } catch (exception: java.lang.Exception) {
        RequestState.Error(exception)
    }
}
```

LoginViewModel

Agora na **LoginFragment** realize a implementação do **UserRemoteFirebaseDataSourceImpl**

```
private val loginViewModel: LoginViewModel by lazy {
    ViewModelProvider(
        this,
        LoginViewModelFactory(LoginUseCase(UserRepositoryImpl(
            UserRemoteFirebaseDataSourceImpl(FirebaseAuth.getInstance())
        )))
    ).get(LoginViewModel::class.java)
}
```

LoginViewModel

Agora na **BaseAuthFragment** realize a implementação do **UserRemoteFirebaseDataSourceImpl**

```
private val baseAuthViewModel: BaseAuthViewModel by lazy {
    ViewModelProvider(
        this,
        BaseViewModelFactory(GetUserLoggedUseCase(UserRepositoryImpl(
            UserRemoteFirebaseDataSourceImpl(
                FirebaseAuth.getInstance()
            )))
    ).get(BaseAuthViewModel::class.java)
}
```

Rode o aplicativo e veja o resultado



Reenviando senha



Reenvio de senha

Apague o arquivo **UserRemoteFakeDataSourceImpl**

Reenvio de senha

Abra o arquivo **UserRepository** e adicione a seguinte função

```
suspend fun resetPassword(email: String): RequestState<String>
```

Reenvio de senha

Abra o arquivo **UserRemoteDataSource** e adicione a seguinte função

```
suspend fun resetPassword(email: String): RequestState<String>
```

Reenvio de senha

Abra o arquivo **UserRemoteFirebaseDataSourceImpl** e adicione a seguinte função

```
override suspend fun resetPassword(email: String):  
RequestState<String> {  
    return try{  
        firebaseAuth.sendPasswordResetEmail(email).await()  
        RequestState.Success("Verifique sua caixa de e-mail")  
    } catch (e: java.lang.Exception) {  
        RequestState.Error(e)  
    }  
}
```

Reenvio de senha

Abra o arquivo **UserRepositoryImpl** e adicione a seguinte função

```
override suspend fun resetPassword(email: String):  
RequestState<String> {  
    return userRemoteDataSource.resetPassword(email)  
}
```

Reenvio de senha

Dentro do pacote **usecases** crie o **ResetPasswordUseCase**:

```
class ResetPasswordUseCase(  
    private val userRepository: UserRepository  
) {  
  
    suspend fun resetPassword(email: String): RequestState<String> =  
        userRepository.resetPassword(email)  
}
```

Reenvio de senha

Abra o arquivo **LoginViewModelFactory** e adicione o caso de uso criado:

```
class LoginViewModelFactory (
    private val loginUseCase: LoginUseCase,
    private val resetPasswordUseCase: ResetPasswordUseCase
) : ViewModelProvider.Factory {

    override fun <T : ViewModel> create(modelClass: Class<T>): T {
        return modelClass.getConstructor(
            LoginUseCase::class.java,
            ResetPasswordUseCase::class.java
        ).newInstance(loginUseCase, resetPasswordUseCase)
    }
}
```

Reenvio de senha

Abra o arquivo **LoginFragment** e adicione o código em negrito:

```
private val loginViewModel: LoginViewModel by lazy {
    ViewModelProvider(
        this,
        LoginViewModelFactory(
            LoginUseCase(
                UserRepositoryImpl(
                    UserRemoteFirebaseDataSourceImpl(FirebaseAuth.getInstance())
                )
            ),
            ResetPasswordUseCase(
                UserRepositoryImpl(
                    UserRemoteFirebaseDataSourceImpl(FirebaseAuth.getInstance())
                )
            )
        )
    ).get(LoginViewModel::class.java)
}
```

Reenvio de senha

Abra o arquivo **LoginViewModel** e adicione a seguinte função:

```
class LoginViewModel (
    private val loginUseCase: LoginUseCase,
    private val resetPasswordUseCase: ResetPasswordUseCase
) : ViewModel() {

    val resetPasswordState = MutableLiveData<RequestState<String>>()

    fun resetPassword(email: String) {
        viewModelScope.launch {
            resetPasswordState.value =
                resetPasswordUseCase.resetPassword(email)
        }
    }
}
```

Reenvio de senha

Abra o **LoginFragment** novamente e adicione o seguinte código:

```
tvResetPassword.setOnClickListener {
    if(etEmailLogin.text.toString().isEmpty()) {
        showMessage("Informe o e-mail que deseja alterar a senha")
    } else {
        loginViewModel.resetPassword(etEmailLogin.text.toString())
    }
}
```

Reenvio de senha

Abra o **LoginFragment** novamente e adicione o seguinte código:

```
private fun registerObserver () {
    //Restante dos observers

    loginViewModel.resetPasswordState.observe(viewLifecycleOwner, Observer
    {
        when (it) {
            is RequestState.Success -> {
                hideLoading()
                showMessage(it.data)
            }
            is RequestState.Error -> showError(it.throwable)
            is RequestState.Loading -> showLoading("Reenviando o e-mail
para alteração")
        }
    })
}
```

Reenvio de senha

Vá até o console do **Firebase em Authentication** → **Templates** → **Redefinição de senha** e altere o e-mail da forma que desejar, mantendo o link para alteração da senha:

Nome do remetente De
noreply noreply @calcula-flex-19mob.firebaseio.com
[personalizar domínio](#)

Responder para
endereço de e-mail de resposta

Assunto
Alteração de senha Calcula Flex

Mensagem ⓘ

```
<p>Olá,</p>
<p>Segue o link para reiniciar sua senha.</p>
<p><a href=%LINK%>%LINK%</a></p>
<p>Se você não solicitou ignore este e-mail</p>
<p>Obrigado,</p>
<p>Equipe Calcula Flex</p>
```

URL açãoável (valor %LINK%) ⓘ
https://calcula-flex-19mob.firebaseioapp.com/_/auth/action
[personalizar URL açãoável](#)

[Cancelar](#) [Salvar](#)

Criando uma conta



Reenvio de senha

Antes de criar a conta iremos realizar a implementação do contrato. Para chegar nesta tela abra o **SignUpFragment** e adicione o seguinte código:

```
class SignUpFragment : BaseFragment() {  
  
    override val layout = R.layout.fragment_sign_up  
  
    private lateinit var tvTerms: TextView  
  
    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {  
        super.onViewCreated(view, savedInstanceState)  
  
        tvTerms = view.findViewById(R.id.tvTerms)  
  
        tvTerms.setOnClickListener {  
            NavHostFragment.findNavController(this)  
                .navigate(R.id.action_signUpFragment_to_termsFragment)  
        }  
    }  
}
```

MELHORANDO A NAVEGAÇÃO

Crie um arquivo chamado **slide_in_right.xml** dentro de **res/anim** e adicione o seguinte código:

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">

    <translate android:fromXDelta="100%" android:toXDelta="0%"
        android:fromYDelta="0%" android:toYDelta="0%"
        android:duration="700"/>
</set>
```

MELHORANDO A NAVEGAÇÃO

Crie um arquivo chamado **slide_out_left.xml** dentro de **anim** e adicione o seguinte código:

```
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <translate android:fromXDelta="0" android:toXDelta="-50%p"
              android:duration="@android:integer/config_mediumAnimTime"/>
    <alpha android:fromAlpha="1.0" android:toAlpha="0.0"
           android:duration="@android:integer/config_mediumAnimTime" />
</set>
```

MELHORANDO A NAVEGAÇÃO

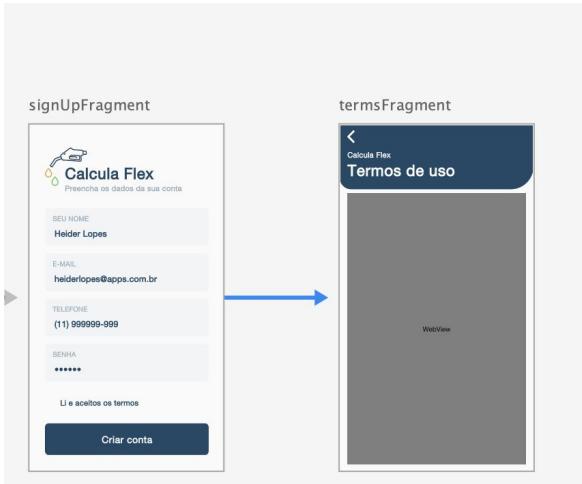
Abra o **login_graph** clique sobre a **Action** que nos leva a fluxo de **SignUp** e configure a animação:

The screenshot shows the Android Studio navigation graph editor. On the left, there are two screens: 'loginFragment' and 'signUpFragment'. A blue arrow points from the bottom right corner of the 'loginFragment' screen towards the top left corner of the 'signUpFragment' screen, indicating the direction of the transition. On the right, the details of this transition are shown in a card:

- id:** action_loginFragment_to_signUpFragment
- destination:** signUpFragment
- Animations:** This section is highlighted with a red border.
 - enterAnim:** @anim/slide_in_right
 - exitAnim:** @anim/slide_out_left
 - popEnterAnim:** @android:anim/slide_in_left
 - popExitAnim:** @android:anim/slide_out_right
- Argument Default Values:** Nothing to show
- Pop Behavior:**
 - popUpTo:** none
 - popUpToInclusive:** -
- Launch Options:** None

MELHORANDO A NAVEGAÇÃO

Aplique a mesma animação para o fluxo entre à tela de Criação de Conta e os Termos de Uso



The screenshot shows the Android navigation graph interface. On the left, there is a diagram of two fragments: 'signUpFragment' on the left and 'termsFragment' on the right. An arrow points from the signUpFragment to the termsFragment, indicating the flow between them. To the right of the diagram is a detailed configuration panel for this transition.

id	action_signUpFragment_to_termsFragment
destination	termsFragment
Animations	
enterAnim	@anim/slide_in_right
exitAnim	@anim/slide_out_left
popEnterAnim	@android:anim/slide_in_left
popExitAnim	@android:anim/slide_out_right
Argument Default Values	
Nothing to show	
Pop Behavior	
popUpTo	none
popUpToInclusive	—
Launch Options	

CONHEÇA MAIS ALGUMAS ANIMAÇÕES
QUE PODEM SER UTILIZADAS NO SEU APP



blink

```
<?xml version="1.0" encoding="utf-8"?>
<set
    xmlns:android="http://schemas.android.com/apk/res/android">
    <alpha
        android:fromAlpha="0.0"
        android:toAlpha="1.0"
        android:interpolator="@android:anim/accelerate_interpolator"
        android:duration="600"
        android:repeatMode="reverse"/>
    <!--android:repeatCount="infinite"-->
</set>
```

bounce

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:fillAfter="true"
    android:interpolator="@android:anim/bounce_interpolator">
    <scale
        android:duration="500"
        android:fromXScale="1.0"
        android:fromYScale="0.0"
        android:toXScale="1.0"
        android:toYScale="1.0"
    />
</set>
```

fade in

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:fillAfter="true">
    <alpha
        android:duration="1000"
        android:fromAlpha="0.0"
        android:interpolator="@android:anim/accelerate_interpolator"
        android:toAlpha="1.0"
    />
</set>
```

fade out

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:fillAfter="true"
    >
    <alpha
        android:duration="1000"
        android:fromAlpha="1.0"
        android:interpolator="@android:anim/accelerate_interpolator"
        android:toAlpha="0.0"
    />
</set>
```

translate

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:interpolator="@android:anim/linear_interpolator"
    android:fillAfter="true">
    <translate
        android:fromXDelta="0%p"
        android:toXDelta="75%p"
        android:duration="800"
    />
</set>
```

rotate

```
<?xml version="1.0" encoding="utf-8"?>
<set
    xmlns:android="http://schemas.android.com/apk/res/android">
    <rotate
        android:fromDegrees="0"
        android:toDegrees="360"
        android:pivotX="50%"
        android:pivotY="50%"
        android:duration="600"
        android:repeatMode="restart"
        android:repeatCount="infinite"
        android:interpolator="@android:anim/cycle_interpolator"/>
</set>
```

slide down

```
<?xml version="1.0" encoding="utf-8"?>
<set
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:fillAfter="true">
    <scale
        android:duration="500"
        android:fromXScale="1.0"
        android:fromYScale="0.0"
        android:interpolator="@android:anim/linear_interpolator"
        android:toXScale="1.0"
        android:toYScale="1.0"
    />
</set>
```

slide up

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:fillAfter="true">
    <scale
        android:duration="500"
        android:fromXScale="1.0"
        android:fromYScale="1.0"
        android:interpolator="@android:anim/linear_interpolator"
        android:toXScale="1.0"
        android:toYScale="0.0"
    />
</set>
```

zoom in

```
<?xm version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:fillAfter="true">
    <scale xmlns:android="http://schemas.android.com/apk/res/android"
        android:duration="1000"
        android:fromXScale="1"
        android:fromYScale="1"
        android:pivotX="50%"
        android:pivotY="50%"
        android:toXScale="3"
        android:toYScale="3"
    >
</scale>
</set>
```

zoom out

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
      android:fillAfter="true">
    <scale
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:duration="1000"
        android:fromXScale="1.0"
        android:fromYScale="1.0"
        android:pivotX="50%"
        android:pivotY="50%"
        android:toXScale="0.5"
        android:toYScale="0.5"
    >
    </scale>
</set>
```

sequencial - 1/3

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:fillAfter="true"
    android:interpolator="@android:anim/linear_interpolator">
    <!-- Use startOffset to give delay between animations -->
    <!-- Move -->
    <translate
        android:duration="800"
        android:fillAfter="true"
        android:fromXDelta="0%p"
        android:startOffset="300"
        android:toXDelta="75%p"
    />
    <translate
        android:duration="800"
        android:fillAfter="true"
        android:fromYDelta="0%p"
        android:startOffset="1100"
        android:toYDelta="70%p"
    />
```

sequencial - 2/3

```
<translate
    android:duration="800"
    android:fillAfter="true"
    android:fromXDelta="0%p"
    android:startOffset="1900"
    android:toXDelta="-75%p"
/>
<translate
    android:duration="800"
    android:fillAfter="true"
    android:fromYDelta="0%p"
    android:startOffset="2700"
    android:toYDelta="-70%p"
/>

```

sequencial - 3/3

```
<rotate
    android:duration="1000"
    android:fromDegrees="0"
    android:interpolator="@android:anim/cycle_interpolator"
    android:pivotX="50%"
    android:pivotY="50%"
    android:startOffset="3800"
    android:repeatCount="infinite"
    android:repeatMode="restart"
    android:toDegrees="360"
/>
</set>
```

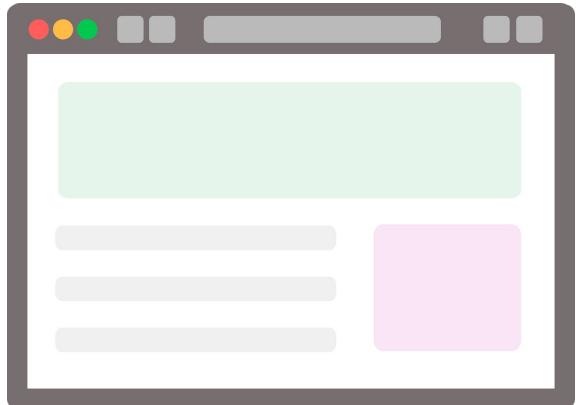
CONHECENDO O HOSTING



O QUE É O HOSTING DO FIREBASE

O **hosting** é indicado para sites estáticos ou micro-serviços, ou seja, podemos adicionar sites construídos em **HTML, CSS, JavaScript** que não precisam de mudança de conteúdo como sistema de chat ou formulário.

Um site estático destina-se a funcionalidades específicas



O FIREBASE-CLI

Para hospedar seu site com o Firebase Hosting, é necessário usar a **Firebase CLI** (uma ferramenta de linha de comando).

Execute o seguinte comando do npm para instalar a CLI ou atualizar para a versão mais recente da CLI.

Firebase CLI depende do **Nodejs** e o **NPM**, se você não tiver pode instalar a versão mais recente.

Após instalados Nodejs e npm, instale a Firebase CLI por meio do NPM executando o código a seguir:

npm install -g firebase-tools

Observação: se o comando falhar, talvez seja necessário consultar a referência da Firebase CLI ou altere suas permissões de npm.

O FIREBASE-CLI

No terminal de comando realize o login no firebase

firebase login

Crie um diretório chamado

mkdir calcula_flex_website

Acesse o diretório criado

cd calcula_flex_website

E vamos inicializar o projeto:

firebase init

O FIREBASE-CLI

Selecione o Hosting e aperte a barra de espaço para selecionar.

```
> firebase init

#####
# # # #
#####
# # # #
#####
# # # #
#####
# # # #

You're about to initialize a Firebase project in this directory:

/Users/heiderlopes/Documents/projetos/fiap/calcula	flex_website

? Which Firebase features do you want to set up for this directory? Press Space to select features, then Enter to confirm your choices. (Press <space> to select, <a> to toggle all, <i> to invert selection, and <enter> to proceed)
o Realtime Database: Configure a security rules file for Realtime Database and (optionally) provision default instance
o Firestore: Configure security rules and indexes files for Firestore
o Functions: Configure a Cloud Functions directory and its files
>o Hosting: Configure files for Firebase Hosting and (optionally) set up GitHub Action deploys
o Hosting: Set up GitHub Action deploys
o Storage: Configure a security rules file for Cloud Storage
o Emulators: Set up local emulators for Firebase products
(Move up and down to reveal more choices)
```

O FIREBASE-CLI

E escolha o projeto que criamos anteriormente:

```
? Please select an option: (Use arrow keys)
❯ Use an existing project
  Create a new project
  Add Firebase to an existing Google Cloud Platform project
  Don't set up a default project
```

Responda as perguntas seguintes e teremos o projeto criado:

```
? Select a default Firebase project for this directory: (Use arrow keys)
❯ calcula-flex-mob (Calcula Flex MOB)
  calculadora-flex-ead (Calculadora Flex EAD)
  fir-mvvm-37124 (FirebaseMVVM)
```

O FIREBASE-CLI

Abra o diretório no seu editor. Por exemplo: **Visual Studio Code**

Crie sua página de termos de uso dentro da pasta **public**:

```
<!DOCTYPE html>

<html>
  <head>
    <title>Calcula Flex - Termos de Uso</title>
  </head>
  <body>
    Termos de uso do aplicativo
  </body>
</html>
```

O FIREBASE-CLI

Após criar a página volte ao terminal e realize o deploy:

firebase deploy

Ou execute

firebase deploy --only hosting

```
Project Console: https://console.firebaseio.google.com/project/calcular-flex-f8d41/overview
Hosting URL: https://calcular-flex-f8d41.web.app

android/calculadora_flex_site/public took 12s
→ └─
```

Copie a url gerada acima em **Hosting URL**

CARREGANDO OS TERMOS

Abra o arquivo **TermsFragment.kt** e adicione o seguinte código:

```
class TermsFragment : BaseFragment() {
    override val layout = R.layout.fragment_terms

    private lateinit var wvTerms: WebView
    private lateinit var ivBack: ImageView
    override fun onViewCreated(view: View, savedInstanceState:
    Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        wvTerms = view.findViewById(R.id.wvTerms)
        ivBack = view.findViewById(R.id.ivBack)

        ivBack.setOnClickListener {
            activity?.onBackPressed()
        }

        wvTerms.loadUrl("https://calcula-flex-f8d41.web.app")
    }
}
```

EXERCÍCIO:

IMPLEMENTE A AÇÃO PARA CRIAR UMA CONTA
NO FIREBASE AUTH E IR PARA HOME



CRIANDO UMA NOVA CONTA



Criação de conta

Crie um arquivo chamado **NewUser** dentro de **entity** e adicione os seguintes campos:

```
data class NewUser(  
    val name: String,  
    val email: String,  
    val phone: String,  
    val password: String  
)
```

Criação de conta

Abra o arquivo **UserRepository** e adicione a seguinte função

```
suspend fun create(newUser: NewUser): RequestState<User>
```

Criação de conta

Abra o arquivo **UserRemoteDataSource** e adicione a seguinte função

```
suspend fun create(newUser: NewUser): RequestState<User>
```

Criação de conta

Abra o arquivo **UserRemoteFirebaseDataSourceImpl** e adicione a seguinte função

```
override suspend fun create(newUser: NewUser): RequestState<User> {
    return try{
        firebaseAuth.createUserWithEmailAndPassword(newUser.email,
newUser.password).await()
        RequestState.Success(User(newUser.name))
    } catch (e: java.lang.Exception) {
        RequestState.Error(e)
    }
}
```

Criação de conta

Abra o arquivo **UserRepositoryImpl** e adicione a seguinte função

```
override suspend fun create(newUser: NewUser): RequestState<User> {  
    return userRemoteDataSource.create(newUser)  
}
```

Criação de conta

Dentro do pacote **usecases** crie o **CreateUserUseCase**:

```
class CreateUserService(
    private val userRepository: UserRepository
) {

    suspend fun create(newUser: NewUser): RequestState<User> =
        userRepository.create(newUser)
}
```

Criação de conta

No pacote **signup** crie a **SignUpViewModelFactory** e adicione o seguinte código:

```
class SignUpViewModelFactory(
    private val createUserUseCase: CreateUserUseCase
) : ViewModelProvider.Factory {

    override fun <T : ViewModel> create(modelClass: Class<T>): T {
        return
    }
}
```

Criação de conta

No pacote **signup** crie a **SignUpViewModel** e adicione o seguinte código:

```
class SignUpViewModel (
    private val createUserUseCase: CreateUserUseCase
) : ViewModel() {
    val newUserState = MutableLiveData<RequestState<User>>()
    fun create(name: String, email: String, phone: String, password: String) {
        viewModelScope.launch {
            newUserState.value = createUserUseCase.create(
                NewUser(
                    name,
                    email,
                    phone,
                    password
                )
            )
        }
    }
}
```

MÁSCARA DE CAMPOS: CRIANDO MÁSCARA PARA O CAMPO DE TELEFONE



BIBLIOTECA CANARINHO

Esta biblioteca é um conjunto de utilitários para trabalhar com **padrões brasileiros no Android**.

Entre os padrões implementados encontramos:

Formatador e validador de CPF

Formatador e validador de CNPJ

Formatador e validador de boleto bancário (e linha digitável)

Formatador e validador de CEP

Formatador de telefone

Formatador de valores financeiros (vírgula para milhares e ponto para decimais com duas casas)

Estes são utilizados para implementar **TextWatchers** que formatam e validam a digitação do usuário.

Para mais informações acesse o link da biblioteca:

<https://github.com/concretesolutions/canarinho>

COMPLEMENTO

Caso queira um exemplo e não utilizar lib segue o link:

<https://receitasdecodigo.com.br/android/como-inserir-mascara-em-um-edittext-no-android>

UTILIZANDO A BIBLIOTECA CANARINHO

Abra o arquivo **build.gradle (app)** e adicione a seguinte linha em dependencies:

implementation 'br.com.concrete:canarinho:2.0.1'

Lembre-se após adicionar a biblioteca sincronize o projeto.

UTILIZANDO A BIBLIOTECA CANARINHO

Exemplo de utilização:

```
etPhoneSignUp.addTextChangedListener(TelefoneTextWatcher(object:  
EventoDeValidacao{  
    override fun totalmenteValido(valorAtual: String?) {  
  
    }  
  
    override fun invalido(valorAtual: String?, mensagem: String?) {  
  
    }  
  
    override fun parcialmenteValido(valorAtual: String?) {  
  
    }  
}))
```

CRIANDO A CONTA: AGORA IREMOS CRIAR UM USUÁRIO NO AUTH



Criação de conta

Abra o arquivo **SignUpFragment** e adicione o código em negrito:

```
@ExperimentalCoroutinesApi
class SignUpFragment : BaseFragment() {

    override val layout = R.layout.fragment_sign_up

    private val signUpViewModel: SignUpViewModel by lazy {
        ViewModelProvider(
            this,
            SignUpViewModelFactory(
                CreateUserUseCase(
                    UserRepositoryImpl(
                        UserRemoteDataSourceImpl( FirebaseAuth.getInstance() )
                    )
                )
            )
        ).get(SignUpViewModel::class.java)
    }
}
```

Criação de conta

Abra o arquivo **SignUpFragment** e adicione o código em negrito:

```
private lateinit var etUserNameSignUp : EditText
private lateinit var etEmailSignUp : EditText
private lateinit var etPhoneSignUp : EditText
private lateinit var etPasswordSignUp : EditText
private lateinit var cbTermsSignUp : LottieAnimationView
private lateinit var tvTerms : TextView
private lateinit var btCreateAccount : Button
private lateinit var btLoginSignUp : TextView

private var checkBoxDone = false

override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)

    setUpView(view)

    registerObserver()
}
```

Criação de conta

Abra o arquivo **SignUpFragment** e adicione o código em negrito:

```
private fun setUpView(view: View) {  
  
    etUserNameSignUp = view.findViewById(R.id.etUserNameSignUp)  
    etEmailSignUp = view.findViewById(R.id.etEmailSignUp)  
    etPhoneSignUp = view.findViewById(R.id.etPhoneSignUp)  
    etPasswordSignUp = view.findViewById(R.id.etPasswordSignUp)  
    cbTermsSignUp = view.findViewById(R.id.cbTermsSignUp)  
    tvTerms = view.findViewById(R.id.tvTerms)  
    btCreateAccount = view.findViewById(R.id.btCreateAccount)  
    btLoginSignUp = view.findViewById(R.id.btLoginSignUp)  
    setUpListener()  
}
```

Criação de conta

Abra o arquivo **SignUpFragment** e adicione o código em negrito:

```
private fun setUpListener() {  
  
    etPhoneSignUp.addTextChangedListener(TelefoneTextWatcher(object : EventoDeValidacao {  
        override fun totalmenteValido(valorAtual: String?) {}  
        override fun invalido(valorAtual: String?, mensagem: String?) {}  
        override fun parcialmenteValido(valorAtual: String?) {}  
    }))  
    tvTerms.setOnClickListener {  
        NavHostFragment.findNavController(this)  
            .navigate(R.id.action_signUpFragment_to_termsFragment)  
    }  
    btCreateAccount.setOnClickListener {  
        signUpViewModel.create(  
            etUserNameSignUp.text.toString(),  
            etEmailSignUp.text.toString(),  
            etPhoneSignUp.text.toString(),  
            etPasswordSignUp.text.toString()  
        )  
    }  
    setUpCheckboxListener()  
}
```

Criação de conta

Abra o arquivo **SignUpFragment** e adicione o código em negrito:

```
private fun setUpCheckboxListener() {
    cbTermsSignUp.setOnClickListener {
        if (checkBoxDone) {
            cbTermsSignUp.speed = -1f
            cbTermsSignUp.playAnimation()
            checkBoxDone = false
        } else {
            cbTermsSignUp.speed = 1f
            cbTermsSignUp.playAnimation()
            checkBoxDone = true
        }
    }
}
```

Criação de conta

Abra o arquivo **SignUpFragment** e adicione o código em negrito:

```
private fun registerObserver() {
    this.signUpViewModel.newUserState.observe(viewLifecycleOwner, Observer {
        when (it) {
            is RequestState.Success -> {
                hideLoading()
                NavHostFragment.findNavController(this)
                    .navigate(R.id.main_nav_graph)
            }
            is RequestState.Error -> {
                hideLoading()
                showMessage(it.throwable.message)
            }
            is RequestState.Loading -> showLoading("Realizando a autenticação")
        }
    })
}
```

PERSISTINDO DADOS: GRAVANDO OS DADOS DO USUÁRIO NO FIRESTORE



FIRESTORE O QUE É?

O Firestore armazena dados **NoSQL** com a sintaxe **JSON**. O Firestore tem uma estrutura diferente da Realtime Database onde seus dados são divididos em **collections(coleções)** e documents(documentos) ao invés de nodes (**nós**).



Cloud
Firestore

Para complementar os estudos:

<https://www.youtube.com/watch?v=A4olAn5CtyA>

FIRESTORE - DOCUMENTS

Documento é como se fosse um objeto (sim, aquele a que estamos acostumados). Em cada objecto temos alguns valores com as suas respectivas chaves. Estes valores podem ser do tipo boolean, byte, int, float, String, arrays, datas e até null. [veja todos os tipos na documentação]

```
{  
  "id" : 1,  
  "nome": "Heider Lopes",  
  "email" : "heiderlopes@exemplo.com  
}
```

FIRESTORE - COLLECTIONS

As coleções são o conjuntos dos documentos. Elas não podem conter outras coleções e os documentos não podem conter outros documentos.

Sendo assim, uma coleção só pode conter documentos e os documentos só podem conter sub-coleções.

Para mais informações:

<https://medium.com/android-dev-moz/firestore-8d225062ffe8>

FIRESTORE - TIPO DE DADOS

O Cloud Firestore permite gravar uma variedade de tipos de dados em um documento, incluindo strings, booleanos, números, datas, nulos, além de matrizes e objetos aninhados. O Cloud Firestore sempre armazena números como duplos, independentemente do tipo de número que você usa no código.

```
val docData = hashMapOf  
    "stringExample" to "Hello world!",  
    "booleanExample" to true,  
    "numberExample" to 3.14159265,  
    "dateExample" to Timestamp(Date()),  
    "listExample" to arrayListOf(1, 2, 3),  
    "nullExample" to null  
)
```

```
val nestedData = hashMapOf  
    "a" to 5,  
    "b" to true  
)
```

```
docData["objectExample"] = nestedData  
  
db.collection("data").document("one")  
    .set(docData)  
    .addOnSuccessListener { Log.d(TAG,  
        "DocumentSnapshot successfully written!") }  
    .addOnFailureListener { e -> Log.w(TAG,  
        "Error writing document", e) }
```

FIRESTORE - OBJETOS PERSONALIZADOS

Além de ser possível utilizar objetos Map ou Dictionary para representar seus documentos, o Cloud Firestore é compatível com a gravação de documentos com classes personalizadas. Ele converte os objetos em tipos de dados compatíveis.

```
data class City(  
    val name: String? = null,  
    val state: String? = null,  
    val country: String? = null,  
    val isCapital: Boolean? = null,  
    val population: Long? = null,  
    val regions: List<String>? = null  
)
```

```
val city = City("Los Angeles", "CA",  
    "USA",  
    false, 5000000L, listOf("west_coast",  
    "socal"))  
db.collection("cities").document("LA").set(  
    city)
```

FIRESTORE - DEFINIR UM DOCUMENTO

Para criar ou substituir um único documento, use o método set(). Se o documento não existir, ele será criado.

```
val cidade = hashMapOf(  
    "nome" to "São Paulo",  
    "estado" to "SP",  
    "pais" to "Brasil"  
)  
  
db.collection("cidades").document("SP")  
    .set(cidade)  
    .addOnSuccessListener { Log.d(TAG, "Gravado com sucesso") }  
    .addOnFailureListener { e -> Log.w(TAG, "Erro na gravação", e) }
```

FIRESTORE - DEFINIR UM DOCUMENTO

Se o documento existir, o conteúdo dele será substituído pelos dados recém-fornecidos da seguinte forma, a menos que você especifique que os dados devem ser incorporados ao documento existente:

// Atualize um campo, criando o document se ele ainda não existir.

```
val data = hashMapOf("capital" to true)
```

```
db.collection("cidades").document("SP")
```

```
.set(data, SetOptions.merge())
```

FIRESTORE - ADICIONAR UM DOCUMENTO

Ao usar **set()** para criar um documento, você precisa especificar um ID para ele. Conforme exemplo apresentado anteriormente. Porém é possível que o Cloud Firestore gere um automaticamente. Para fazer isso, basta chamar **add()**:

```
val data = hashMapOf(  
    "name" to "Tokyo",  
    "country" to "Japan"  
)  
  
db.collection("cities")  
    .add(data)  
    .addOnSuccessListener { documentReference ->  
        Log.d(TAG, "DocumentSnapshot written with ID: ${documentReference.id}")  
    }  
    .addOnFailureListener { e ->  
        Log.w(TAG, "Error adding document", e)  
    }
```

FIRESTORE - ADICIONAR UM DOCUMENTO

Para atualizar alguns campos de um documento sem substituir o documento inteiro, use o método **update()**:

```
val washingtonRef = db.collection("cities").document("DC")

// Set the "isCapital" field of the city 'DC'
washingtonRef
    .update("capital", true)
    .addOnSuccessListener { Log.d(TAG, "DocumentSnapshot successfully
updated!") }
    .addOnFailureListener { e -> Log.w(TAG, "Error updating document", e) }
```

FIRESTORE - ADICIONAR UM DOCUMENTO

Em alguns casos, pode ser útil criar uma referência de documento com um ID gerado automaticamente, para usá-la mais tarde. Para este caso de uso, chame **doc()**:

```
val data = HashMap<String, Any>()

val newCityRef = db.collection("cities").document()

// Later...
newCityRef.set(data)
```

FIRESTORE VS REALTIME DATABASE

O **Firestore** nos possibilita realizar queries e filtragens de dados, onde o Realtime Database nos limita a utilizar queries e filtros utilizando apenas um atributo e/ou uma condição:

Realtime Database:

```
ref.orderByChild("carro").equalTo("Cruze");
```

Firestore:

```
ref.whereEqualTo("carro", "Cruze").whereLessThan("ano", 2014);
```

FIREBASE FIRESTORE: PERSISTINDO OUTROS DADOS DO USUÁRIO



PERSISTÊNCIA DE DADOS

No console do **Firebase** crie o banco de dados



PERSISTÊNCIA DE DADOS

Neste primeiro momento deixaremos o nosso banco aberto para gravações:

Criar banco de dados

1 Regras seguras para o Cloud Firestore 2 Defina o local do Cloud Firestore

Depois de definir sua estrutura de dados, será necessário criar regras para proteger seus dados.
[Saiba mais](#)

Iniciar no modo de produção
Por padrão, seus dados se tornarão privados. O acesso a gravação/leitura do cliente será apenas concedido conforme especificado por suas regras de segurança.

Iniciar no modo de teste
Por padrão, seus dados serão abertos para permitir uma configuração rápida. O acesso a gravação/leitura do cliente será negado após 30 dias se as regras de segurança não forem atualizadas.

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /{document} {
      allow read, write: if
        request.time < timestamp.date(2020, 7, 19);
    }
  }
}
```

! Qualquer pessoa com a referência do seu banco de dados poderá fazer leituras ou gravações nele por 30 dias

Ativar o Cloud Firestore impedirá que você use o Cloud Datastore neste projeto, principalmente no aplicativo associado ao App Engine

Cancelar **Próxima**

PERSISTÊNCIA DE DADOS

Defina o local que irá criar seu banco de dados:

The screenshot shows a step in the Cloud Firestore setup process titled 'Criar banco de dados'. It consists of two main sections: 'Regras seguras para o Cloud Firestore' (Step 1) and 'Defina o local do Cloud Firestore' (Step 2). A progress bar indicates the user is at Step 2. Below the sections, a note states: 'Sua configuração de local é onde os dados do Cloud Firestore serão armazenados.' A warning message in a red box says: '⚠ Não será possível alterar o local depois de configurá-lo. Além disso, esse será o local do seu bucket padrão do Cloud Storage.' A 'Saiba mais' link is available for more information. At the bottom, a dropdown menu shows 'nam5 (us-central)' selected under 'Local do Cloud Firestore'. A note at the bottom left says: 'Ativar o Cloud Firestore impedirá que você use o Cloud Datastore neste projeto, principalmente no aplicativo associado ao App Engine'. The bottom right features 'Cancelar' and 'Concluído' buttons.

Criar banco de dados

1 Regras seguras para o Cloud Firestore 2 Defina o local do Cloud Firestore

Sua configuração de local é onde os dados do Cloud Firestore serão armazenados.

⚠ Não será possível alterar o local depois de configurá-lo. Além disso, esse será o local do seu bucket padrão do Cloud Storage.

Saiba mais

Local do Cloud Firestore

nam5 (us-central)

Ativar o Cloud Firestore impedirá que você use o Cloud Datastore neste projeto, principalmente no aplicativo associado ao App Engine

Cancelar Concluído

PERSISTÊNCIA DE DADOS

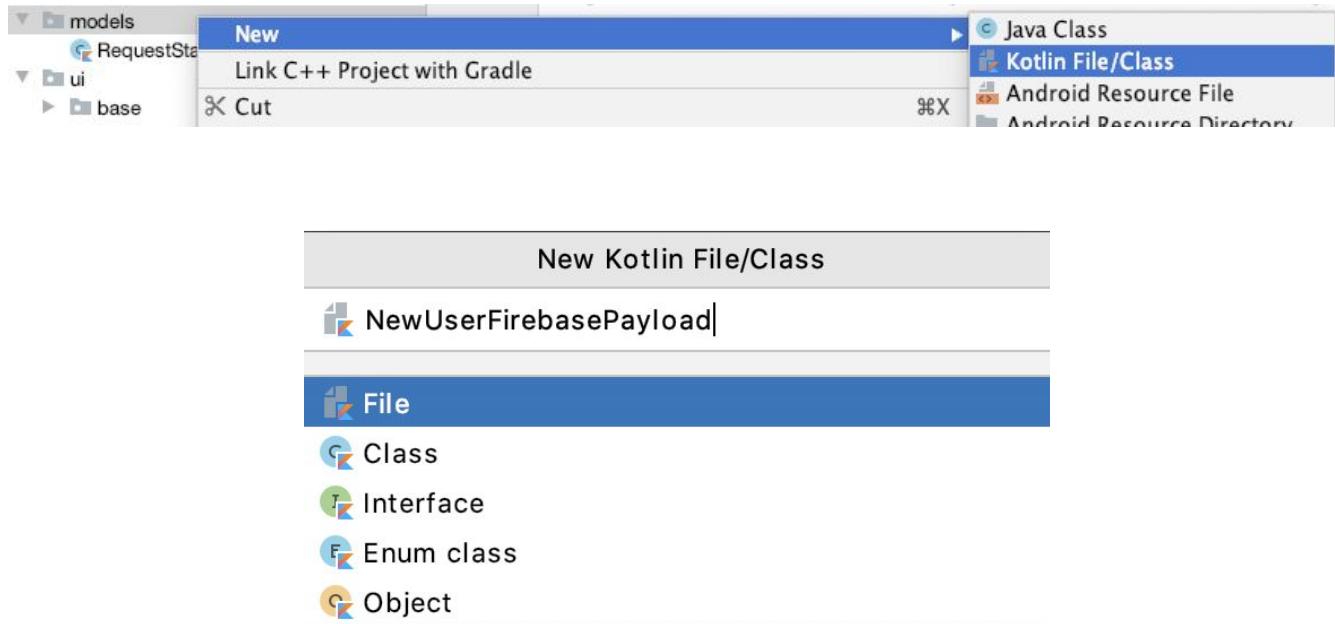
Abra o arquivo **build.gradle (app)** e adicione à seguinte dependência:

```
implementation 'com.google.firebaseio:firebase-firebase-ktx'
```

Em seguida, sincronize o projeto

PERSISTÊNCIA DE DADOS

Dentro do package **data/remote** adicione um novo pacote chamado de **models** adicione uma nova classe que irá persistir os dados no Firestore.



PERSISTÊNCIA DE DADOS

Adicione o seguinte código na classe criada do slide anterior. Observe que o campo password não será gravado no Firestore. Ele será utilizado apenas na gravação do auth para mantermos a segurança do usuário.

```
@IgnoreExtraProperties
data class NewUserFirebasePayload(
    val name: String? = null,
    val email: String? = null,
    val phone: String? = null,
    @get:Exclude val password: String? = null
)
```

PERSISTÊNCIA DE DADOS

Agora crie uma nova classe chamada **NewUserFirebasePayloadMapper** (data/remote/mapper) e adicione o seguinte código.

```
object NewUserFirebasePayloadMapper {  
  
    fun mapToNewUser (newUserFirebasePayload: NewUserFirebasePayload) = NewUser(  
        name = newUserFirebasePayload.name ?: "",  
        email = newUserFirebasePayload.email ?: "",  
        phone = newUserFirebasePayload.phone ?: "",  
        password = newUserFirebasePayload.password ?: ""  
    )  
  
    fun mapToNewUserFirebasePayload (newUser: NewUser) = NewUserFirebasePayload(  
        name = newUser.name,  
        email = newUser.email,  
        phone = newUser.phone,  
        password = newUser.password  
    )  
  
    fun mapToUser (newUserFirebasePayload: NewUserFirebasePayload) = User(  
        name = newUserFirebasePayload.name ?: ""  
    )  
}
```

PERSISTÊNCIA DE DADOS

Dentro do **UserRemoteFirebaseDataSourceImpl** adicione a dependência do Firestore no construtor

```
class UserRemoteFirebaseDataSourceImpl(  
    private val firebaseAuth: FirebaseAuth,  
    private val firebaseFirestore: FirebaseFirestore  
) : UserRemoteDataSource {
```

PERSISTÊNCIA DE DADOS

Em seguida, altere o método **create** para realizar a persistência:

```
override suspend fun create(newUser: NewUser): RequestState<User> {
    return try {
        firebaseAuth.createUserWithEmailAndPassword(newUser.email, newUser.password).await()

        val newUserFirebasePayload =
            NewUserFirebasePayloadMapper.mapToNewUserFirebasePayload(newUser)

        val userId = firebaseAuth.currentUser?.uid
        if (userId == null) {
            RequestState.Error(java.lang.Exception("Não foi possível criar a conta"))
        } else {
            firebaseFirestore
                .collection("users")
                .document(userId)
                .set(newUserFirebasePayload)
                .await()
            RequestState.Success(NewUserFirebasePayloadMapper.mapToUser(newUserFirebasePayload))
        }
    } catch (e: java.lang.Exception) {
        RequestState.Error(e)
    }
}
```

PERSISTÊNCIA DE DADOS

Abra o arquivo **BaseAuthFragment** e adicione o Firestore no Construtor do **UserRemoteFirebaseDataSourceImpl**

```
private val baseAuthViewModel: BaseAuthViewModel by lazy {  
    ViewModelProvider(  
        this,  
        BaseViewModelFactory(GetUserLoggedUseCase(UserRepositoryImpl(  
            UserRemoteFirebaseDataSourceImpl(  
                FirebaseAuth.getInstance(),  
                FirebaseFirestore.getInstance()  
            ))))  
    ).get(BaseAuthViewModel::class.java)  
}
```

PERSISTÊNCIA DE DADOS

Abra o arquivo **SignUpFragment** e adicione o Firestore no Construtor do **UserRemoteFirebaseDataSourceImpl**

```
private val signUpViewModel: SignUpViewModel by lazy {
    ViewModelProvider(
        this,
        SignUpViewModelFactory(
            CreateUserUseCase(
                UserRepositoryImpl(
                    UserRemoteDataSourceImpl(
                        FirebaseAuth.getInstance(),
                        FirebaseFirestore.getInstance()
                    )
                )
            )
        )
    )
}.get(SignUpViewModel::class.java)
}
```

PERSISTÊNCIA DE DADOS

Abra o arquivo **LoginFragment** e adicione o Firestore no Construtor do **UserRemoteFirebaseDataSourceImpl**

```
private val loginViewModel : LoginViewModel by lazy {
    ViewModelProvider(
        this,
        LoginViewModelFactory(
            LoginUseCase(
                UserRepositoryImpl(
                    UserRemoteFirebaseDataSourceImpl(
                        FirebaseAuth.getInstance(),
                        FirebaseFirestore.getInstance()
                    )
                )
            ),
            ResetPasswordUseCase(
                UserRepositoryImpl(
                    UserRemoteFirebaseDataSourceImpl(
                        FirebaseAuth.getInstance(),
                        FirebaseFirestore.getInstance()
                    )
                )
            )
        )
    ).get(LoginViewModel::class.java)
}
```

RODE O APP:
E CRIE UM NOVO USUÁRIO

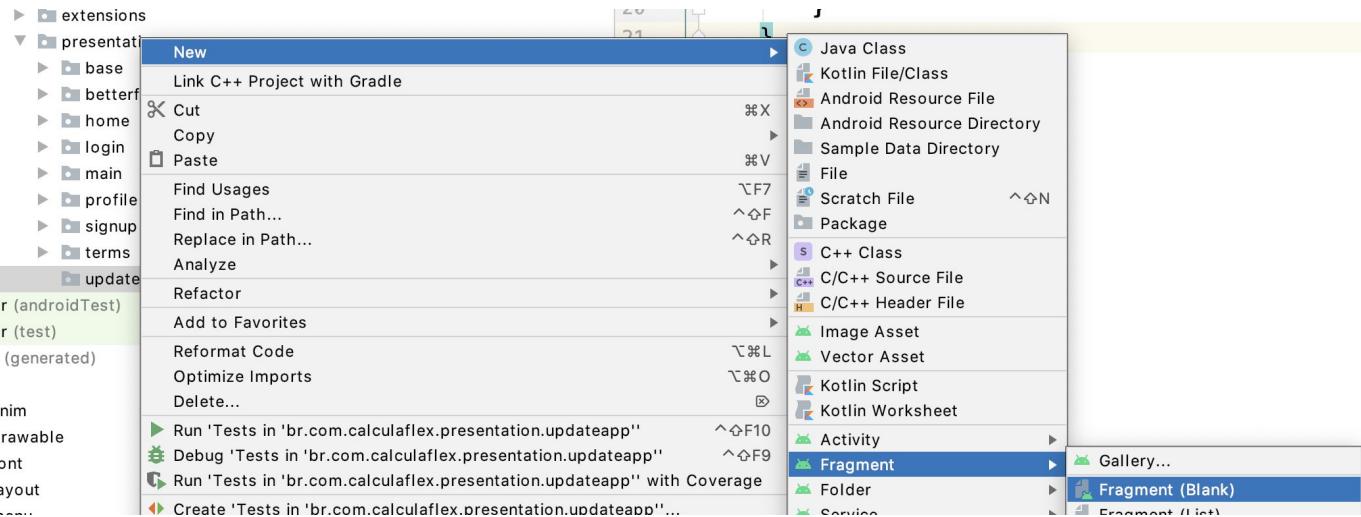


EXERCÍCIO:

CRIE UMA TELA PARA ATUALIZAÇÃO DO APPLICATIVO

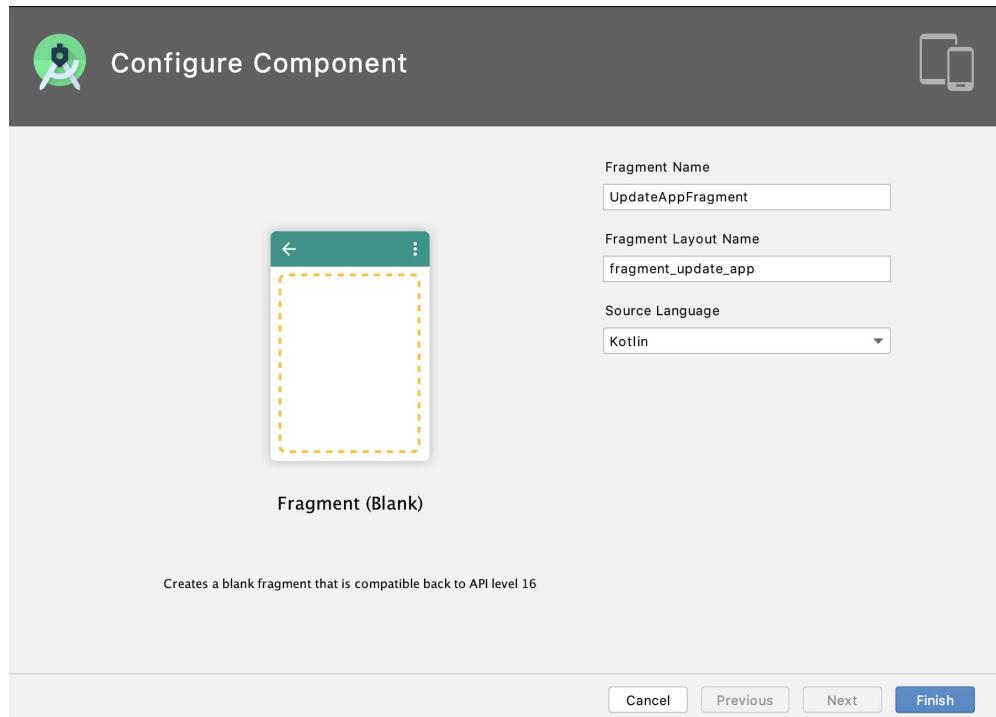
CRIANDO O UPDATE APP FRAGMENT

Dentro de **presentation** crie um package chamado **updateapp**. Dentro deste pacote adicione o **UpdateAppFragment**.



CRIANDO O UPDATE APP FRAGMENT

Dentro de **presentation** crie um package chamado **updateapp**. Dentro deste pacote adicione o **UpdateAppFragment**.





Eitcha!

Você está utilizando uma versão muito antiga do app. Você precisa atualizar para continuar utilizando

Atualizar aplicativo

Atualizar depois

CRIANDO O UPDATE APP FRAGMENT

Baixe o arquivo disponível no link abaixo e adicione-o dentro da pasta **raw** com o nome de **car_app_update.json**

<https://lottiefiles.com/14526-car-engine-pouring-oil>

O LAYOUT UPDATE APP FRAGMENT

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/containerUpdateApp"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@android:color/white"
    android:clickable="true"
    android:focusable="true"
    android:gravity="center"
    android:orientation="vertical"
    android:padding="32dp">

    <com.airbnb.lottie.LottieAnimationView
        android:id="@+id/ivUpdateApp"
        android:layout_width="192dp"
        android:layout_height="192dp"
        app:lottie_autoPlay="true"
        app:lottie_loop="true"
        ...>
```

O LAYOUT UPDATE APP FRAGMENT

```
<Button  
    android:id="@+id/btUpdateApp"  
    style="@style/button"  
    android:text="Atualizar aplicativo" />  
  
<TextView  
    android:id="@+id/btUpdateLater"  
    style="@style/link"  
    android:text="Atualizar depois"  
    android:layout_marginTop="32dp" />  
  
</LinearLayout>
```

O CÓDIGO UPDATE APP FRAGMENT

Abra o fragment **UpdateAppFragment** e adicione o seguinte código:

```
class UpdateAppFragment : Fragment() {  
  
    private lateinit var btUpdateApp: Button  
    private lateinit var btUpdateLater: TextView  
  
    override fun onViewCreated(view: View, savedInstanceState:  
        Bundle?) {  
        super.onViewCreated(view, savedInstanceState)  
  
        setUpView(view)  
    }  
}
```

O CÓDIGO UPDATE APP FRAGMENT

Abra o fragment **UpdateAppFragment** e adicione o seguinte código:

```
override fun onCreateView(
    inflater: LayoutInflater, container: ViewGroup?,
    savedInstanceState: Bundle?
): View? {
    // Inflate the layout for this fragment
    return inflater.inflate(R.layout.fragment_update_app,
    container, false)
}
```

O CÓDIGO UPDATE APP FRAGMENT

Abra o fragment **UpdateAppFragment** e adicione o seguinte código:

```
private fun setUpView(view: View) {
    btUpdateApp = view.findViewById(R.id.btUpdateApp)
    btUpdateLater = view.findViewById(R.id.btUpdateLater)

    btUpdateApp.setOnClickListener {
        openAppInStore()
    }

    btUpdateLater.setOnClickListener {
        activity?.finish()
    }
}
```

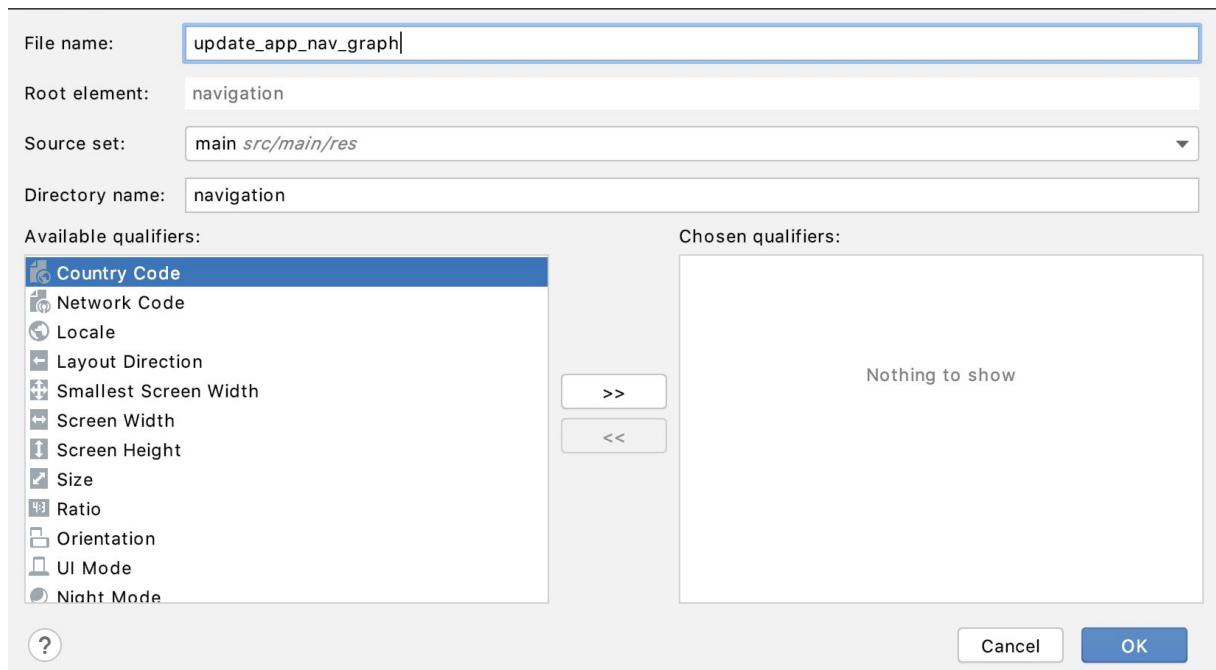
O CÓDIGO UPDATE APP FRAGMENT

Abra o fragment **UpdateAppFragment** e adicione o seguinte código:

```
private fun openAppInStore () {
    var intent: Intent
    val packageName = activity?.packageName
    try {
        intent = Intent(Intent.ACTION_VIEW,
Uri.parse("market://details?id= $packageName"))
        startActivity(intent)
    } catch (e: android.content.ActivityNotFoundException) {
        intent = Intent(
            Intent.ACTION_VIEW,
            Uri.parse("https://play.google.com/store/apps/details?id= $packageName"))
        startActivity(intent)
    }
}
```

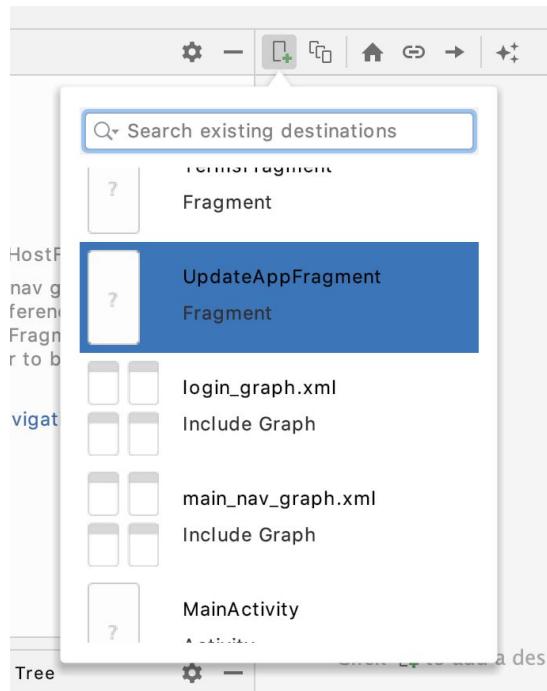
CRIANDO O UPDATE NAV GRAPH

Agora dentro de **res/navigation** crie o arquivo **update_app_nav_graph**.



CRIANDO O UPDATE NAV GRAPH

Adicione o fragmento criado conforme imagem abaixo:



REMOTE CONFIG:

ATUALIZANDO DADOS SEM BAIXAR UMA
NOVA VERSÃO DO APLICATIVO



Remote Config

É um serviço em nuvem que permite alterar o comportamento e a aparência do aplicativo sem exigir que os usuários baixem uma atualização de aplicativo. Ao usar o **Remote config**, você cria valores padrão no aplicativo que controlam o comportamento e a aparência do seu aplicativo.

Em seguida, você poderá usar posteriormente o console do **Firebase** ou a **API REST** de configuração remota para substituir os valores padrão no aplicativo de todos os usuários do aplicativo ou de segmentos da sua base de usuários.

Link da documentação:

<https://firebase.google.com/docs/remote-config/use-config-android?hl=pt-br#kotlin+ktx>

No nosso projeto, neste primeiro momento utilizaremos o **Remote Config** para configurarmos remotamente a **URL dos termos de uso** e também teremos uma chave para controlar a **versão mínima ativa do aplicativo**.

Siga os próximos passos:

Remote Config - Limitações

Se um aplicativo fizer muitas buscas em um curto período, as chamadas de busca serão limitadas e o SDK retornará **FirebaseRemoteConfigFetchThrottledException**. Antes da versão **17.0.0 do SDK**, o limite era de cinco solicitações de busca em um intervalo de 60 minutos (as versões mais recentes têm limites mais permissivos).

Para possibilitar a iteração rápida em um projeto com até 10 desenvolvedores, adicione temporariamente um **objeto FirebaseRemoteConfigSettings** com um intervalo mínimo de busca baixo (setMinimumFetchIntervalInSeconds) no seu app.

O intervalo de busca **mínimo padrão do Remote Config é de 12 horas**, o que significa que a busca pelas configurações no back-end não acontecerá mais de uma vez em uma janela de 12 horas, independentemente de quantas chamadas de busca forem realmente realizadas. O intervalo mínimo de busca é determinado especificamente na seguinte ordem:

1. O parâmetro em fetch(long)
2. O parâmetro em FirebaseRemoteConfigSettings.setMinimumFetchIntervalInSeconds(long)
3. O valor padrão de 12 horas

Para definir o intervalo mínimo de busca para um valor personalizado, use FirebaseRemoteConfigSettings.Builder.setMinimumFetchIntervalInSeconds(long).

IMPLEMENTANDO O REMOTE CONFIG

Integrando o Remote Config

Abra o arquivo **build.gradle (app)** e atualize à biblioteca importada caso necessário e adicione à **firebase-analytics** (necessária para a segmentação condicional de instâncias de apps para propriedades do usuário, públicos-alvo e Firebase Previsões).

```
implementation 'com.google.firebaseio:firebase-core'  
implementation 'com.google.firebaseio:firebase-analytics-ktx'  
implementation 'com.google.firebaseio:firebase-config-ktx'
```

E sincronize o projeto:

A screenshot of the 'Sync Now' button from the Android Studio interface. The button is rectangular with rounded corners, a light blue gradient background, and white text. It is positioned at the bottom of a yellow rectangular bar.

Sync Now

Integrando o Remote Config

Adicione as configurações no Remote Config (console do Firebase)

The screenshot shows the Firebase console interface for creating a new parameter. On the left, there's a sidebar with various analytics and marketing tools like Dashboards, Events, Conversions, Audiences, Funnels, User Properties, Latest Release, Retention, StreamView, and DebugView. The 'Remote Config' option is highlighted with a red box. The main area is titled 'Criar o primeiro parâmetro' (Create your first parameter). It explains that parameters are key-value pairs used for signaling resources and more, allowing remote updates without a release. A text input field is filled with 'min_version_app'. The 'Type of data' dropdown is set to 'Número' (Number) with value '123'. Below it, a 'Description' field contains the placeholder 'Opcional'. Under 'Default value', there's a text input with the number '1'. A checkbox 'Usar como padrão no app' (Use as default in app) is checked. At the bottom, there are 'Cancelar' (Cancel) and 'Salvar' (Save) buttons.

Criar o primeiro parâmetro

Parâmetros são pares de chave-valor que podem ser usados como sinalizações de recursos e muito mais. O app periodicamente fará uma busca de parâmetros no servidor, permitindo que você altere remotamente as configurações do app sem fazer o envio de um lançamento. [Saiba mais](#)

Nome do parâmetro (chave) ②

min_version_app

Tipo de dados

123 Número

Description

Opcional

Default value

1

Usar como padrão no app

Adicionar novo

Cancelar

Salvar

Integrando o Remote Config

Após adicionar a chave clique em **Publicar alterações**

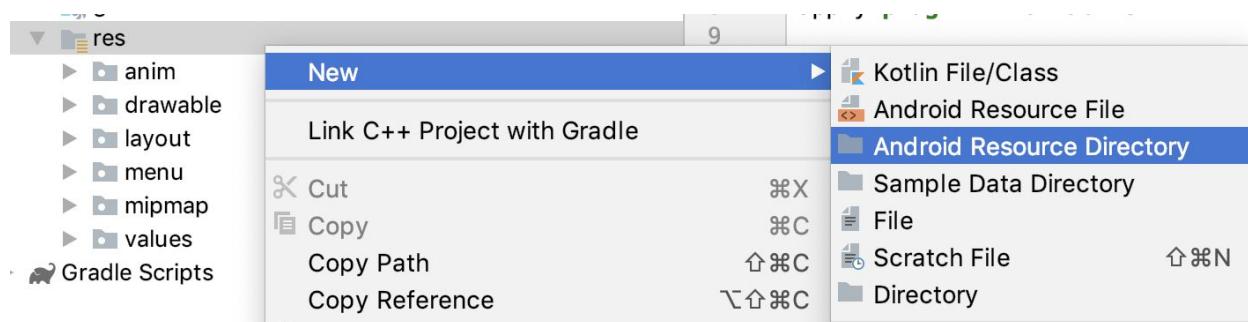
The screenshot shows the Firebase Remote Config interface. At the top, there is a message 'Alterações não publicadas' with an exclamation mark icon. To its right are two buttons: 'Descartar todas' and 'Publicar alterações', with 'Publicar alterações' being highlighted by a red rectangle. Below this, a section titled 'Buscas nas últimas 24 horas' shows a message: 'Não há métricas de busca disponíveis para este modelo.' In the main configuration area, there is a table with two rows. The first row has columns for 'Nome' (min_version_app), 'Condições e valores' (with a 'Rascunho' note), and '% de busca'. The second row has columns for 'Nome' (terms_url), 'Condições e valores' (with a 'Rascunho' note), and 'Valor padrão' (https://calcula-flex-f8d41.web.app/). There are also buttons for 'Adicionar parâmetro', a folder icon, a clock icon, and three vertical dots.

Nome	Condições e valores	% de busca
min_version_app	Valor padrão 1 ! Rascunho	
terms_url	Valor padrão https://calcula-flex-f8d41.web.app/ ! Rascunho	

Integrando o Remote Config

Agora é hora de implementar a **configuração do Firebase Remote** em nosso aplicativo.

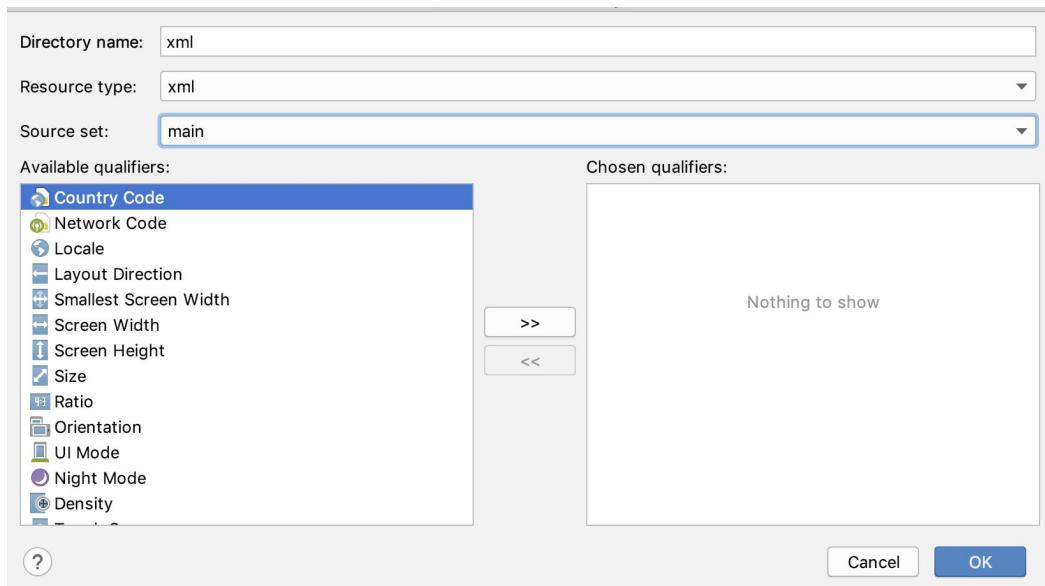
Primeiro precisamos criar um arquivo chamado **remote_config_defaults.xml** na pasta **xml** (dentro de resources - caso não exista basta criá-la).



Integrando o Remote Config

Agora é hora de implementar a **configuração do Firebase Remote** em nosso aplicativo.

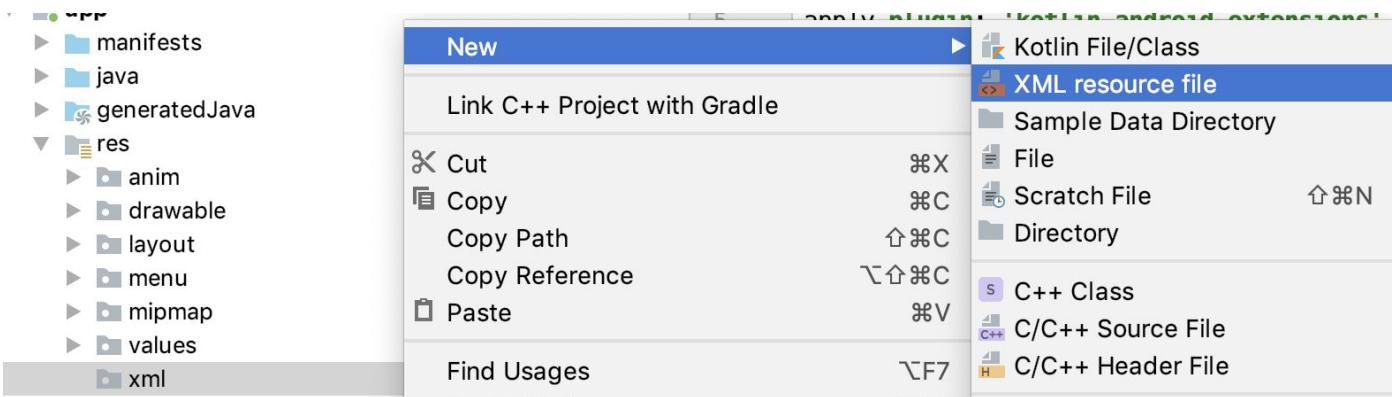
Primeiro precisamos criar um arquivo chamado **remote_config_defaults.xml** na pasta **xml** (dentro de resources - caso não exista basta criá-la).



Integrando o Remote Config

Agora é hora de implementar a **configuração do Firebase Remote** em nosso aplicativo.

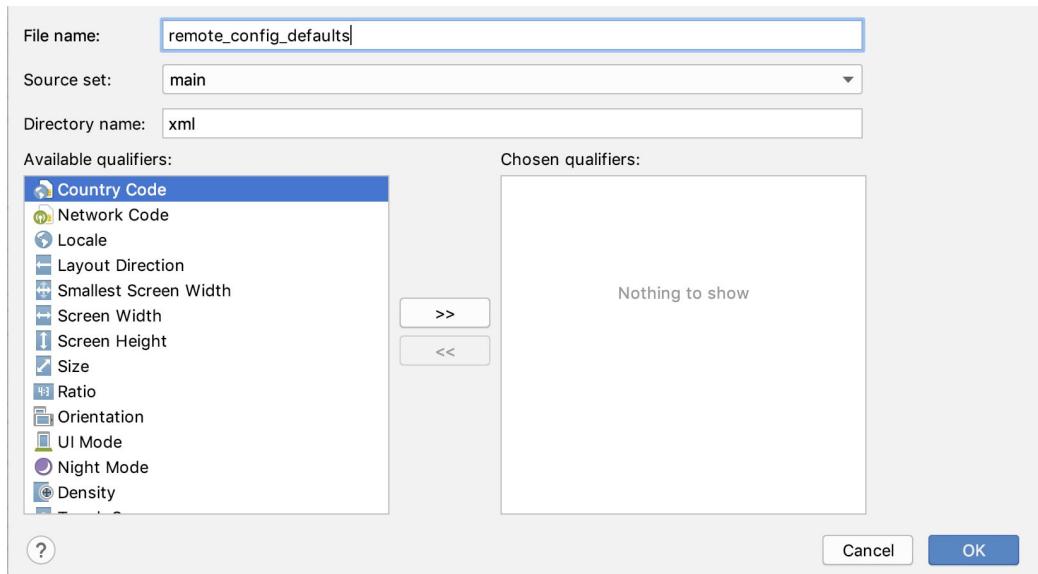
Primeiro precisamos criar um arquivo chamado **remote_config_defaults.xml** na pasta **xml** (dentro de resources - caso não exista basta criá-la).



Integrando o Remote Config

Agora é hora de implementar a **configuração do Firebase Remote** em nosso aplicativo.

Primeiro precisamos criar um arquivo chamado **remote_config_defaults.xml** na pasta **xml** (dentro de resources - caso não exista basta criá-la).



Integrando o Remote Config

Adicione o seguinte código:

```
<?xml version="1.0" encoding="utf-8"?>
<defaultsMap>
    <entry>
        <key>min_version_app</key>
        <value>1</value>
    </entry>
</defaultsMap>
```

Integrando o Remote Config

Agora iremos criar uma classe que irá retornar uma instância de objeto do **Remote Config** com a definição do intervalo mínimo de busca para permitir atualizações frequentes e outras configurações do nosso objeto.

Integrando o Remote Config

Crie uma nova classe Kotlin chamada **RemoteConfigUtils.kt** dentro de um pacote chamado **data/remote/utils/firebase**



CÓDIGO DA CLASSE REMOTECONFIGUTILS

```
object RemoteConfigUtils {

    suspend fun getFirebaseRemoteConfig(): FirebaseRemoteConfig {

        val remoteConfig = Firebase.remoteConfig

        val configSettings = remoteConfigSettings {
            minimumFetchIntervalInSeconds = 3600
            fetchTimeoutInSeconds = 60
        }

        remoteConfig.setDefaultsAsync(R.xml.remote_config_defaults)
        remoteConfig.setConfigSettingsAsync(configSettings)

        val cacheExpiration = if (BuildConfig.DEBUG) 0L else 720L
        remoteConfig.fetch(cacheExpiration).await()
        remoteConfig.activate().await()

        return remoteConfig
    }

    suspend fun fetchAndActivate(): Task<Boolean> {
        return getFirebaseRemoteConfig().activate()
    }
}
```

Integrando o Remote Config

Dentro do pacote **domain/repository** crie um novo repositório chamado **AppRepository** e adicione o seguinte código:

```
interface AppRepository {  
    suspend fun getMinVersionApp(): RequestState<Int>  
}
```

Integrando o Remote Config

Dentro do pacote **data/datasource** crie um novo datasource chamado **AppRemoteDataSource** e adicione o seguinte código:

```
interface AppRemoteDataSource {  
    suspend fun getMinVersionApp(): RequestState<Int>  
}
```

Integrando o Remote Config

Dentro do pacote **calculaflex** crie um pacote chamado de **extensions** e adicione o arquivo **GsonExt.kt**



Integrando o Remote Config

Abra o arquivo **build.gradle (app)** e adicione o seguinte código:

```
implementation 'com.google.code.gson:gson:2.8.7'
```

GsonExt

Segue o código da classe criada anteriormente:

```
@Throws(JsonSyntaxException::class)
suspend fun <T> Gson.fromRemoteConfig(keyRemoteConfig: String, classoft:
Class<T>): T? {
    val json =
        RemoteConfigUtils.getFirebaseRemoteConfig().getString(keyRemoteConfig)
    val `object` = fromJson(json, classoft)
    return Primitives.wrap(classoft).cast(`object`)
}

suspend fun <T> Gson.fromListRemoteConfig(
    keyRemoteConfig: String,
    clazz: Class<T>?
): MutableList<T>? {
    val json =
        RemoteConfigUtils.getFirebaseRemoteConfig().getString(keyRemoteConfig)
    val typeoft = TypeToken.getParameterizedMutableList::class.java, clazz).type
    return Gson().fromJson(json, typeoft)
}
```

Integrando o Remote Config

Dentro do pacote **data/datasource** crie uma nova implementação do datasource chamado **AppRemoteFirebaseDataSourceImpl** e adicione o seguinte código:

```
@ExperimentalCoroutinesApi
class AppRemoteFirebaseDataSourceImpl: AppRemoteDataSource {

    override suspend fun getMinVersionApp(): RequestState<Int> {
        val minVersion = Gson().fromRemoteConfig("min_version_app",
Int::class.java) ?: 0
        return RequestState.Success(minVersion)
    }
}
```

Integrando o Remote Config

Dentro do pacote **data/repository** crie uma nova implementação do repositório chamado **AppRepositoryImpl** e adicione o seguinte código:

```
class AppRepositoryImpl(  
    private val appRemoteDataSource: AppRemoteDataSource  
) : AppRepository {  
  
    override suspend fun getMinVersionApp(): RequestState<Int>  
{  
        return appRemoteDataSource.getMinVersionApp()  
    }  
}
```

Integrando o Remote Config

Agora dentro de domain/usecases crie o caso de uso para pegar a versão minima **GetMinAppVersionUseCase**.

```
class GetMinAppVersionUseCase(
    private val appRespository: AppRepository
) {

    suspend fun getMinVersionApp(): RequestState<Int> =
        appRespository.getMinVersionApp()

}
```

INTEGRANDO COM A BASE FRAGMENT

Integrando o Remote Config

Crie a classe **BaseViewModelFactory** dentro do pacote **presentation/base** e adicione o seguinte código:

```
class BaseViewModelFactory(
    private val getMinAppVersionUseCase: GetMinAppVersionUseCase
): ViewModelProvider.Factory {

    override fun <T : ViewModel> create(modelClass: Class<T>): T {
        return
    modelClass.getConstructor(GetMinAppVersionUseCase::class.java)
        .newInstance(getMinAppVersionUseCase)
    }
}
```

Integrando o Remote Config

Crie a classe **BaseViewModel** dentro do pacote **presentation/base** e adicione o seguinte código:

```
class BaseViewModel(
    private val getMinAppVersionUseCase: GetMinAppVersionUseCase
) : ViewModel() {

    var minVersionAppState = MutableLiveData<RequestState<Int>>()

    fun getMinVersion() {
        viewModelScope.launch {
            minVersionAppState.value =
                getMinAppVersionUseCase.getMinVersionApp()
        }
    }
}
```

ALTERANDO A BASEFRAGMENT

```
@ExperimentalCoroutinesApi
abstract class BaseFragment : Fragment() {

    abstract val layout: Int

    private lateinit var loadingView: View

    private val baseViewModel: BaseViewModel by lazy {
        ViewModelProvider(
            this,
            BaseViewModelFactory(
                GetMinAppVersionUseCase(
                    AppRepositoryImpl(
                        AppRemoteFirebaseDataSourceImpl()
                    )
                )
            )
        ).get(BaseViewModel::class.java)
    }
}
```

ALTERANDO A BASEFRAGMENT

```
override fun onCreateView(
    inflater: LayoutInflater,
    container: ViewGroup?,
    savedInstanceState: Bundle?
): View? {

    val screenRootView = FrameLayout(requireContext())

    val screenView = inflater.inflate(layout, container, false)

    loadingView = inflater.inflate(R.layout.include_loading, container,
false)

    screenRootView.addView(screenView)
    screenRootView.addView(loadingView)

    registerObserver()
    return screenRootView
}
```

ALTERANDO A BASEFRAGMENT

```
private fun registerObserver() {
    baseViewModel.minVersionAppState.observe(viewLifecycleOwner, Observer {
        when (it) {
            is RequestState.Loading -> {
                showLoading()
            }
            is RequestState.Success -> {
                hideLoading()
                if (it.data > BuildConfig.VERSION_CODE) {
                    startUpdateApp()
                }
            }
            is RequestState.Error -> {
                NavHostFragment.findNavController(this).navigate(
                    R.id.login_graph, bundleOf(
                        NAVIGATION_KEY to
                )
                NavHostFragment.findNavController(this).currentDestination?.id
                    )
            }
            hideLoading()
        }
    })
}
```

ALTERANDO A BASEFRAGMENT

```
private fun startUpdateApp() {  
    val navOptions = NavOptions.Builder()  
        .setPopUpTo(R.id.updateAppFragment, true)  
        .build()  
  
    findNavController().setGraph(R.navigation.update_app_nav_graph)  
    findNavController().navigate(R.id.updateAppFragment, null,  
navOptions)  
}  
  
override fun onResume() {  
    super.onResume()  
    baseViewModel.getMinVersion()  
}
```

ALTERANDO A BASEFRAGMENT

```
fun showLoading(message: String = "Processando a requisição") {  
    loadingView.visibility = View.VISIBLE  
  
    if (message.isNotEmpty())  
        loadingView.findViewById<TextView>(R.id.tvLoading).text  
    = message  
}  
  
fun hideLoading() {  
    loadingView.visibility = View.GONE  
}  
  
fun showMessage(message: String?) {  
    Toast.makeText(requireContext(), message,  
    Toast.LENGTH_SHORT).show()  
}
```

IMPLEMENTANDO A UPDATE APP FRAGMENT

UpdateAppFragment

```
class UpdateAppFragment : Fragment() {  
  
    private lateinit var btUpdateApp: Button  
    private lateinit var btUpdateLater: TextView  
  
    override fun onCreateView(  
        inflater: LayoutInflater, container: ViewGroup?,  
        savedInstanceState: Bundle?  
    ): View? {  
        return inflater.inflate(R.layout.fragment_update_app, container,  
false)  
    }  
  
    override fun onViewCreated(view: View, savedInstanceState: Bundle?)  
    {  
        super.onViewCreated(view, savedInstanceState)  
  
        setUpView(view)  
    }  
}
```

UpdateAppFragment

```
private fun setUpView(view: View) {  
    btUpdateApp = view.findViewById(R.id.btUpdateApp)  
    btUpdateLater = view.findViewById(R.id.btUpdateLater)  
  
    btUpdateApp.setOnClickListener {  
        openAppInStore()  
    }  
  
    btUpdateLater.setOnClickListener {  
        activity?.finish()  
    }  
}
```

UpdateAppFragment

```
private fun openAppInStore() {
    var intent: Intent
    val packageName = activity?.packageName
    try {
        intent = Intent(Intent.ACTION_VIEW,
Uri.parse("market://details?id=$packageName"))
        startActivity(intent)
    } catch (e: android.content.ActivityNotFoundException) {
        intent = Intent(
            Intent.ACTION_VIEW,
            Uri.parse("https://play.google.com/store/apps/details?id=$packageName"))
        startActivity(intent)
    }
}
```

RODE O APP NOVAMENTE E OBSERVE O RESULTADO :)

CRIANDO NOSSA DASHBOARD:

IREMOS CRIAR NOSSA DASHBOARD DINÂMICO E
APLICANDO O CONCEITO DE FEATURE TOGGLE



Integrando o Storage

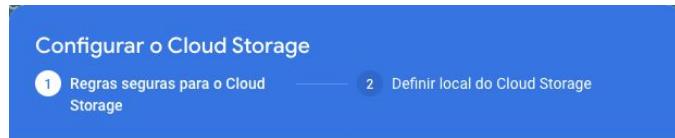
Primeiro, iremos adicionar as imagens no servidor. Para isso, utilizaremos o serviço **Storage** do Firebase.

Clique em **Storage**, em seguida, **Primeiros Passos**

The screenshot shows the Firebase console interface. On the left, there's a sidebar with project settings and links to Authentication, Database, Storage, Hosting, Functions, and Machine Learning. The main area is titled 'Storage' and contains the following text: 'Armazenar e recuperar arquivos gerados pelo usuário, como imagens, áudio e vídeos, sem o código do servidor'. Below this is a 'Primeiros passos' button. To the right of the text is a graphic featuring several overlapping files and images, including a blue folder, a white document with a blue logo, and two photo frames showing a night sky and a landscape.

FIREBASE STORAGE

Utilizaremos as configurações padrões nesse primeiro momento:

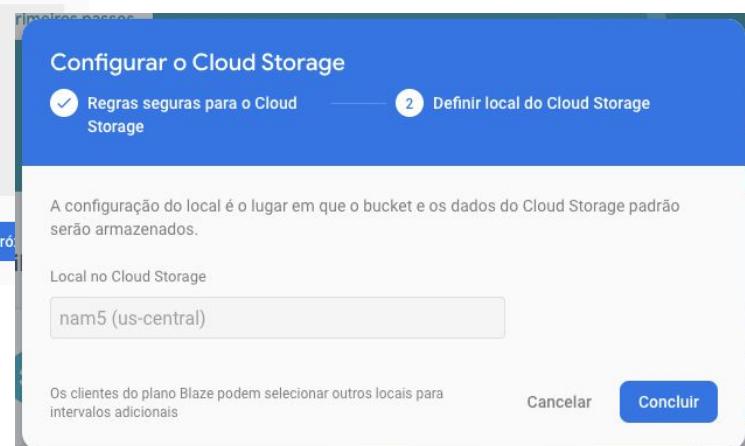


Por padrão, suas regras permitem todas as leituras e gravações de usuários autenticados.

Após definir a estrutura de dados, será necessário criar regras para proteger seus dados. [Saiba mais](#)

```
service firebase.storage {  
  match /b/{bucket}/o {  
    match /{allPaths=**} {  
      allow read, write: if request.auth != null;  
    }  
  }  
}
```

A screenshot of the Cloud Storage security rules editor. It shows the JSON-based security rules above. Below the rules, there are "Cancelar" (Cancel) and "Próximo" (Next) buttons.



FIREBASE STORAGE

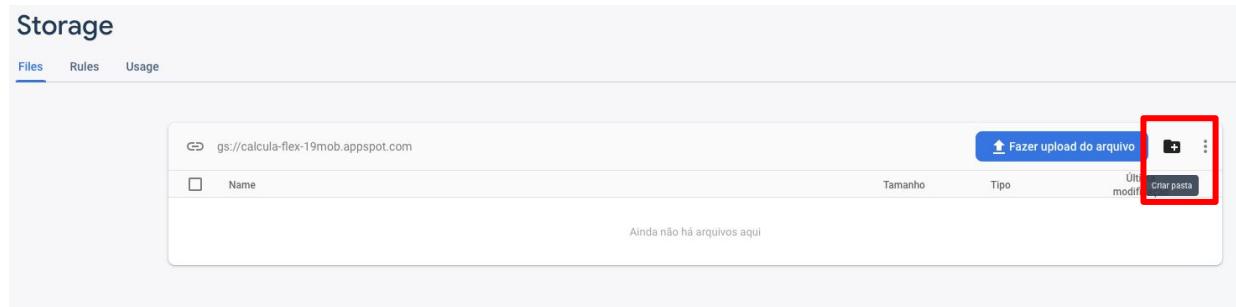
Crie um novo diretório chamado **menu_dashboard**

Storage

Files Rules Usage

	Name	Tamanho	Tipo	Última modificação	Ações
<input type="checkbox"/>	gs://calcula-flex-19mob.appspot.com				Fazer upload do arquivo + ... Criar pasta

Ainda não há arquivos aqui



FIREBASE STORAGE

Crie um novo diretório chamado **menu_dashboard**

Nome da pasta

[Cancelar](#) [Adicionar pasta](#)

Baixe as imagens do repositório:

https://github.com/heiderlopes/calcular_flex_2_dashboard_icones

Faça o **upload** das imagens para o diretório criado

REMOTE CONFIG

Segue o link com uma versão do menu Dashboard com as primeiras ações do aplicativo:

Copie o link e adicione o conteúdo disponível no arquivo para o **Remote Config** na chave **menu_dashboard**

<https://gist.github.com/heiderlopes/e4745e4b4ac8cefe0e3e2e067a3ebd62>

CONFIGURANDO DEEPLINKS



DEEPLINK

O deep link usado para aplicativos móveis possibilita que os usuários explorem outras áreas do app e navegue por lugares específicos por um comando, que no caso são os links direcionados.

Esse tipo de link tem a função de direcionar o usuário para uma página do app ou de outro aplicativo, então, por exemplo, você pode direcionar um usuário dentro do seu aplicativo para a página do app dentro do aplicativo do Facebook.

Ou então, outro exemplo muito recorrente: no Instagram, se alguém postar uma música linkada do Spotify nos stories, é possível abrir o link dentro do aplicativo mesmo, ouvir a música ou ter a opção de ser redirecionado para o app do Spotify, tudo isso graças ao deep link.

Além de engajar o usuário com um app que possui uma série de funcionalidades, direcionando-os para assuntos que eles têm interesse, também é possível atrair novos usuários.

Os deep links são uma ótima fonte de informação para controlar quem abre e instala o aplicativo. Muitas vezes, os desenvolvedores não sabem dizer de onde vem os downloads, se são das lojas ou de anúncios.

É possível, por meio deles, atrair usuários por anúncios, e-mails, mensagens e até mesmo fazer parcerias com outros aplicativos que linkem o seu app para dentro deles.

DEEPLINK

Os usuários podem personalizar a experiência que tem dentro do app, já que o desenvolvedor com o deep link pode fazer um convite personalizado, criar um incentivo para que o usuário fique engajado acessando promoções e ofertas.

Sobre a monetização, imagine um aplicativo de varejo. Uma notificação push direcionada para determinado produto e que também foi enviada aos usuários certos tem muito potencial de venda.

Assim, com o deep link, fará com que a pessoa entre na página certa já para consumir o produto.

O Google também é um grande aliado dos deep links. É possível criar uma ligação e aparecer nos mecanismos de busca do site, de acordo com o que for procurado pela pessoa e tiver relação com o aplicativo.

Além de trazer mais usuários para o app e também visibilidade, não deixam de ser usuários orgânicos, ou seja, com muito potencial de engajamento.

Documentação deeplinks com navigation:

<https://developer.android.com/guide/navigation/navigation-deep-link?hl=pt-br>

Dicas:

<https://proandroiddev.com/best-practices-for-deeplinking-in-android-1dc1ea060c0c>

ADICIONANDO DEEPLINK VIA NAVIGATION

Selecione a tela que será o destino do deeplink e no menu **Deep Links** clique em adicionar:

The screenshot shows the navigation editor interface. On the left, there are two fragments: 'betterFuelFragment' (selected and highlighted with a red border) and 'calculateTripFragment'. On the right, the fragment details are shown in a table:

id	betterFuelFragment
label	BetterFuelFragment
name	BetterFuelFragment (br.com.heiderlopes.calculaflex)
Arguments	
Nothing to show	
Actions	
Deep Links	
Nothing to show	

A red box highlights the 'Deep Links' section under 'Actions'.

This is a modal dialog for adding a deep link. It contains the following fields:

- URI:** Enter URI - `https://www.example.com/person/{id}`
- Auto Verify:** A checkbox.
- Add:** A blue button at the bottom right.
- Cancel:** A white button at the bottom left.

The screenshot shows the 'Deep Links' section expanded in the navigation editor. It displays a single entry:

↳ `https://www.calculaflex.com.br/betterfuel` (deepLink)

ADICIONANDO DEEPLINK VIA NAVIGATION

No modo **text** encontramos o seguinte código:

```
<fragment
    android:id="@+id/betterFuelFragment"

    android:name="br.com.calculaflex.presentation.betterfuel.BetterFuelFragment"
    android:label="BetterFuelFragment"
    tools:layout="@layout/fragment_better_fuel">
    <deepLink
        android:id="@+id/deepLink"
        app:uri="https://www.calculaflex.com.br/betterfuel" />
</fragment>
```

ADICIONANDO DEEPLINK VIA NAVIGATION

Para ativar o link direto implícito, você também precisa fazer adições ao arquivo **manifest.xml** do app.

Adicione o elemento **<nav-graph>** a uma atividade que aponta para um gráfico de navegação existente, como mostrado no exemplo abaixo:

```
<activity android:name=".MainActivity">

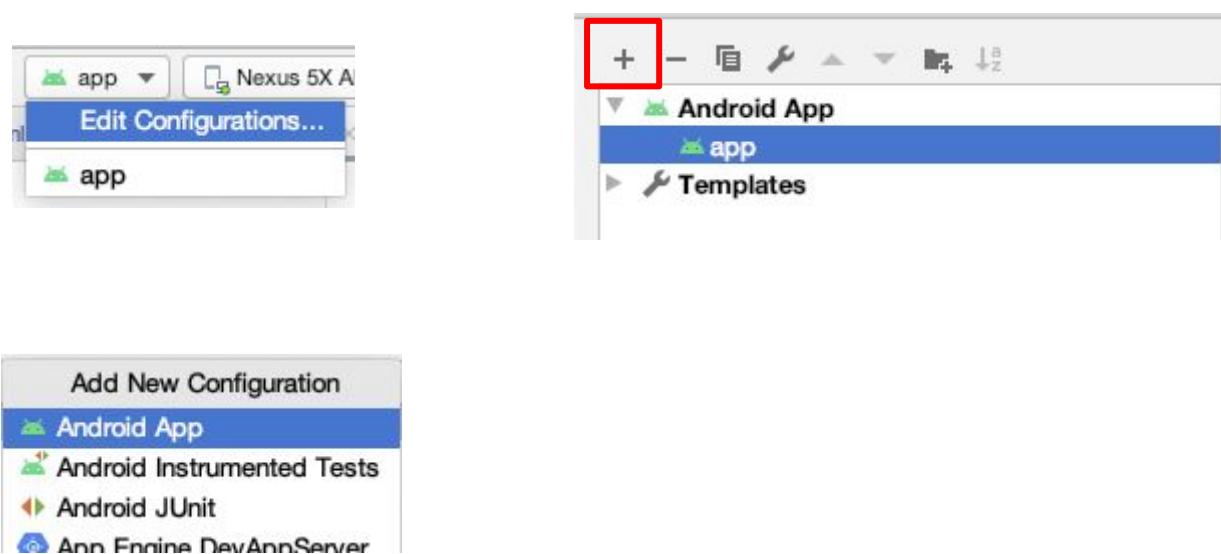
    <nav-graph android:value="@navigation/main_nav_graph" />

    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

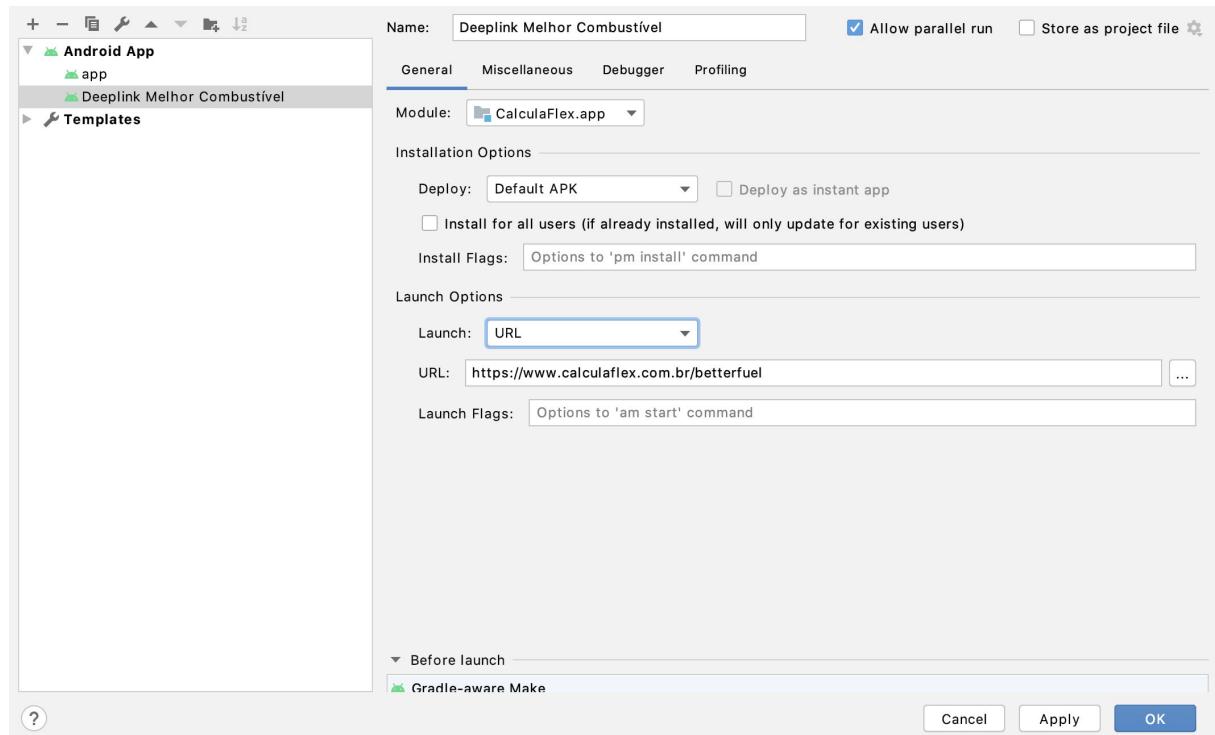
TESTANDO O DEEPLINK VIA NAVIGATION

Iremos criar o teste pelo próprio Android Studio. Para isso, cria uma nova configuração de execução:



TESTANDO O DEEPLINK VIA NAVIGATION

Realize a seguinte configuração:



TESTANDO O DEEPLINK VIA NAVIGATION

Realize a seguinte configuração:



TESTANDO O DEEPLINK VIA NAVIGATION

Crie um arquivo chamado **FragmentExt.kt** dentro do package **extensions** e adicione o método que utilizaremos nos cliques dos ítems do dashboard.

```
fun Fragment.startDeeplink(deeplink: String)
    if (deeplink.contains("://")) {
        val uri = Uri.parse(deeplink)
        val responseIntent = Intent(Intent.ACTION_VIEW, uri)
        startActivity(responseIntent)
    } else {
        val internalIntent = Intent(deeplink)
        startActivity(internalIntent)
    }
}
```

CRIANDO NOSSO MENU DINÂMICO



REMOTE CONFIG

Dentro do pacote **entity** crie um pacote chamado **enums** e adicione o arquivo **FeatureToggleState**.

```
enum class FeatureToggleState(val type: String) {  
    INVISIBLE("INVISIBLE"),  
    ENABLED("ENABLED"),  
    DISABLED("DISABLED")  
}
```

REMOTE CONFIG

Dentro do pacote **entity** e dentro dele uma classe chamada **DashboardMenu**:

```
data class DashboardMenu(  
    val title: String,  
    val subTitle: String,  
    val items: List<DashboardItem>  
)  
  
data class DashboardItem(  
    val feature: String,  
    val image: String,  
    val label: String,  
    val status: FeatureToggleState,  
    val action: DashboardAction,  
    var onDisabledListener: ((Context) -> Unit)?  
)  
  
data class DashboardAction(  
    val deeplink: String,  
    val params: HashMap<String, Any>  
)
```

REMOTE CONFIG

Dentro de **AppRepository** adicione:

```
interface AppRepository {  
  
    suspend fun getMinVersionApp(): RequestState<Int>  
  
    suspend fun getDashboardMenu(): RequestState<DashboardMenu>  
  
}
```

REMOTE CONFIG

Dentro de **AppRemoteDataSource** adicione:

```
interface AppRemoteDataSource {  
  
    suspend fun getMinVersionApp(): RequestState<Int>  
  
    suspend fun getDashboardMenu(): RequestState<DashboardMenu>  
}
```

REMOTE CONFIG

Dentro de **data/utils/firebase** adicione um arquivo **RemoteConfigKeys** e adicione

```
object RemoteConfigKeys {  
  
    const val MIN_VERSION_APP = "min_version_app"  
    const val MENU_DASHBOARD = "menu_dashboard"  
  
}
```

REMOTE CONFIG

Dentro de **AppRemoteDataSourceImpl** adicione:

```
@ExperimentalCoroutinesApi
class AppRemoteFirebaseDataSourceImpl: AppRemoteDataSource {

    override suspend fun getMinVersionApp(): RequestState<Int> {
        val minVersion = Gson().fromRemoteConfig(RemoteConfigKeys.MIN_VERSION_APP,
Int::class.java) ?: 0
        return RequestState.Success(minVersion)
    }

    override suspend fun getDashboardMenu(): RequestState<DashboardMenu> {
        val dashboardMenu = Gson().fromRemoteConfig(RemoteConfigKeys.MENU_DASHBOARD,
DashboardMenu::class.java)
        if(dashboardMenu == null) {
            return RequestState.Error(Exception("Não foi possível carregar o menu
principal"))
        } else {
            return RequestState.Success(dashboardMenu)
        }
    }
}
```

REMOTE CONFIG

Adicione o código na classe **AppRepositoryImpl**:

```
class AppRepositoryImpl(
    private val appRemoteDataSource: AppRemoteDataSource
) : AppRepository {

    override suspend fun getMinVersionApp(): RequestState<Int> {
        return appRemoteDataSource.getMinVersionApp()
    }

    override suspend fun getDashboardMenu(): RequestState<DashboardMenu> {
        return appRemoteDataSource.getDashboardMenu()
    }

}
```

REMOTE CONFIG

Crie um caso de uso chamado **GetDashboardMenuUseCase** e adicione o seguinte código:

```
class GetDashboardMenuUseCase(
    private val appRespository: AppRepository
) {

    suspend fun getDashboardMenu(): RequestState<DashboardMenu> =
        appRespository.getDashboardMenu()

}
```

REMOTE CONFIG

Dentro do pacote **presentation/utils/** crie um pacote chamado **featuretoggle** crie uma interface chamada **FeatureToggleListener**:

```
interface FeatureToggleListener {  
    fun onEnabled()  
    fun onInvisible()  
    fun onDisabled(clickListener: (Context) -> Unit)  
}
```

REMOTE CONFIG

Dentro do pacote **featuretoggle** crie uma classe chamada **FeatureToggleHelper**:

Código a partir do próximo slide

FEATURETOGGLEHELPER

```
class FeatureToggleHelper {  
  
    fun configureFeature(  
        dashboardItem: DashboardItem,  
        featureToggleListener: FeatureToggleListener  
    ) {  
        setFeatureToggleListener(dashboardItem,  
        featureToggleListener)  
    }  
}
```

FEATURETOGGLEHELPER

```
private fun setFeatureToggleListener(  
    dashboardItem: DashboardItem,  
    featureToggleListener: FeatureToggleListener  
) {  
    when (dashboardItem.status) {  
        FeatureToggleState.INVISIBLE -> {  
            featureToggleListener.onInvisible()  
        }  
        FeatureToggleState.ENABLED -> {  
            featureToggleListener.onEnabled()  
        }  
        FeatureToggleState.DISABLED -> {  
  
            featureToggleListener.onDisabled(this::showMessageUnavailable)  
        }  
    }  
}
```

FEATURETOGGLEHELPER

```
private fun showMessageUnavailable(ctx: Context) {  
    val builder = AlertDialog.Builder(ctx)  
    builder.setTitle("Eitcha!")  
    builder.setMessage("Funcionalidade temporariamente  
indisponível")  
    builder.setPositiveButton(android.R.string.yes) { dialog,  
which ->  
        Toast.makeText(  
            ctx,  
            android.R.string.yes, Toast.LENGTH_SHORT  
        ).show()  
    }  
    builder.show()  
}  
}
```

REMOTE CONFIG

Dentro do pacote **home** adicione **HomeViewModelFactory**:

```
class HomeViewModelFactory(
    private val getDashboardMenuUseCase: GetDashboardMenuUseCase
) : ViewModelProvider.Factory {

    override fun <T : ViewModel> create(modelClass: Class<T>): T {
        return
    modelClass.getConstructor(GetDashboardMenuUseCase::class.java)
        .newInstance(getDashboardMenuUseCase)
    }
}
```

REMOTE CONFIG

Dentro do pacote **home** adicione **HomeViewModel**:

```
class HomeViewModel(  
    private val getDashboardMenuUseCase: GetDashboardMenuUseCase  
) : ViewModel() {  
  
    var dashboardMenuState =  
        MutableLiveData<RequestState<DashboardMenu>>()  
  
    fun getDashboardMenu() {  
        viewModelScope.launch {  
            dashboardMenuState.value =  
                getDashboardMenuUseCase.getDashboardMenu()  
        }  
    }  
}
```

TRABALHANDO COM IMAGENS

Para a manipulação da imagem iremos utilizar a biblioteca **Picasso**. Abra o arquivo **build.gradle (app)** e adicione a seguinte dependência:

```
implementation 'com.squareup.picasso:picasso:2.71828'
```

CRIANDO O ADAPTER

Dentro da pasta home crie um arquivo chamado **HomeAdapter**.

```
class HomeAdapter(
    private var menuItems: List<DashboardItem>,
    private var clickListener: (DashboardItem) -> Unit
) : RecyclerView.Adapter<HomeAdapter.ViewHolder>() {

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {
        val view = LayoutInflater.from(parent.context).inflate(R.layout.dash_item,
parent, false)
        return ViewHolder(view)
    }

    override fun getItemCount(): Int {
        return menuItems.size
    }

    override fun onBindViewHolder(holder: ViewHolder, position: Int) {
        holder.bind(menuItems[position], clickListener)
    }
}
```

CRIANDO O ADAPTER

Dentro da pasta home crie um arquivo chamado **HomeAdapter**.

```
class ViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {

    fun bind(item: DashboardItem, clickListener: (DashboardItem) ->
Unit) {
        val label = itemView.findViewById<TextView>(R.id.textView8)
        val imageView =
itemView.findViewById<ImageView>(R.id.imageView4)

        label.text = item.label

        Picasso.get()
            .load(item.image)
            .into(imageView)

        itemView.setOnClickListener { clickListener(item) }
    }
}
```

HomeViewModel

Dentro do pacote **home** adicione **HomeViewModel**:

```
class HomeViewModel(
    private val getDashboardMenuUseCase: GetDashboardMenuUseCase
) : ViewModel() {

    var dashboardItemsState = MutableLiveData<RequestState<List<DashboardItem>>>()

    fun getDashboardMenu() {
        viewModelScope.launch {
            val response = getDashboardMenuUseCase.getDashboardMenu()
            when (response) {
                is RequestState.Success -> {
                    createMenu(response.data.items)
                }
                RequestState.Loading -> {
                    dashboardItemsState.value = RequestState.Loading
                }
                is RequestState.Error -> {
                    dashboardItemsState.value = RequestState.Error(response.throwable)
                }
            }
        }
    }
}
```

HomeViewModel

```
private fun createMenu(dashboardItem: List<DashboardItem>) {
    val dashBoardItems = arrayListOf<DashboardItem>()

    for (itemMenu in dashboardItem) {
        FeatureToggleHelper().configureFeature(
            itemMenu,
            object : FeatureToggleListener {
                override fun onEnabled() {
                    dashBoardItems.add(itemMenu)
                }
                override fun onInvisible() {}
                override fun onDisabled(clickListener: Context -> Unit) {
                    itemMenu.onDisabledListener = clickListener
                    dashBoardItems.add(itemMenu)
                }
            })
    }
    dashboardItemsState.value = RequestState.Success(dashBoardItems)
}
```

HomeFragment

Altere o código da **HomeFragment**

```
@ExperimentalCoroutinesApi
class HomeFragment : BaseAuthFragment() {

    override val layout = R.layout.fragment_home

    private lateinit var rvHomeDashboard: RecyclerView

    private val homeViewModel: HomeViewModel by lazy {
        ViewModelProvider(
            this,
            HomeViewModelFactory(
                GetDashboardMenuUseCase(
                    AppRepositoryImpl(
                        AppRemoteFirebaseDataSourceImpl()
                    )
                )
            )
        ).get(HomeViewModel::class.java)
    }
}
```

HomeFragment

Altere o código da **HomeFragment**

```
override fun onViewCreated(view: View, savedInstanceState:  
Bundle?) {  
    super.onViewCreated(view, savedInstanceState)  
    registerBackPressedAction()  
  
    setUpView(view)  
  
    registerObserver()  
    homeViewModel.getDashboardMenu()  
}  
  
private fun setUpView(view: View) {  
    rvHomeDashboard = view.findViewById(R.id.rvHomeDashboard)  
}
```

HomeFragment

Altere o código da **HomeFragment**

```
private fun registerObserver() {
    homeViewModel.dashboardItemsState.observe(viewLifecycleOwner, Observer {
        when (it) {
            is RequestState.Loading -> {
                showLoading()
            }

            is RequestState.Success -> {
                setUpMenu(it.data)
                hideLoading()
            }

            is RequestState.Error -> {
                hideLoading()
                showMessage(it.throwable.message)
            }
        }
    })
}
```

HomeFragment

Altere o código da **HomeFragment**

```
private fun setUpMenu(items: List<DashboardItem>) {
    rvHomeDashboard.adapter = HomeAdapter(items, this::clickItem)
}

private fun clickItem(item: DashboardItem) {
    item.onDisabledListener.let {
        it?.invoke(requireContext())
    }

    if (item.onDisabledListener == null) {
        when (item.feature) {
            "SIGN_OUT" -> {
                //chamar o metodo de logout
            }
            else -> {
                startDeepLink(item.action.deeplink)
            }
        }
    }
}
```

HomeFragment

Altere o código da **HomeFragment**

```
private fun registerBackPressedAction() {
    val callback = object : OnBackPressedCallback(true) {
        override fun handleOnBackPressed() {
            activity?.finish()
        }
    }
    requireActivity().onBackPressedDispatcher.addCallback(viewLifecycleOwner,
callback)
}
```

EXERCÍCIO:

IMPLEMENTE O CÓDIGO PARA EXIBIR O NOME DO USUÁRIO NO CABEÇALHO DA HOME



HomeViewModel

Altere o código da **HomeViewModel**

```
class HomeViewModel(
    private val getDashboardMenuUseCase: GetDashboardMenuUseCase,
    private val getUserLoggedUseCase: GetUserLoggedUseCase
) : ViewModel() {

    var dashboardItemsState = MutableLiveData<RequestState<List<DashboardItem>>>()
    var headerState = MutableLiveData<RequestState<Pair<String, String>>>()
    var userLogged: User? = null

    fun getDashboardMenu() {
        viewModelScope.launch {
            val dashResponse = getDashboardMenuUseCase.getDashboardMenu()
            val userReponse = getUserLoggedUseCase.getUserLogged()

            setUpUser(userReponse)
            setUpHeader(dashResponse, userReponse)
            setUpDashboard(dashResponse)
        }
    }
}
```

HomeViewModel

Altere o código da **HomeViewModel**

```
private fun setUpUser(userResponse: RequestState<User>) {
    when(userResponse) {
        is RequestState.Success -> userLogged = userResponse.data
        else -> userLogged = null
    }
}

private fun setUpHeader(
    dashResponse: RequestState<DashboardMenu>,
    userResponse: RequestState<User>
) {

    if (dashResponse is RequestState.Success && userResponse is
RequestState.Success) {
        headerState.value =
            RequestState.Success(Pair(dashResponse.data.title,
userResponse.data.name))
    } else {
        headerState.value = RequestState.Error(Exception())
    }
}
```

HomeViewModel

Altere o código da **HomeViewModel**

```
private fun setUpDashboard(dashResponse:  
RequestState<DashboardMenu>) {  
    when (dashResponse) {  
        is RequestState.Success -> {  
            createMenu(dashResponse.data.items)  
        }  
        RequestState.Loading -> {  
            dashboardItemsState.value = RequestState.Loading  
        }  
        is RequestState.Error -> {  
            dashboardItemsState.value =  
RequestState.Error(dashResponse.throwable)  
        }  
    }  
}
```

HomeViewModel

Altere o código da **HomeViewModel**

```
private fun createMenu(dashboardItem: List<DashboardItem>) {
    val dashBoardItems = arrayListOf<DashboardItem>()

    for (itemMenu in dashboardItem) {
        FeatureToggleHelper().configureFeature(
            itemMenu,
            object : FeatureToggleListener {
                override fun onEnabled() {
                    dashBoardItems.add(itemMenu)
                }

                override fun onInvisible() {}

                override fun onDisabled(clickListener: Context) -> Unit {
                    itemMenu.onDisabledListener = clickListener
                    dashBoardItems.add(itemMenu)
                }
            })
    }
    dashboardItemsState.value = RequestState.Success(dashBoardItems)
}
```

HomeViewModelFactory

Altere o código da **HomeViewModelFactory**

```
class HomeViewModelFactory(
    private val getDashboardMenuUseCase: GetDashboardMenuUseCase,
    private val getUserLoggedUseCase: GetUserLoggedUseCase
) : ViewModelProvider.Factory {

    override fun <T : ViewModel> create(modelClass: Class<T>): T {
        return modelClass.getConstructor(
            GetDashboardMenuUseCase::class.java,
            GetUserLoggedUseCase::class.java
        )
            .newInstance(getDashboardMenuUseCase,
        getUserLoggedUseCase)
    }
}
```

HomeFragment

Altere o código da **HomeFragment**

```
private val homeViewModel: HomeViewModel by lazy {
    ViewModelProvider(
        this,
        HomeViewModelFactory(
            GetDashboardMenuUseCase(
                AppRepositoryImpl(
                    AppRemoteFirebaseDataSourceImpl()
                )
            ),
            GetUserLoggedUseCase(
                UserRepositoryImpl(
                    UserRemoteFirebaseDataSourceImpl(
                        FirebaseAuth.getInstance(),
                        FirebaseFirestore.getInstance()
                    )
                )
            )
        )
    ).get(HomeViewModel::class.java)
}
```

HomeFragment

Altere o código da **HomeFragment**

```
private lateinit var tvHomeHelloUser: TextView

private fun setUpView(view: View) {
    rvHomeDashboard = view.findViewById(R.id.rvHomeDashboard)
    tvHomeHelloUser = view.findViewById(R.id.tvHomeHelloUser)
}
```

HomeFragment

Adicione no **registerObserver** da **HomeFragment**

```
homeViewModel.headerState.observe(viewLifecycleOwner, Observer {  
    when (it) {  
        is RequestState.Loading -> {  
            tvHomeHelloUser.text = "Carregando o usuário"  
        }  
  
        is RequestState.Success -> {  
            val (title, userName) = it.data  
            tvHomeHelloUser.text = String.format(title, userName)  
            hideLoading()  
        }  
  
        is RequestState.Error -> {  
            hideLoading()  
            showMessage(it.throwable.message)  
        }  
    }  
})
```

EXERCÍCIO:

CRIE A TELA PARA CALCULAR O MELHOR COMBUSTÍVEL E REALIZAR A PERSISTÊNCIA NO FIRESTORE



COMO REALIZAR O CÁLCULO?

1 - Divida o desempenho do etanol pelo desempenho da gasolina (se seu carro faz 7,3 km/litro com etanol e 10 km/l com gasolina, você deve dividir 7,3 por 10 = a 0,73 ou 73%. Pronto você achou o rendimento do carro com etanol!)

2- Faça agora o cálculo da relação de preço etanol/gasolina na bomba: divida o preço do etanol pelo preço da gasolina (ex: atualmente o litro do etanol está em R\$ 2,74 e da gasolina R\$ 4,64 = relação, então, de 0,59 ou 59%).

3 - A relação atual de preço na bomba (59%) dá uma enorme economia ao consumidor se abastecer com etanol e quando este cálculo estiver em 73%, pelo rendimento do carro neste exemplo, também estará economizando se usar gasolina.

EXERCÍCIO

Crie a tela para realizar o cálculo para se compensa abastecer com álcool ou gasolina e persista os dados no **Firestore**.



EXERCÍCIO

Abra o arquivo **styles.xml** e adicione:

```
<style name="container_edit_text_inline">
    <item name="android:layout_width">match_parent</item>
    <item name="android:layout_height">wrap_content</item>
    <item name="android:orientation">horizontal</item>
</style>

<style name="container_edit_text_inline_left" parent="container_edit_text">
    <item name="android:layout_marginEnd">8dp</item>
    <item name="android:layout_weight">0.5</item>
</style>

<style name="container_edit_text_inline_right"
parent="container_edit_text">
    <item name="android:layout_marginStart">8dp</item>
    <item name="android:layout_weight">0.5</item>
</style>
```

EXERCÍCIO

Abra o arquivo **styles.xml** e adicione:

```
<style name="section_title">
    <item name="android:layout_width">match_parent</item>
    <item name="android:layout_height">wrap_content</item>
    <item name="android:textAllCaps">false</item>
    <item name="android:textColor">@color/primaryTextColor</item>
    <item name="android:textSize">18sp</item>
    <item name="android:textStyle">bold</item>
    <item name="android:layout_gravity">center_vertical</item>
</style>

<style name="container_section">
    <item name="android:layout_width">match_parent</item>
    <item name="android:layout_height">wrap_content</item>
    <item name="android:layout_marginTop">8dp</item>
    <item name="android:layout_marginBottom">16dp</item>
</style>
```

EXERCÍCIO

Abra o arquivo **styles.xml** e adicione:

```
<style name="icon_section">
    <item name="android:layout_width">32dp</item>
    <item name="android:layout_height">32dp</item>
    <item name="android:layout_marginStart">8dp</item>
</style>

<style name="field_edit_text_number" parent="field_edit_text">
    <item name="android:maxLength">5</item>
    <item name="android:inputType">number</item>
</style>
```

EXERCÍCIO

Copie o conteúdo disponível no link abaixo para o arquivo **fragment_better_fuel.xml**:

<https://gist.github.com/heiderlopes/702fb33f7e2819a8de7de64ce0f12130>

ALTERANDO O USER

Altere o código da classe **User**:

```
data class User(  
    var id: String = "",  
    val name: String = "",  
    val email: String = "",  
    val phone: String = ""  
)
```

ALTERANDO O USER

Acesse a classe **UserRemoteFirebaseDataSourceImpl**

```
override suspend fun getUserLogged(): RequestState<User> {
    firebaseAuth.currentUser?.reload()
    val firebaseUser = firebaseAuth.currentUser
    return if (firebaseUser == null) {
        RequestState.Error(Exception("Usuário deslogado"))
    } else {
        val user = firebaseFirestore.collection("users")
            .document(firebaseUser.uid).get().await().toObject(User::class.java)

        user?.id = firebaseUser.uid

        if(user == null) {
            RequestState.Error(java.lang.Exception("Usuário não encontrado"))
        } else {
            RequestState.Success(user)
        }
    }
}
```

GRAVANDO OS DADOS DO CARRO

Crie uma classe chamada **Car**:

```
data class Car(  
    val vehicle: String = "",  
    val kmGasolinePerLiter: Double = 0.0,  
    val kmEthanolPerLiter: Double = 0.0,  
    val priceGasolinePerLiter: Double = 0.0,  
    val priceEthanolPerLiter: Double = 0.0,  
    var userId: String = ""  
)
```

GRAVANDO OS DADOS DO CARRO

Crie uma classe chamada **CarRepository**:

```
interface CarRepository {  
    suspend fun save(car: Car): RequestState<Car>  
}
```

GRAVANDO OS DADOS DO CARRO

Crie uma classe chamada **CarRemoteDataSource**:

```
interface CarRemoteDataSource {  
    suspend fun save(car: Car): RequestState<Car>  
}
```

GRAVANDO OS DADOS DO CARRO

Crie uma classe chamada **CarRemoteDataSourceImpl**:

```
class CarRemoteDataSourceImpl(
    private val firebaseFirestore: FirebaseFirestore
) : CarRemoteDataSource {

    override suspend fun save(car: Car): RequestState<Car> {
        return try {
            firebaseFirestore.collection("cars")
                .document(car.userId)
                .set(car)
                .await()
            RequestState.Success(car)
        } catch (e: Exception) {
            RequestState.Error(e)
        }
    }
}
```

GRAVANDO OS DADOS DO CARRO

Crie uma classe chamada **CarRepositoryImpl**:

```
class CarRepositoryImpl(  
    private val carRemoteDataSource: CarRemoteDataSource  
) : CarRepository {  
  
    override suspend fun save(car: Car): RequestState<Car> {  
        return carRemoteDataSource.save(car)  
    }  
}
```

GRAVANDO OS DADOS DO CARRO

Crie uma classe chamada **SaveCarUseCase**:

```
class SaveCarUseCase (
    private val getUserLoggedUseCase : GetUserLoggedUseCase ,
    private val carRepository : CarRepository
) {
    suspend fun save(car: Car): RequestState<Car> {
        val userLogged = getUserLoggedUseCase .getUserLogged()

        return when (userLogged) {
            is RequestState .Success -> {
                car. userId = userLogged .data .id
                carRepository .save(car)
            }

            is RequestState .Loading -> {
                RequestState .Loading
            }

            is RequestState .Error -> {
                RequestState .Error(Exception( "Usuário não encontrado para associar o carro" ))
            }
        }
    }
}
```

GRAVANDO OS DADOS DO CARRO

Crie uma classe chamada **FuelType** dentro do pacote **entity/enums** e adicione o seguinte código:

```
enum class FuelType {  
    GASOLINE,  
    ETHANOL  
}
```

GRAVANDO OS DADOS DO CARRO

Crie uma classe chamada **BetterFuelHolder** dentro do pacote **entity/holder** e adicione o seguinte código:

```
data class BetterFuelHolder(  
    val ethanolAverage: Double,  
    val gasAverage: Double,  
    val ethanolPrice: Double,  
    val gasPrice: Double  
)
```

GRAVANDO OS DADOS DO CARRO

Dentro do pacote **domain** crie um novo pacote chamado **utils** e dentro dele uma classe chamada **FuelCalculator.kt**

```
class FuelCalculator {

    fun betterFuel(
        betterFuelHolder: BetterFuelHolder
    ): FuelType {

        val performanceOfMyCar = betterFuelHolder.ethanolAverage /
betterFuelHolder.gasAverage

        val priceOfFuelIndice = betterFuelHolder.ethanolPrice /
betterFuelHolder.gasPrice

        return if (priceOfFuelIndice <= performanceOfMyCar) {
            FuelType.ETHANOL
        } else {
            FuelType.GASOLINE
        }
    }
}
```

GRAVANDO OS DADOS DO CARRO

Crie uma classe chamada **CalculateBetterFuelUseCase**:

```
class CalculateBetterFuelUseCase (
    private val fuelCalculator: FuelCalculator
) {

    suspend fun calculate(
        params: Params
    ): RequestState<FuelType> {

        return try {

            val betterFuel = fuelCalculator.betterFuel(
                params.betterFuelHolder
            )

            RequestState.Success(betterFuel)
        } catch (ex: Exception) {
            RequestState.Error(ex)
        }
    }

    data class Params(
        val betterFuelHolder: BetterFuelHolder
    )
}
```

GRAVANDO OS DADOS DO CARRO

Crie uma classe chamada **BetterFuelViewModelFactory**:

```
class BetterFuelViewModelFactory(
    private val saveCarUseCase: SaveCarUseCase,
    private val calculateBetterFuelUseCase: CalculateBetterFuelUseCase
) : ViewModelProvider.Factory {

    override fun <T : ViewModel> create(modelClass: Class<T>): T {
        return modelClass.getConstructor(
            SaveCarUseCase::class.java,
            CalculateBetterFuelUseCase::class.java
        )
            .newInstance(saveCarUseCase, calculateBetterFuelUseCase)
    }
}
```

GRAVANDO OS DADOS DO CARRO

Crie uma classe chamada **BetterFuelViewModel**:

```
class BetterFuelViewModel (
    private val saveCarUseCase: SaveCarUseCase,
    private val calculateBetterFuelUseCase: CalculateBetterFuelUseCase
) : ViewModel() {
    var calculateState = MutableLiveData<RequestState<FuelType>>()

    fun calculateBetterFuel(
        vehicle: String,
        kmGasolinePerLiter: Double,
        kmEthanolPerLiter: Double,
        priceGasolinePerLiter: Double,
        priceEthanolPerLiter: Double
    ) {
        val car = Car(
            vehicle,
            kmGasolinePerLiter,
            kmEthanolPerLiter,
            priceGasolinePerLiter,
            priceEthanolPerLiter,
            ""
        )
    }
}
```

GRAVANDO OS DADOS DO CARRO

Crie uma classe chamada **BetterFuelViewModel**:

```
viewModelScope.launch {
    val response = saveCarUseCase.save(car)
    when (response) {
        is RequestState.Success -> {
            val params = CalculateBetterFuelUseCase.Params(
                BetterFuelHolder(
                    car.kmEthanolPerLiter,
                    car.kmGasolinePerLiter,
                    car.priceEthanolPerLiter,
                    car.priceGasolinePerLiter
                )
            )
            calculateState.value = calculateBetterFuelUseCase.calculate(params)
        }
        is RequestState.Error -> {
            calculateState.value = RequestState.Error(Exception("Não foi possível calcular"))
        }
        is RequestState.Loading -> {
            calculateState.value = RequestState.Loading
        }
    }
}
```

CRIANDO NOSSO WATCHER

Dentro de **presentation** crie um pacote chamado **watchers**. Dentro dele uma classe chamada **DecimalTextWatcher** com o seguinte código:

```
class DecimalTextWatcher(editText: EditText, val totalDecimalNumber: Int = 2) :  
    TextWatcher {  
  
    private val editTextWeakReference: WeakReference<EditText> =  
        WeakReference(editText)  
  
    init {  
        formatValue(editTextWeakReference.get()!!).text  
    }  
  
    override fun beforeTextChanged(s: CharSequence, start: Int, count: Int, after:  
        Int) {}  
  
    override fun onTextChanged(s: CharSequence, start: Int, before: Int, count: Int)  
    {}  
  
    override fun afterTextChanged(editable: Editable) {  
        formatValue(editable)  
    }  
}  
  
fun formatValue(editText: EditText): TextFormat {  
    val decimalFormat = DecimalFormat("###,###,###,###,###,###,###,###,###,###,###")  
    decimalFormat.minimumFractionDigits = totalDecimalNumber  
    decimalFormat.maximumFractionDigits = totalDecimalNumber  
    return TextFormat(decimalFormat, true)  
}
```

CRIANDO NOSSO WATCHER

Dentro de **presentation** crie um pacote chamado **watchers**. Dentro dele uma classe chamada **DecimalTextWatcher** com o seguinte código:

```
private fun getTotalDecimalNumber(): String {  
    val decimalNumber = StringBuilder()  
    for (i in 1..totalDecimalNumber) {  
        decimalNumber.append("0")  
    }  
    return decimalNumber.toString()  
}
```

CRIANDO NOSSO WATCHER

Dentro de **presentation** crie um pacote chamado **watchers**. Dentro dele uma classe chamada **DecimalTextWatcher** com o seguinte código:

```
private fun formatValue(editable: Editable) {
    val editText = editTextWeakReference.get() ?: return
    val cleanString = editable.toString().trim().replace(" ", "")
    editText.removeTextChangedListener( this )
    val number = Math.pow(10.toDouble(), totalDecimalNumber.toDouble())
    val parsed = when (cleanString) {
        null -> BigDecimal( 0 )
        "" -> BigDecimal( 0 )
        "null" -> BigDecimal( 0 )
        else -> BigDecimal(cleanString.replace("\\D+".toRegex(), ""))
            .setScale( totalDecimalNumber, BigDecimal.ROUND_FLOOR )
            .divide(BigDecimal( number.toInt() ),
                BigDecimal.ROUND_FLOOR)
    }
    val dfnd = DecimalFormat( "#,##0.${getTotalDecimalNumber() }" )
    val formatted = dfnd.format(parsed)
    editText.setText(formatted.replace(',', '.'))
    editText.setSelection(formatted.length)
    editText.addTextChangedListener( this )
}
```

CRIANDO NOSSO WATCHER

Dentro do pacote **extensions** crie uma nova classe chamada **DoubleExt.kt** e adicione o seguinte código:

```
fun Double.format(digits: Int) = String.format("%.${digits}f", this)
```

CRIANDO NOSSO WATCHER

Dentro do pacote **extensions** crie uma nova classe chamada **EditTextExt.kt** e adicione o seguinte código:

```
fun EditText.getDouble() : Double {  
    return this.getString().toDouble()  
}
```

```
fun EditText.getString(): String {  
    return this.text.toString()  
}
```

Calculando melhor combustível

Altere o **BetterFuelFragment.kt** para realizar o cálculo:

```
@ExperimentalCoroutinesApi
class BetterFuelFragment : BaseAuthFragment() {

    override val layout = R.layout.fragment_better_fuel

    private lateinit var etCar: EditText
    private lateinit var etKmGasoline: EditText
    private lateinit var etKmEthanol: EditText
    private lateinit var etPriceGasoline: EditText
    private lateinit var etPriceEthanol: EditText
    private lateinit var btCalculate: Button
    private lateinit var btClear: TextView
```

Calculando melhor combustível

```
private val betterFuelViewModel : BetterFuelViewModel by lazy {
    ViewModelProvider(
        this,
        BetterFuelViewModelFactory(
            SaveCarUseCase(
                GetUserLoggedUseCase(
                    UserRepositoryImpl(
                        UserRemoteFirebaseDataSourceImpl(
                            FirebaseAuth.getInstance(),
                            FirebaseFirestore.getInstance()
                        )
                    )
                )
            ),
            CarRepositoryImpl(
                CarRemoteDataSourceImpl(
                    FirebaseFirestore.getInstance()
                )
            )
        ),
        CalculateBetterFuelUseCase(
            FuelCalculator()
        )
    )
}.get(BetterFuelViewModel ::class.java)
}
```

Calculando melhor combustível

```
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {  
    super.onViewCreated(view, savedInstanceState)  
    setUpView(view)  
    setUpListener()  
  
    registerObserver()  
}
```

Calculando melhor combustível

```
private fun setUpView(view: View) {  
    etCar = view.findViewById(R.id.etCar)  
    etKmGasoline = view.findViewById(R.id.etKmGasoline)  
    etKmEthanol = view.findViewById(R.id.etKmEthanol)  
    etPriceGasoline = view.findViewById(R.id.etPriceGasoline)  
    etPriceEthanol = view.findViewById(R.id.etPriceEthanol)  
    btCalculate = view.findViewById(R.id.btCalculate)  
    btClear = view.findViewById(R.id.btClear)  
}
```

Calculando melhor combustível

```
private fun setUpListener () {  
  
    etPriceGasoline .addTextChangedListener(DecimalTextWatcher( etPriceGasoline  
))  
  
    etPriceEthanol .addTextChangedListener(DecimalTextWatcher( etPriceEthanol ))  
  
    etKmGasoline .addTextChangedListener(DecimalTextWatcher( etKmGasoline ,  
1))  
    etKmEthanol .addTextChangedListener(DecimalTextWatcher( etKmEthanol , 1))  
  
    btCalculate .setOnClickListener {  
        betterFuelViewModel .calculateBetterFuel(  
            etCar .getString() ,  
            etKmGasoline .getDouble() ,  
            etKmEthanol .getDouble() ,  
            etPriceGasoline .getDouble() ,  
            etPriceEthanol .getDouble()  
        )  
    }  
}
```

Calculando melhor combustível

```
private fun registerObserver() {
    betterFuelViewModel.calculateState.observe(viewLifecycleOwner, Observer {
        when(it) {
            is RequestState.Success -> {
                hideLoading()

                val betterFuelMessage = when(it.data) {
                    FuelType.GASOLINE -> "Melhor abastecer com gasolina"
                    FuelType.ETHANOL -> "Melhor abastecer com álcool"
                }

                showMessage(betterFuelMessage)
            }
            is RequestState.Error -> {
                hideLoading()
                showMessage(it.throwable.message)
            }
            is RequestState.Loading -> {
                showLoading("Calculando o melhor combustível para você")
            }
        }
    })
}
```

EXERCÍCIO:

RECUPERANDO O CARRO NO FIRESTORE



Recuperando o carro

Adicione o código abaixo em **CarRepository**:

```
suspend fun findBy(id: String): RequestState<Car>
```

Recuperando o carro

Adicione o código abaixo em **CarRepositoryImpl**:

```
override suspend fun findBy(id: String): RequestState<Car> {  
    return carRemoteDataSource.findBy(id)  
}
```

Recuperando o carro

Adicione o código abaixo em **CarRemoteDataSource**:

```
suspend fun findBy(id: String): RequestState<Car>
```

Recuperando o carro

Adicione o código abaixo em **CarRemoteDataSourceImpl**:

```
override suspend fun findBy(id: String): RequestState<Car> {
    return try {
        val car = firebaseFirestore.collection("cars")
            .document(id)
            .get()
            .await().toObject(Car::class.java) ?: Car()
        RequestState.Success(car)
    } catch (e: Exception) {
        RequestState.Error(e)
    }
}
```

Recuperando o carro

Crie o **GetCarUseCase** dentro de **domain/usecase**

```
class GetCarUseCase(
    private val carRepository: CarRepository
) {
    suspend fun findBy(id: String): RequestState<Car> {
        return carRepository.findBy(id)
    }
}
```

Recuperando o carro

Altere o **BetterFuelViewModel**

```
class BetterFuelViewModelFactory (
    private val saveCarUseCase: SaveCarUseCase,
    private val calculateBetterFuelUseCase: CalculateBetterFuelUseCase,
    private val getCarUseCase: GetCarUseCase
) : ViewModelProvider.Factory {

    override fun <T : ViewModel?> create(modelClass: Class<T>): T {
        return modelClass.getConstructor(
            SaveCarUseCase ::class.java,
            CalculateBetterFuelUseCase ::class.java,
            GetCarUseCase::class.java
        )
        .newInstance( saveCarUseCase, calculateBetterFuelUseCase,
getCarUseCase)
    }
}
```

Recuperando o carro

Altere o **BetterFuelViewModel**

```
class BetterFuelViewModel (
    private val saveCarUseCase: SaveCarUseCase,
    private val calculateBetterFuelUseCase: CalculateBetterFuelUseCase ,
    private val getCarUseCase: GetCarUseCase
) : ViewModel() {
```

Recuperando o carro

Altere o **BetterFuelViewModel**

```
val carSelectedState = MutableLiveData<RequestState<Car>>()

fun getCar(id: String) {
    viewModelScope.launch {
        carSelectedState.value = getCarUseCase.findById(
            id
        )
    }
}
```

Recuperando o carro

Abra o arquivo **main_nav_graph.xml** e altere o deeplink conforme abaixo:

```
<deepLink
    android:id="@+id/deepLink"
    app:uri="https://www.calculaflex.com.br/betterfuel?id={id}" />

<argument android:name="id" app:argType="string"/>
```

Recuperando o carro

Abra o arquivo **HomeFragment.xml** e altere o método para chamar o formulário de cálculo:

```
private fun clickItem(item: DashboardItem) {
    item.onDisabledListener?.let {
        it?.invoke(requireContext())
    }
    if (item.onDisabledListener == null) {
        when (item.feature) {
            "SIGN_OUT" -> {
                //chamar o metodo de logout
            }
            "ETHANOL_OR_GASOLINE" -> {

startDeeplink("${item.action.deeplink}?id=${homeViewModel.userLogged?.id}")
            }
            else -> {
                //startDeeplink("${item.action.deeplink}?id=123")
                startDeeplink(item.action.deeplink)
            }
        }
    }
}
```

Recuperando o carro

Abra o arquivo **BetterFuelFragment** e altere o deeplink conforme abaixo:

```
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {  
    super.onViewCreated(view, savedInstanceState)  
    setUpView(view)  
    setUpListener()  
  
    registerObserver()  
  
    betterFuelViewModel.getCar(arguments?.getString("id") ?: "")  
}
```

Recuperando o carro

Abra o arquivo **BetterFuelFragment** e altere o deeplink conforme abaixo:

```
private val betterFuelViewModel : BetterFuelViewModel by lazy {
    ViewModelProvider(
        this,
        BetterFuelViewModelFactory(
            SaveCarUseCase(
                GetUserLoggedUseCase(
                    UserRepositoryImpl(
                        UserRemoteDataSourceImpl(
                            FirebaseAuth.getInstance(),
                            FirebaseFirestore.getInstance()
                        )
                    )
                )
            ),
            CarRepositoryImpl(
                CarRemoteDataSourceImpl(
                    FirebaseFirestore.getInstance()
                )
            )
        ),
        CalculateBetterFuelUseCase(
            FuelCalculator()
        ),
        GetCarUseCase(
            CarRepositoryImpl(
                CarRemoteDataSourceImpl(
                    FirebaseFirestore.getInstance()
                )
            )
        )
    )
} [BetterFuelViewModel ::class.java]
```

Recuperando o carro

No arquivo **BetterFuelFragment** adicione no método **registerObserver()**

```
betterFuelViewModel.carSelectedState.observe(viewLifecycleOwner, Observer {  
    when (it) {  
        is RequestState.Success -> {  
            val car = it.data  
            etCar.setText(car.vehicle)  
            etKmGasoline.setText(car.kmGasolinePerLiter.toString())  
            etKmEthanol.setText(car.kmEthanolPerLiter.toString())  
            etPriceGasoline.setText(car.priceGasolinePerLiter.toString())  
            etPriceEthanol.setText(car.priceEthanolPerLiter.toString())  
            hideLoading()  
  
        }  
        is RequestState.Error -> {  
            hideLoading()  
        }  
        is RequestState.Loading -> {  
            showLoading("Aguarde um momento")  
        }  
    }  
})
```

EXERCÍCIO:

IMPLEMENTE O SIGNOUT DO APLICATIVO



HORA DAS NOTIFICAÇÕES:

FIREBASE CLOUD MESSAGE (FCM)



FIREBASE CLOUD MESSAGING

É uma solução de mensagens entre plataformas que permite a entrega confiável de mensagens sem custo.

Usando o **FCM**, você pode notificar um app cliente de que novos e-mails ou outros dados estão disponíveis para sincronização.

Você pode enviar mensagens de notificação para promover novas interações e a retenção de usuários. Para casos de uso como mensagens instantâneas, ela pode transferir uma **payload** de até **4 KB** para um app cliente.

Para saber mais:

<https://firebase.google.com/docs/cloud-messaging/android/receive?hl=pt-br>

Para implementar no Android:

<https://firebase.google.com/docs/cloud-messaging/android/client?hl=pt-br>

IMPORTANTE:

Para todas as mensagens em que **onMessageReceived** é fornecido, seu serviço deve processar qualquer mensagem em até 20 segundos após o recebimento (10 segundos no Android Marshmallow).

O período pode ser mais curto, dependendo dos atrasos do SO incorridos antes de chamar **onMessageReceived**.

Depois disso, vários comportamentos do SO, como os limites de execução em segundo plano do Android O, podem interferir na capacidade de concluir seu trabalho. Para mais informações, consulte nossa visão geral sobre prioridade de mensagens.

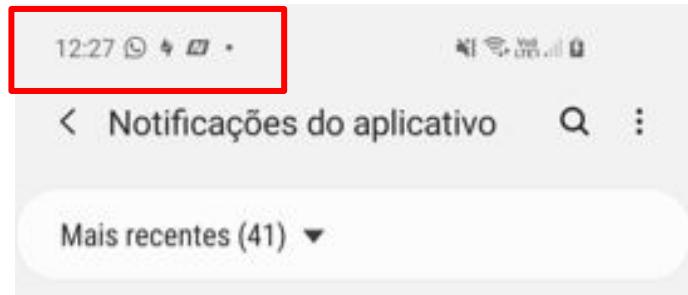
NOTIFICAÇÕES NO ANDROID



NOTIFICAÇÃO NO ANDROID

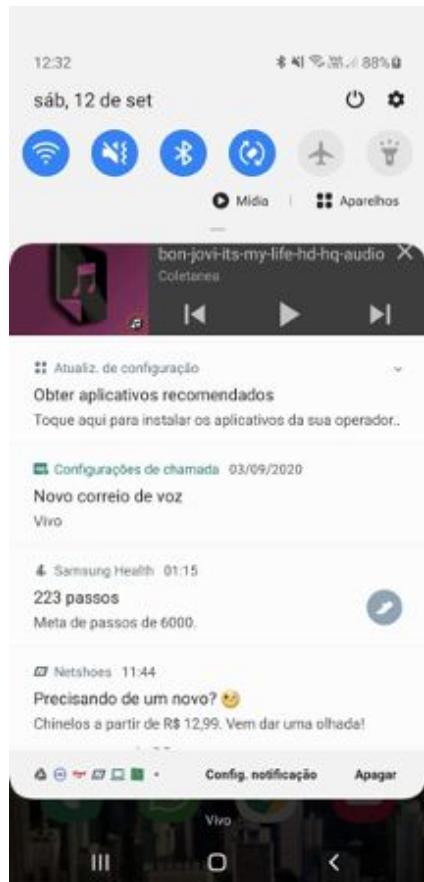
O Android possui uma API de notificações para exibir as mensagens que podem ser criadas internamente ou disparadas via FCM. Essas notificações permitem alertar o usuário sobre algo que aconteceu mesmo que ele não esteja utilizando a aplicação ou até mesmo o aparelho, por exemplo, você pode não estar olhando o device mas receber um alerta referente ao status do seu pedido, uma mensagem que chegou, entre outros.

Quando uma notificação é disparada, ela fica visível na barra de status do aparelho podendo ser visualizada ao deslizá-la.



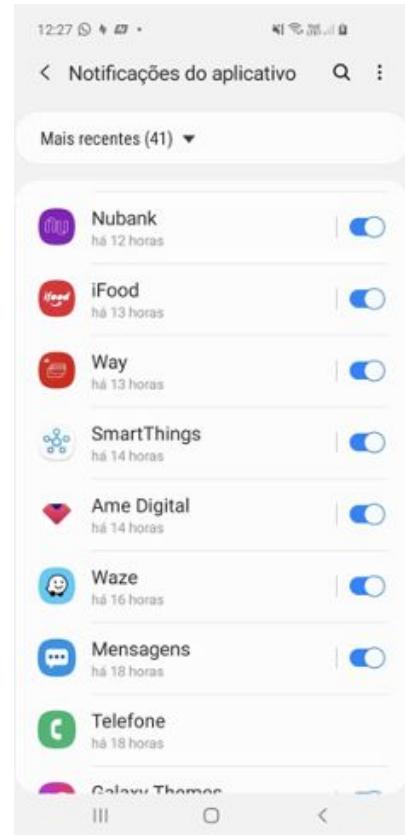
NOTIFICAÇÃO NO ANDROID

Exemplo de notificação ao deslizar a barra de notificação:



NOTIFICAÇÃO NO ANDROID

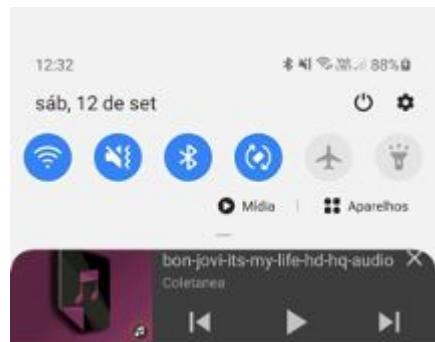
O uso das notificações pode aumentar o engajamento do usuário, porém, cuidado ao utilizá-las para não bombardear o usuário a ponto de incomodá-lo e ele desligar as notificações do seu aplicativo. Desde a versão Android 5 (Lollipop) o usuário tem como desativar as notificações nas configurações do aparelho. As configurações podem ser diferentes de acordo com a versão do Android ou fabricante do aparelho, porém, nesta tela é possível ver uma lista dos aplicativos instalados no aparelho e se a notificação está ativa ou não:



NOTIFICAÇÃO NO ANDROID

As notificações estão presentes no Android desde o início, porém, devido a sua importância ela deixou de ser apenas o básico presente até a versão 2.3 que era um título, detalhe, ícone e hora. Para que haja a compatibilidade com todas as versões do sistema operacional, deve-se utilizar a classe `NotificationCompat.Builder` da biblioteca de compatibilidade.

Através das notificações é possível disparar uma activity ou uma tarefa específica. Para isso, deverá ser criado um objeto da classe `PendingIntent`, que será disparado pelo usuário a partir da notificação. Como pode ser observado na figura abaixo, a notificação pode além de exibir mensagem, controlar o aplicativo, no exemplo, tocar/pausar, avançar/voltar a música.

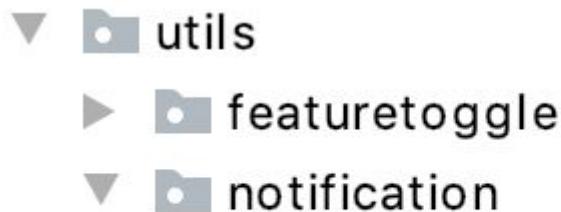


CRIANDO NOSSA CLASSE DE NOTIFICAÇÕES



NOTIFICAÇÃO NO ANDROID

Dentro do pacote **presentation/utils** crie um pacote chamado **notification** e dentro dele uma classe chamada **NotificationUtils**



NOTIFICAÇÃO NO ANDROID

```
object NotificationUtils {  
  
    data class NotifChannel (val id: String, val name: String, val description: String)  
  
    @RequiresApi (Build.VERSION_CODES.O)  
    private fun createNotificationChannel (context: Context, notifChannel: NotifChannel) {  
  
        val notificationManager =  
            context.getSystemService( Context.NOTIFICATION_SERVICE) as NotificationManager  
  
        val channel = NotificationChannel(  
            notifChannel.id,  
            notifChannel.name,  
            NotificationManager.IMPORTANCE_DEFAULT  
)  
        .apply {  
            description = notifChannel.description  
            enableLights(true)  
            enableVibration(true)  
            vibrationPattern = longArrayOf(100, 200, 300, 400, 500, 400, 300, 200, 400)  
        }  
        notificationManager.createNotificationChannel(channel)  
    }  
}
```



NOTIFICAÇÃO NO ANDROID

```
fun notificationSimple(context: Context, title: String, message: String, notifChannel: NotifChannel) {  
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {  
        createNotificationChannel(context, notifChannel)  
    }  
    val notificationBuilder = NotificationCompat.Builder(  
        context,  
        notifChannel.id  
    )  
        .setSmallIcon(R.mipmap.ic_launcher)  
        .setContentTitle(title)  
        .setContentText(message)  
        .setPriority(NotificationCompat.PRIORITY_DEFAULT)  
        .setAutoCancel(true)  
        .setColor(  
            ActivityCompat.getColor(  
                context,  
                R.color.colorAccent  
            )  
        )  
        .setDefaults(Notification.DEFAULT_ALL)  
    val notificationManager = NotificationManagerCompat.from(context)  
    notificationManager.notify(1, notificationBuilder.build())  
}
```



CANAIS DE NOTIFICAÇÃO

Introduzido no Android Oreo (api 26) é possível categorizar e agrupar as notificações do aplicativo, e desta forma, em vez do usuário desabilitar todas as notificações do aplicativo, ele pode desabilitar alguns tipos de notificações. Por exemplo, em um aplicativo de e-commerce poderia haver um canal de pedidos e um de promoções, onde o usuário poderia desligar o de promoções caso não queira receber este tipo de mensagem, porém continuaria recebendo as referente aos status do pedido.

O Android Oreo também trouxe o recurso que ao deslizar uma notificação para um dos lados, um ícone de engrenagem é exibido. Ao clicar neste ícone é possível visualizar o canal que essa notificação faz parte. Em algumas versões será possível silenciar a notificação fazendo o mesmo movimento, clicando no ícone do sino que aparece.



CANAIS DE NOTIFICAÇÃO



CANAIS DE NOTIFICAÇÃO

No código do tópico anterior foi criado um canal através do seguinte método:

```
@RequiresApi (Build.VERSION_CODES.O)
private fun createNotificationChannel(context: Context, notifChannel: NotifChannel) {

    val notificationManager =
        context.getSystemService( Context.NOTIFICATION_SERVICE) as NotificationManager

    val channel = NotificationChannel(
        notifChannel.id,
        notifChannel.name,
        NotificationManager.IMPORTANCE_DEFAULT
    ).apply {
        description = notifChannel.description
        enableLights( true)
        enableVibration( true)
        vibrationPattern = longArrayOf(100, 200, 300, 400, 500, 400, 300, 200, 400)
    }
    notificationManager.createNotificationChannel( channel)
}
```

CANAIS DE NOTIFICAÇÃO

O método anterior só será chamado se a aplicação estiver sendo executada em aparelhos com o Android Oreo (API 26) ou superior. Neste método será criado o canal por meio da classe **NotificationChannel** através dos parâmetros id, nome e importância.

NÍVEIS DE IMPORTÂNCIA DE UM CANAL DE NOTIFICAÇÃO

Importância	Propósito
PRIORITY_MIN	Prioridade mínima. O sistema poderá exibir essa notificação menor ou no fim da lista de notificações.
PRIORITY_LOW	Prioridade baixa. O sistema poderá exibir essa notificação menor ou abaixo das notificações com prioridade padrão.
PRIORITY_DEFAULT	Prioridade padrão.
PRIORITY_HIGH	Prioridade alta. O sistema poderá exibir essa notificação maior ou acima das notificações com prioridade padrão.
PRIORITY_MAX	Prioridade máxima. O sistema poderá exibir essa notificação maior ou no topo da lista de notificações.

CANAIS DE NOTIFICAÇÃO

A classe **NotificationManager** fornece métodos para gerenciar os canais de notificação, onde é possível obter os canais do aplicativo por meio dos métodos:

getNotificationChannel (String): você deve informar o id do canal que deseja obter

getNotificationChannels(): retorna a lista de todos os canais.

Para excluir um canal, basta invocar o método **deleteNotificationChannel(String)**, onde o parâmetro enviado é o canal que deseja apagar.

VISIBILIDADE DAS NOTIFICAÇÕES

A partir do Android 5 (Lollipop) as notificações por padrão também aparecem na tela de bloqueio. Dependendo da configuração ou da versão do Android, o conteúdo da notificação pode não ser exibido, mas isso pode ser configurado pelo método `setVisibility(int)`, que pode assumir os valores os seguintes valores:

VISIBILITY_PUBLIC: a notificação e o seu conteúdo são exibidos na tela de bloqueio.

VISIBILITY_PRIVATE (padrão): exibe a notificação na lockscreen, mas não o seu conteúdo.

VISIBILITY_SECRET: a notificação não aparecerá na tela de bloqueio.

VISIBILIDADE DAS NOTIFICAÇÕES

Exemplo de utilização:

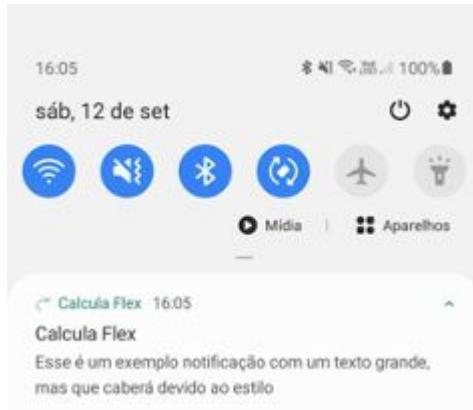
```
val notificationBuilder = NotificationCompat.Builder(  
    context,  
    notifChannel.id  
) .setVisibility(NotificationCompat.VISIBILITY_PRIVATE)
```

TIPOS DE NOTIFICAÇÕES



NOTIFICAÇÃO COM TEXTO LONGO

Muito comum em aplicativos de redes sociais, e-mail ou troca de mensagens, pois possuem grande quantidade de texto. Por padrão, as notificações são exibidas em uma linha. Para exibir em mais existe o BigText.

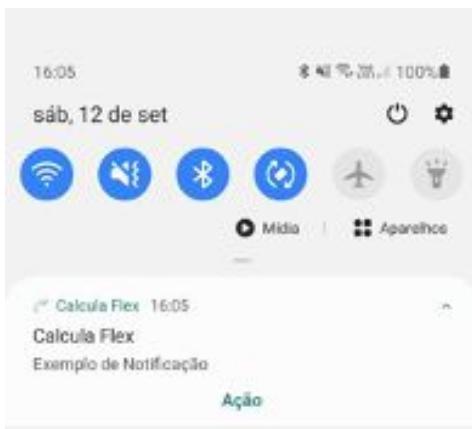


NOTIFICAÇÃO COM TEXTO LONGO

```
fun notificationBigText (context: Context) {  
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {  
        createNotificationChannel(context)  
    }  
    val bigTextStyle = NotificationCompat.BigTextStyle()  
        .bigText(context.getString( R.string.notif_big_message ))  
    val notificationBuilder = NotificationCompat.Builder(  
        context,  
        CHANNEL_ID  
    )  
        .setSmallIcon( R.drawable.ic_favorite)  
        .setContentTitle(context.getString( R.string.notif_title ))  
        .setPriority( NotificationCompat.PRIORITY_DEFAULT)  
        .setColor( ActivityCompat.getColor(context, R.color.colorAccent))  
        .setDefaults( Notification.DEFAULT_ALL)  
        .setContentIntent(getContentIntent(context))  
        .setAutoCancel( true)  
        .setStyle( bigTextStyle )  
    val notificationManager = NotificationManagerCompat.from(context)  
    notificationManager.notify( 3, notificationBuilder.build())  
}
```

NOTIFICAÇÃO COM BOTÕES DE AÇÃO

Os botões de ação normalmente realizam alguma ação em background. Por exemplo, tocar/pausar a música, responder um e-mail ou qualquer outra ação de acordo com o aplicativo.

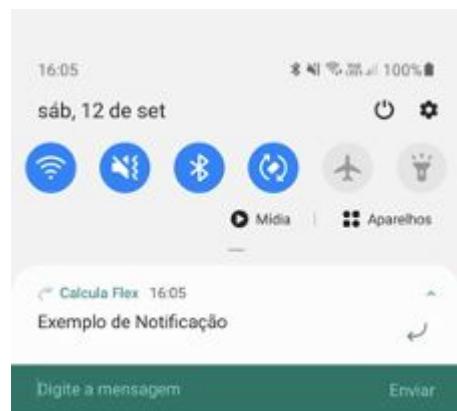
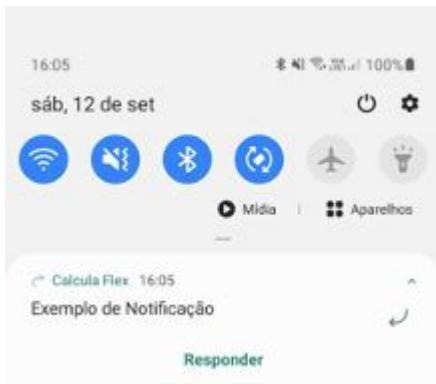


NOTIFICAÇÃO COM BOTÕES DE AÇÃO

```
fun notificationWithButtonAction (context: Context) {  
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {  
        createNotificationChannel(context)  
    }  
    val actionIntent = Intent(context, NotificationActionReceiver::class.java).apply {  
        putExtra(NotificationActionReceiver.EXTRA_MESSAGE, "Ação da notificação")  
    }  
    val pendingIntent = PendingIntent.getBroadcast(context, 0, actionIntent, 0)  
    val notificationBuilder = NotificationCompat.Builder(  
        context,  
        CHANNEL_ID  
    ).setSmallIcon(R.drawable.ic_favorite)  
        .setContentTitle(context.getString(R.string.notif_title))  
        .setContentText(context.getString(R.string.notif_text))  
        .setPriority(NotificationCompat.PRIORITY_DEFAULT)  
        .setColor(ActivityCompat.getColor(context, R.color.colorAccent))  
        .setDefaults(Notification.DEFAULT_ALL)  
        .addAction(0, context.getString(R.string.notif_button_action), pendingIntent)  
        .setAutoCancel(true)  
    val notificationManager = NotificationManagerCompat.from(context)  
    notificationManager.notify(4, notificationBuilder.build())  
}
```

NOTIFICAÇÃO COM DIGITAÇÃO DE TEXTO

A partir do Android 8 é possível responder uma notificação com texto. Ele é adicionado a uma action. Quando a action é clicada, ela se transforma em uma área na qual o usuário poderá digitar a mensagem que desejar. Ao enviar a mensagem, a Intent definida é disparada.



NOTIFICAÇÃO COM DIGITAÇÃO DE TEXTO

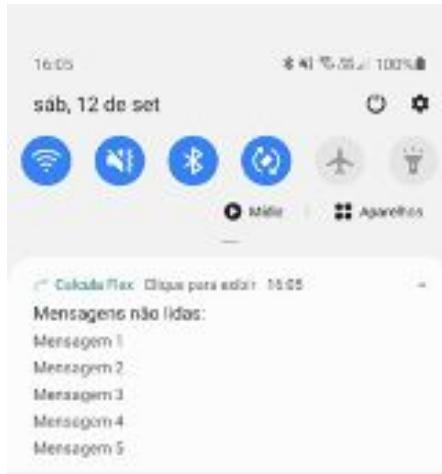
```
fun notificationReplied(context: Context, notificationId: Int) {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        createNotificationChannel(context)
    }
    val timeout = 2000L

    val notificationBuilder = NotificationCompat.Builder(context, NotificationUtils.CHANNEL_ID)
        .setSmallIcon(R.drawable.ic_favorite)
        .setContentTitle(context.getString(R.string.notif_title))
        .setContentText(context.getString(R.string.notif_reply_replied))
        .setColor(
            ContextCompat.getColor(
                context, R.color.colorAccent
            )
        )
        .setPriority(NotificationCompat.PRIORITY_DEFAULT)
        .setDefaults(0)
        .setTimeoutAfter(timeout)

    val notificationManager = NotificationManagerCompat.from(context)
    notificationManager.notify(notificationId, notificationBuilder.build())
    if (Build.VERSION.SDK_INT < Build.VERSION_CODES.O) {
        Handler().postDelayed{ notificationManager.cancel(notificationId) }, timeout
    }
}
```

NOTIFICAÇÕES AGRUPADAS

Recurso utilizado para agrupar mensagens do mesmo tipo para evitar poluir a barra de notificações do aparelho com o mesmo tipo de mensagens. Por exemplo, os aplicativos de comida que agrupam as mensagens do status do pedido.



NOTIFICAÇÕES AGRUPADAS

```
fun notificationInbox(context: Context) {  
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {  
        createNotificationChannel(context)  
    }  
    val number = 5  
    val inboxStyle = NotificationCompat.InboxStyle()  
  
    inboxStyle.setBigContentTitle(context.getString(R.string.notif_big_inbox_title))  
    for (i in 1..number) {  
        inboxStyle.addLine(context.getString(R.string.notif_big_inbox_message, i))  
    }  
}
```

NOTIFICAÇÕES AGRUPADAS

```
inboxStyle.setSummaryText(context.getString(R.string.notif_big_inbox_summary))
```



```
val notificationBuilder = NotificationCompat.Builder(context,
CHANNEL_ID)
    .setSmallIcon(R.drawable.ic_favorite)
    .setColor(ActivityCompat.getColor(context, R.color.colorAccent))
    .setContentTitle(context.getString(R.string.notif_title))
    .setContentText(context.getString(R.string.notif_text))
    .setDefaults(Notification.DEFAULT_ALL)
    .setNumber(number)
    ..setStyle(inboxStyle)
```



```
val nm = NotificationManagerCompat.from(context)
nm.notify(8, notificationBuilder.build())
}
```

PROJETO DE NOTIFICAÇÕES

No repositório abaixo tem um projeto de exemplo de notificações:

<https://github.com/heiderlopes/Notification>

IMPLEMENTANDO

FIREBASE CLOUD MESSAGE (FCM)



IMPORTANTE:

Na raíz do projeto crie um package chamado **fcm**. Dentro do pacote criado adicione uma classe chamada **CalculaFlexFCMService**.

Este classe que estende **FirebaseMessagingService**. Isso é necessário se você quiser processar qualquer mensagem além de simplesmente receber notificações em apps em segundo plano. Para receber notificações em apps em primeiro plano ou payload de dados, enviar mensagens upstream e assim por diante, você precisa estender esse serviço.

```
class CalculaFlexFCMService : FirebaseMessagingService() {  
  
    override fun onMessageReceived(p0: RemoteMessage) {  
        super.onMessageReceived(p0)  
    }  
  
    override fun onNewToken(p0: String) {  
        super.onNewToken(p0)  
    }  
}
```

FCM Service

Abra o arquivo **build.gradle** do **app** e adicione:

```
implementation 'com.google.firebaseio:messaging-ktx'
```

FIREBASE CLOUD MESSAGING

Declare esse serviço no **AndroidManifest.xml**:

```
<service
    android:name=".fcm.CalculaFlexFCMService"
    android:exported="false">
    <intent-filter>
        <action android:name="com.google.firebaseio.MESSAGING_EVENT" />
    </intent-filter>
</service>
```

FIREBASE CLOUD MESSAGING

(Opcional) Elementos de metadados dentro do componente do aplicativo para definir um ícone e uma cor padrão para notificações. O Android usa esses valores sempre que mensagens recebidas não definem explicitamente o ícone ou a cor do ícone:

```
<meta-data  
    android:name="com.google.firebaseio.messaging.default_notification_icon"  
    android:resource="@drawable/ic_launcher_foreground"/>  
  
<meta-data  
    android:name="com.google.firebaseio.messaging.default_notification_color"  
    android:resource="@color/colorAccent" />
```

FIREBASE CLOUD MESSAGING

(Opcional) A partir do Android 8.0 (API nível 26) e posterior, os canais de notificação são aceitos e recomendados. O **FCM** oferece um canal de notificação padrão com configurações básicas. Se você preferir criar e usar seu próprio canal padrão, defina **default_notification_channel_id** como o ID do objeto do canal de notificação, conforme mostrado. O FCM usará esse valor sempre que as mensagens recebidas não definirem explicitamente um canal de notificação.

```
<meta-data  
    android:name="com.google.firebaseio.messaging.default_notification_channel_id"  
    android:value="@string/default_notification_channel_id"/>
```

FIREBASE CLOUD MESSAGING

Abra o arquivo **strings.xml** e adicione a seguinte string:

```
<string name="default_notification_channel_id">fcm_default_channel</string>
<string name="default_notification_channel_name">Canal de Avisos</string>
<string name="default_notification_channel_description">Canal de avisos do
aplicativo Calcula Flex</string>
```

CALCULA FLEX FCM SERVICE

```
class CalculaFlexFCMService : FirebaseMessagingService() {

    override fun onMessageReceived(p0: RemoteMessage) {

        val title = p0.data["title"] ?: p0.notification?.title ?: this.getString(R.string.app_name)
        val message = p0.data["message"] ?: p0.notification?.body ?: ""

        val channelID = this.getString(R.string.default_notification_channel_id)
        val channelName = this.getString(R.string.default_notification_channel_name)
        val channelDescription =
            this.getString(R.string.default_notification_channel_description)

        val notifChannel =
            NotificationUtils.NotifChannel(channelID, channelName, channelDescription)

        NotificationUtils.notificationSimple(this, title, message, notifChannel)
    }

    override fun onNewToken(p0: String) {
        super.onNewToken(p0)
    }
}
```

ESTUDOS COMPLEMENTARES

<https://medium.com/exploring-android/exploring-android-o-notification-channels-94cd274f604c>

<https://code.tutsplus.com/tutorials/android-o-how-to-use-notification-channels--cms-28616>

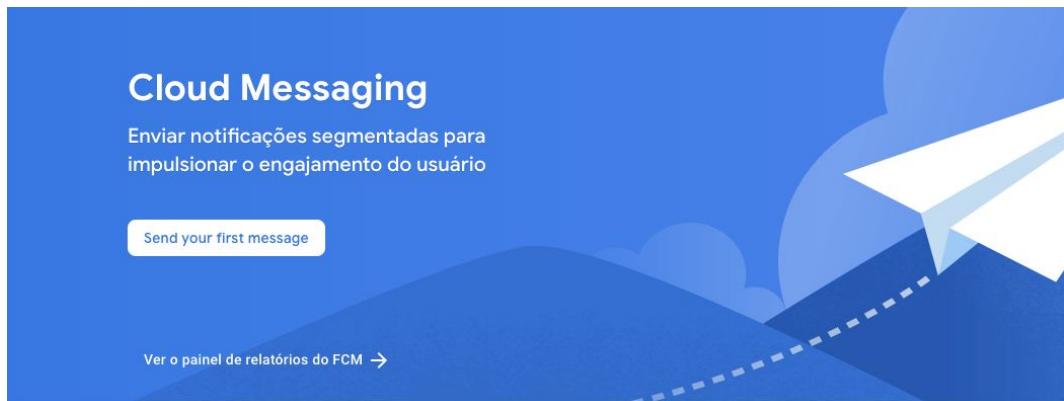
<https://www.youtube.com/watch?v=hyGxr2V0yel>



TESTANDO VIA FIREBASE CLOUD MESSAGING: DISPARANDO MENSAGEM VIA PAINEL DO FIREBASE

DISPARANDO NOTIFICAÇÃO

Clique sobre **Send your first message**



DISPARANDO NOTIFICAÇÃO

Preencha os campos que serão enviados na notificação, onde:

Título da notificação: Mostrado aos usuários finais como título da notificação

Texto da notificação: Mensagem a ser exibida para o usuário.

Imagen da notificação (opcional): Como alternativa, faça upload de uma imagem ou forneça um URL de imagem HTTPS válido.

Nome da notificação (opcional): Nome usado para identificar esta notificação no Firebase. Esse nome não é exibido aos usuários.

DISPARANDO NOTIFICAÇÃO

1 Notificação

Título da notificação ⑦

Calcula Flex

Texto da notificação

Você foi sorteado para ganhar um abastecimento gratuito no seu veículo.

Imagen de notificação (opcional) ⑦

Exemplo: <https://yourapp.com/image.png>

Nome da notificação (opcional) ⑦

Informa nome opcional

Enviar mensagem de teste

Estado inicial Visualização expandida

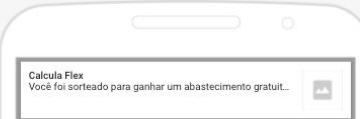
Android

iOS

Próxima

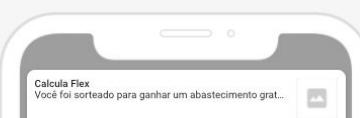
Visualização do dispositivo

Por meio dessa visualização, é possível ter uma ideia geral de como sua mensagem será exibida em um dispositivo móvel. Haverá variação na renderização real da mensagem dependendo do dispositivo. Faça o teste com um dispositivo real para resultados reais.



Calcula Flex
Você foi sorteado para ganhar um abastecimento gratuito...

Android



Calcula Flex
Você foi sorteado para ganhar um abastecimento gratuito...

iOS

DISPARANDO NOTIFICAÇÃO

Após preencher os campos da notificação, clique em **Próxima** para preencher a segmentação.

Neste ponto, você poderá selecionar a qual público se destina esta notificação.

No exemplo abaixo, a notificação será enviada para todos os usuários da aplicação.

Notificação
Você foi sorteado para ganhar um abastecimento gratuito no seu veículo.

2 Segmentação

Segmento do usuário Tópico

Segmentar usuário se...

App br.com.calculaflex

Segmentar outro app

100% dos usuários em potencial estão qualificados para esta campanha: 5 ⓘ

Próxima

DISPARANDO NOTIFICAÇÃO

Após definir a segmentação o próximo passo é definir o momento do envio.

Você pode enviar imediatamente ou agendar.

3

Programação

Enviar para usuários qualificados

Agora

Próxima

DISPARANDO NOTIFICAÇÃO

Métricas que serão utilizadas:

4

Eventos de conversão (opcional)

Enviadas



Abertas



Selecionar métrica de objetivo

[Próxima](#)

DISPARANDO NOTIFICAÇÃO

Opcionalmente você pode preencher os campos:

Canal de notificações do Android: Código do canal de notificação do Android

Dados personalizados: Pares de chave/valor que serão entregues com a mensagem para o seu aplicativo. Neste campo você pode definir por exemplo: o id de um produto que deverá ser exibido na tela de detalhes.

Vencimento: Por quanto tempo a mensagem deve ser mantida para reenvio. O tempo de validade máximo é de quatro semanas a partir da primeira tentativa de envio.

DISPARANDO NOTIFICAÇÃO

5

Outras opções (opcional)

Todos os campos são opcionais

Canal de notificações do Android [?](#)

Dados personalizados [?](#)

 Chave Valor

Som

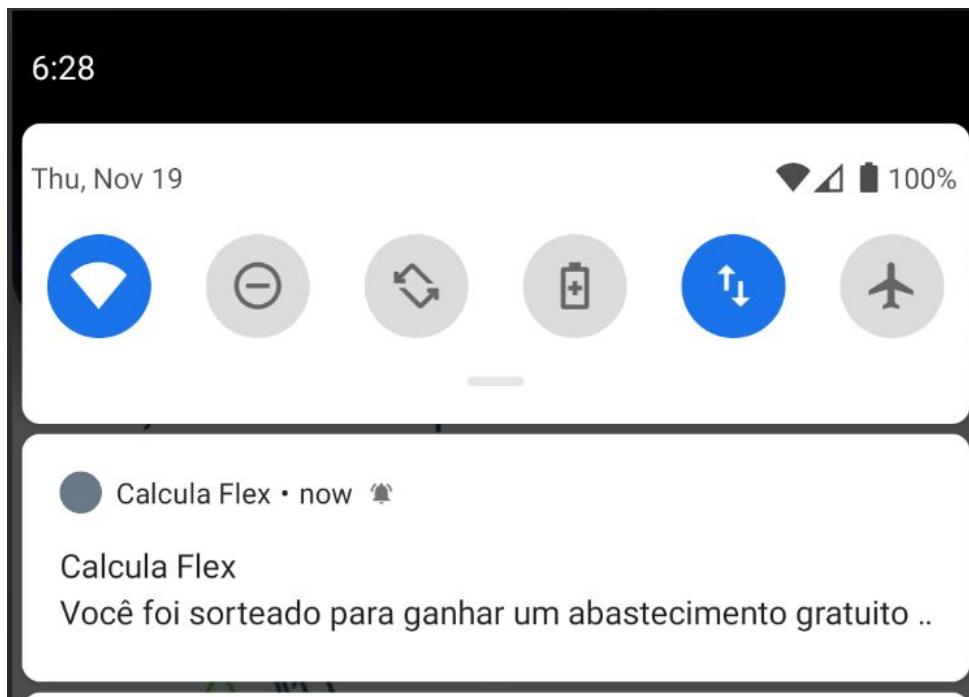
 Desativado ▾

Vencimento [?](#)

 4 ▾ semana ▾

DISPARANDO NOTIFICAÇÃO

Após o preenchimento, clique em **Revisar**, confira os dados e clique em **Publicar**.



DISPARANDO NOTIFICAÇÃO

Após o preenchimento, clique em **Revisar**, confira os dados e clique em **Publicar**.

Revisar mensagem

Conteúdo da notificação



Você foi sorteado para ganhar um abastecimento gratuito no seu veículo.

Destino



Segmentação de usuário correspondente a um critério de segmentação

Agendamento



Enviar agora

Cancelar

Publicar

ESTUDOS COMPLEMENTARES

<https://medium.com/exploring-android/exploring-android-o-notification-channels-94cd274f604c>

<https://code.tutsplus.com/tutorials/android-o-how-to-use-notification-channels--cms-28616>

<https://www.youtube.com/watch?v=hyGxr2V0yel>

TESTANDO VIA POSTMAN:

FIREBASE CLOUD MESSAGE (FCM)



DISPARANDO NOTIFICAÇÃO

Através do Postman é possível realizar a chamada da API rest. Desta mesma forma é possível integrar seu backend com o Firebase e disparar uma notificação para o usuário diretamente do seu backend.

No método **onNewToken** da classe **CalculaFlexFCMService** você terá o token sempre que for atualizado. Neste ponto, seria importante você enviar esse dado para o servidor da sua aplicação para quando houver a necessidade de enviar alguma mensagem para seu usuário você terá essa informação disponível.

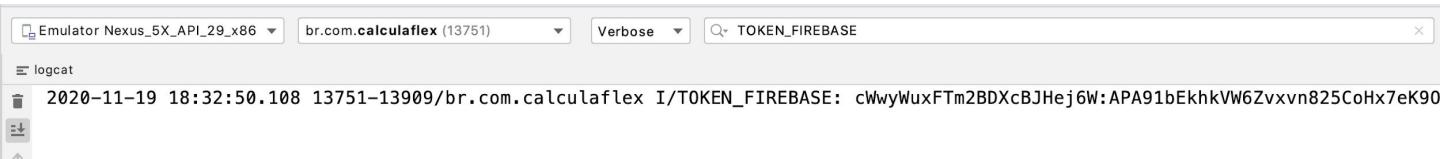
Abra a classe **CalculaFlexFCMService** e adicione um log para vermos o token gerado:

```
override fun onNewToken(p0: String) {  
    super.onNewToken(p0)  
    Log.i("TOKEN_FIREBASE", p0)  
}
```

DISPARANDO NOTIFICAÇÃO

Observação: caso não entre neste método você já obteve o token na primeira instalação. Por hora, remova o aplicativo e execute a instalação novamente.

O resultado deverá ser algo parecido com a seguinte imagem:



The screenshot shows the Android Logcat interface. At the top, there are dropdown menus for 'Emulator Nexus_5X_API_29_x86' (selected), 'br.com.calculaflex (13751)' (selected), 'Verbose' (selected), and a search bar containing 'TOKEN_FIREBASE'. Below the header, the word 'logcat' is preceded by a small icon. The main area displays a single log entry: '2020-11-19 18:32:50.108 13751-13909/br.com.calculaflex I/TOKEN_FIREBASE: cWwyWuxFTm2BDXcBJHej6W:APA91bEkhkVW6Zvxvn825CoHx7eK90'. On the far left of the log entry, there is a small trash bin icon.

DISPARANDO NOTIFICAÇÃO

Clique sobre a opção **Cloud Messaging** e copie a chave do Servidor.

Configurações

Geral **Cloud Messaging** Integrações Contas de serviço Privacidade dos dados Usuários e permissões

Credenciais do projeto

Chave	Token	Adicionar chave do servidor
Chave do servidor	AAAAA0I19gvE qDNrF3mZQP	
ID do remetente	?	
	895727010545	

DISPARANDO NOTIFICAÇÃO

Abra o Postman altere o método para **Post**, em seguida, adicione a url da api do Firebase

https://fcm.googleapis.com/fcm/send.

Clique sobre a aba **Headers** e adicione o **Authorization** (com a chave do servidor) e o Content-Type para application/json.

The screenshot shows the Postman interface with a POST request to `https://fcm.googleapis.com/fcm/send`. The Headers tab is selected, showing two entries: `Authorization` with value `key=AAAA0I19gvE:APA91bFSyhRFp8Qs7lkI7eQE98HVI1zukjLCD1lbWeNw7kLX55KuC` and `Content-Type` with value `application/json`.

KEY	VALUE	DESCRIF
Authorization	key= AAAA0I19gvE:APA91bFSyhRFp8Qs7lkI7eQE98HVI1zukjLCD1lbWeNw7kLX55KuC	
Content-Type	application/json	
Key	Value	Descriç

DISPARANDO NOTIFICAÇÃO

Clique agora sobre a aba **Body** e adicione o objeto que será enviado para o usuário:

```
{  
  "to": "COLOQUE_AQUI_O_USER_DEVICE_TOKEN",  
  "data": {  
    "title": "Calcula Flex",  
    "message": "Você foi sorteado e ganhou um carro 0KM"  
  }  
}
```

FIREBASE CLOUD MESSAGING

Abra o Postman e adicione a seguinte URL:

https://fcm.googleapis.com/fcm/send

Headers 9 hidden

KEY	VALUE	DESCRIPTION	***	Bulk Edit	Presets ▾
<input checked="" type="checkbox"/> Authorization	key=AAAAAxB6Gfdw:APA91bEk4yNSGydJndG2CXVWm_8N153VcGDHE7Dpr ...				
<input checked="" type="checkbox"/> Content-Type	application/json				
Key	Value	Description			

```
{  
  "to": "USER_DEVICE_TOKEN",  
  "data": {  
    "deeplink": "https://www.calculaflex.com.br/betterfuel",  
    "title": "Titulo da mensagem",  
    "message": "Exemplo de mensagem"  
  }  
}
```

ABRINDO DEEPLINK: VIA PUSH NOTIFICATION



FIREBASE CLOUD MESSAGING

Altere body para :

```
{  
  "to": "COLOQUE_AQUI_O_USER_DEVICE_TOKEN",  
  "data": {  
    "deeplink": "https://www.calculaflex.com.br/signup",  
    "title": "Calcula Flex",  
    "message": "Crie uma conta e ganhe descontos"  
  }  
}
```

FIREBASE CLOUD MESSAGING

Abra o **NotificationUtils** e receba a intent que deverá ser executada via parâmetro e adicione no builder da notificação:

```
fun notificationSimple(  
    context: Context,  
    title: String,  
    message: String,  
    notifChannel: NotifChannel,  
    intent: Intent  
) {  
    val pendingIntent = PendingIntent.getActivity(  
        context,  
        0, intent, PendingIntent.FLAG_ONE_SHOT  
    )  
  
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {  
        createNotificationChannel(context, notifChannel)  
    }  
}
```

FIREBASE CLOUD MESSAGING

```
val notificationBuilder = NotificationCompat.Builder(  
    context,  
    notifChannel.id  
)  
    .setContentIntent(pendingIntent)  
    .setVisibility(NotificationCompat.VISIBILITY_PRIVATE)  
    .setSmallIcon(R.drawable.ic_logo_notification)  
    .setContentTitle(title)  
    .setContentText(message)  
    .setPriority(NotificationCompat.PRIORITY_DEFAULT)  
    .setAutoCancel(true)  
    .setColor(  
        ActivityCompat.getColor(  
            context,  
            R.color.colorAccent  
)  
    )  
    .setDefaults(Notification.DEFAULT_ALL)  
val notificationManager = NotificationManagerCompat.from(context)  
notificationManager.notify(1, notificationBuilder.build())  
}
```

FIREBASE CLOUD MESSAGING

Feito isso, adicione a verificação na classe **CalculaFlexFCMService**, conforme o código abaixo:

```
override fun onMessageReceived(p0: RemoteMessage) {  
    val isDeepLink = p0.data.containsKey("deeplink")  
  
    val intent =  
        if (isDeepLink) Intent(Intent.ACTION_VIEW,  
            Uri.parse(p0.data["deeplink"])) else Intent(  
                this,  
                MainActivity::class.java  
        )
```

FIREBASE CLOUD MESSAGING

```
val title = p0.data["title"] ?: p0.notification?.title ?:  
this.getString(R.string.app_name)  
  
val message = p0.data["message"] ?: p0.notification?.body ?: ""  
  
  
val channelID = this.getString(R.string.default_notification_channel_id)  
val channelName = this.getString(R.string.default_notification_channel_name)  
val channelDescription =  
this.getString(R.string.default_notification_channel_description)  
  
  
val notifChannel =  
NotificationUtils.NotifChannel(channelID, channelName, channelDescription)  
  
  
NotificationUtils.notificationSimple(this, title, message, notifChannel, intent)  
}
```

EXERCÍCIO:

GRAVANDO O TOKEN DO USUÁRIO



ADICIONANDO O CRASHLYTICS: SUA FERRAMENTA DE ANÁLISE DE CRASH



FIREBASE CRASHLYTICS

O **Firebase Crashlytics** ajuda a monitorar, priorizar e resolver problemas de estabilidade em tempo real. Através dele conseguimos monitorar a saúde do nosso aplicativo.

Segue a documentação oficial para adicionar no seu projeto:

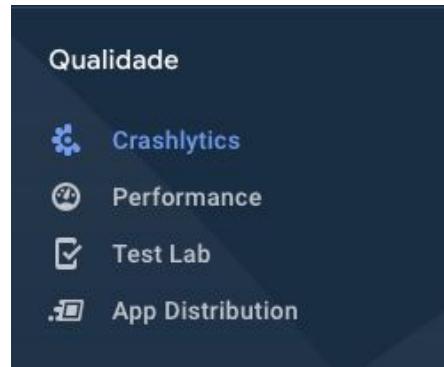
<https://firebase.google.com/docs/crashlytics/get-started?hl=pt-br&platform=android#kotlin+ktx>



FIREBASE CRASHLYTICS

Para adicionar no projeto, adicione no seu arquivo **build.gradle (nível projeto)**

1. Clique em **Crashlytics** no painel de navegação à esquerda do Console do Firebase.
1. Se o projeto do Firebase tiver vários apps registrados nele, selecione aquele que você acabou de adicionar no menu suspenso ao lado da opção Crashlytics na barra superior do Console.
1. Clique em **Ativar Crashlytics**.



FIREBASE CRASHLYTICS

No arquivo **build.gradle** no **nível do projeto**, adicione o plug-in do Gradle para Crashlytics como uma dependência do buildscript.

```
dependencies {  
    // ...  
    // Check that you have the Google Services Gradle plugin v4.3.2 or later  
    // (if not, add it).  
    classpath 'com.google.gms:google-services:4.3.4'  
  
    // Add the Crashlytics Gradle plugin (be sure to add version  
    // 2.0.0 or later if you built your app with Android Studio 4.1).  
    classpath 'com.google.firebaseio:firebase-crashlytics-gradle:2.3.0'  
}  
}
```

FIREBASE CRASHLYTICS

No arquivo **build.gradle** no **nível do app**, aplique o plug-in do Gradle para Crashlytics:

```
plugins {
    id 'com.android.application'
    id 'kotlin-android'
    id 'com.google.gms.google-services'
    id 'com.google.firebaseio.crashlytics'
}

// ...
```

FIREBASE CRASHLYTICS

Usando a **BoM do Firebase para Android**, declare a dependência da biblioteca Android do Crashlytics no seu arquivo do Gradle (nível do app) do módulo, que geralmente é **app/build.gradle**.

```
implementation 'com.google.firebaseio:firebase-crashlytics-ktx'
```

O QUE É O BOM?

O **Firebase Android BoM** permite que você gerencie todas as suas versões da biblioteca Firebase especificando apenas uma versão - **a versão do BoM**.

Quando você usa o Firebase BoM em seu aplicativo, o BoM puxa automaticamente as versões individuais da biblioteca mapeadas para a versão do BoM. Todas as versões de bibliotecas individuais serão compatíveis. Quando você atualiza a versão do BoM em seu aplicativo, todas as bibliotecas do Firebase que você usa em seu aplicativo são atualizadas para as versões mapeadas para essa versão do BoM.

Para saber quais versões da biblioteca do Firebase são mapeadas para uma versão específica do BoM, verifique as notas de lançamento dessa versão do BoM. Se você precisar comparar as versões da biblioteca mapeadas para uma versão do BoM em comparação com outra versão do BoM, use o widget de comparação abaixo.

Saiba mais sobre o suporte do Gradle para plataformas BoM .

Veja como usar o Firebase Android BoM para declarar dependências no seu arquivo Gradle do módulo (nível do aplicativo) (geralmente app/build.gradle). Ao usar o BoM, você não especifica versões de biblioteca individuais nas linhas de dependência.

O QUE É O BOM?

Veja como usar o Firebase Android BoM para declarar dependências no seu arquivo Gradle do módulo (nível do aplicativo) (geralmente app/build.gradle). Ao usar o BoM, você não especifica versões de biblioteca individuais nas linhas de dependência.

```
dependencies {  
    // Import the BoM for the Firebase platform  
    implementation platform('com.google.firebaseio:firebase-bom:26.1.0')  
  
    // Declare the dependencies for the desired Firebase products without  
    specifying versions  
    // For example, declare the dependencies for Firebase Authentication and  
    Cloud Firestore  
    implementation 'com.google.firebaseio:firebase-auth'  
    implementation 'com.google.firebaseio:firebase-firebase'  
}
```

FIREBASE CRASHLYTICS

Após a configuração **Execute o seu projeto.**

SIMULANDO UMA EXCEÇÃO

Para adicionar um botão e simular o crash adicione:

```
throw RuntimeException("Test Crash") // Force a crash
```

Ativar a geração de registros de depuração do Crashlytics

Se a tentativa não causar falha, se a falha ocorrer antes do esperado ou se você estiver enfrentando algum outro problema com o Crashlytics, ative a geração de registros de depuração para rastrear o problema.

Para ativar a geração de registros de depuração no seu dispositivo de desenvolvimento, defina uma sinalização de shell adb antes de executar o app:

```
adb shell setprop log.tag.FirebaseCrashlytics DEBUG
```

Para ver os registros nos registros do dispositivo, execute:

```
adb logcat -s FirebaseCrashlytics
```

Para desativar a geração de registros de depuração, defina a sinalização novamente como INFO:

```
adb shell setprop log.tag.FirebaseCrashlytics INFO
```

EXEMPLO DE RELATÓRIO DE BUGS



EXERCÍCIO

Resultado do relatório:

Calcula Flex ▾ Acessar a documentação 📲 H ?

Crashlytics

Filtros Tipo de evento = "Falhas" ✕

Últimos sete dias De 14 de nov. a 20 de nov.

Estatísticas sem falhas

Usuários que não tiveram falhas ⓘ 66,67%

Data	Retenção (%)
Nov 14	100%
Nov 17	100%
Nov 18	~75%
Nov 20	100%

Tendências do evento

Falhas Usuários afetados

1 1

Data	Falhas	Usuários afetados
Nov 14	1	1
Nov 16	1	1
Nov 18	1	1
Nov 20	1	1

Problemas

Filtrar problemas Estado do problema = "Abertos" ✕

Pesquisar por código de usuário

Problemas	Detalhes	Versões	Eventos	Usuários
HomeFragment.kt – line 132 br.com.calculaflex.presentation.home.HomeFragment.clickItem	Falha	1.0 – 1.0	1	1

ALTERANDO O LOGO DO NOSSO APLICATIVO



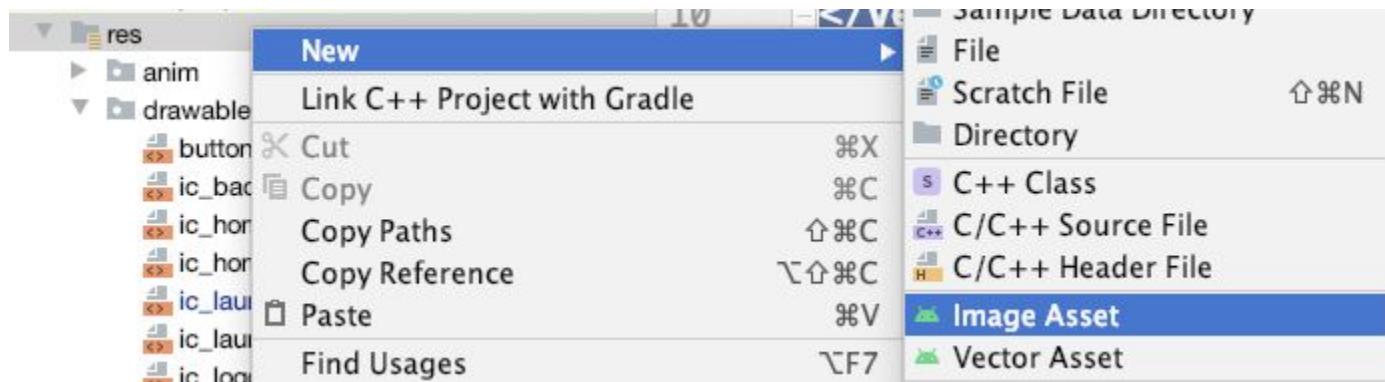
BAIXANDO OS ÍCONES

Baixo os arquivos disponíveis no seguinte link:

https://github.com/heiderlopes/calcu_flex_2_launcher_icon

ADICIONANDO NO PROJETO

Clique com o botão direito sobre a pasta **res**, em seguida, **New → Image Asset**



ADICIONANDO NO PROJETO

Altere o ícone de **Foreground Layer**

 Configure Image Asset

Icon Type: Preview Show Safe Zone Show Grid

Name: ic_launcher

Foreground Layer Background Layer Options

Layer Name: ic_launcher_foreground

Source Asset

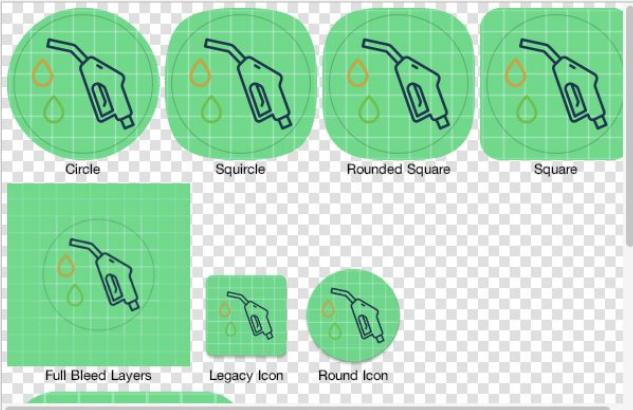
Asset Type: Image Clip Art Text

Path: /eiderlopes/Desktop/ic_logo_prd.svg

Scaling

Trim: Yes No

Resize: 45 %



⚠ An icon with the same name already exists and will be overwritten.

?

Cancel Previous Next Finish

ADICIONANDO NO PROJETO

Altere o **Background Layer**

Configure Image Asset

Icon Type: Launcher Icons (Adaptive and Legacy) Preview xhdpi Show Safe Zone Show Grid

Name: ic_launcher

Foreground Layer Background Layer Options

Layer Name: ic_launcher_background

Source Asset

Asset Type: Color Image

Color: FFFFFF

Scaling

Trim: Yes No

Resize: 100 %

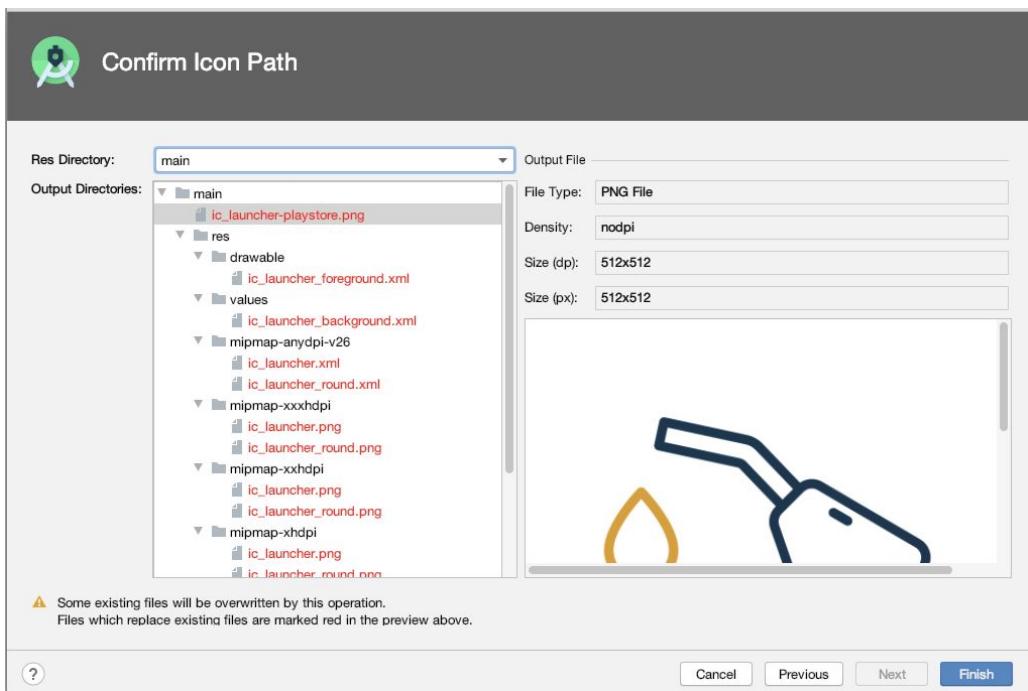
Circle Squircle Rounded Square Square

Full Bleed Layers Legacy Icon Round Icon

Cancel Previous Next Finish

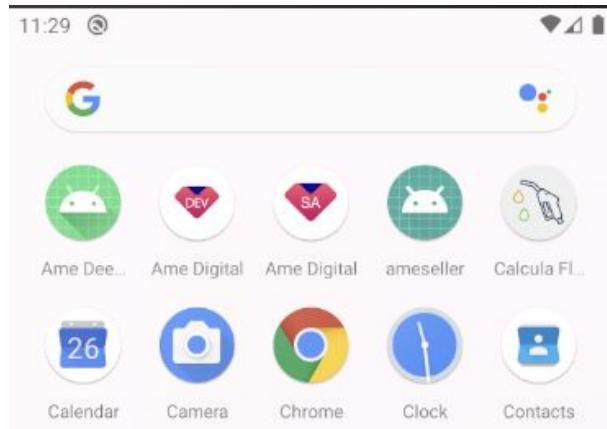
ADICIONANDO NO PROJETO

Após clicar em next, receberemos a seguinte tela que irá mostrar à criação dos ícones nos seus respectivos diretórios:



ADICIONANDO NO PROJETO

Copie o código de **ic_launcher_foreground.xml** para o arquivo **ic_laucher_foreground.xml (v24)**



Source Sets



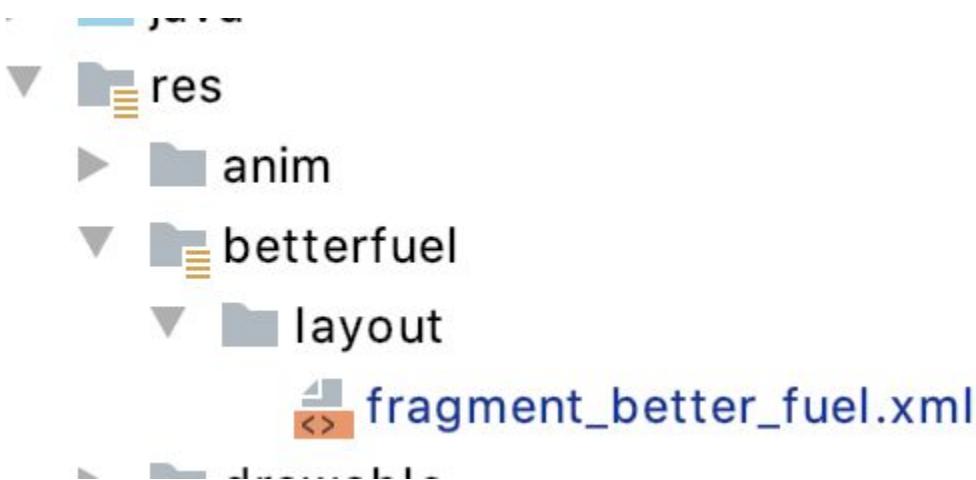
Source Sets

Nossos projetos possuem uma configuração padrão, porém podemos tranquilamente configurar utilizando o bloco sourceSets. Abra o arquivo **build.gradle** do app e adicione:

```
sourceSets {  
    main {  
        res.srcDirs = [  
            'src/main/res/betterfuel',  
            'src/main/res',  
  
        ]  
    }  
}
```

Source Sets

Crie um novo diretório chamado de **betterfuel** e dentro dele uma pasta chamada **layout**. Mova o arquivo **fragment_better_fuel.xml**



Source Sets

Foi definido no slide anterior vários diretórios de recursos (**src/main/res-app/***), um para cada funcionalidade do aplicativo e uma pasta comum para recursos genéricos (como o ícone da aplicação) ou utilizados por mais de uma funcionalidade (**src/main/res/***).

Isso nos ajuda a organizar melhor os arquivos de recursos (imagens, layouts, strings etc.) à medida que o projeto cresce.

Dentro destas pastas é possível criar subdiretórios estabelecidos pelo Android, como:

- drawable
- layout
- values
- Entre outros

. Por exemplo: res-app/products/layout , res-app/products/drawable , res-app/users/values etc.

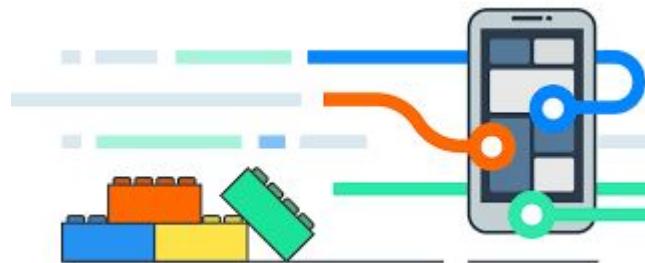
PRODUCT FLAVORS: GERANDO VÁRIOS PRODUTOS E AMBIENTES



PRODUCT FLAVORS

Um mecanismo poderoso no Android é a capacidade de gerar múltiplas versões customizadas de uma aplicação, usando o mesmo projeto.

Um mesmo aplicativo pode ter diferentes versões onde a mudança entre eles **pode ser visual (ícones, cores, textos...) e/ou por funcionalidades**. Este mecanismo é chamado de **Product Flavors**, onde a ideia é que você tenha vários “sabores” do mesmo aplicativo.



PRODUCT FLAVORS - QUAIS CENÁRIOS?

Podemos utilizar products flavors quando:

- o aplicativo tem a versão paga e a versão grátil, com menos funcionalidades.
- precisa lançar uma versão experimental e posteriormente a versão oficial.
- o cliente pediu um aplicativo parecido com um existente e gostaria que mudasse apenas a cor do app, o ícone e tirasse alguns fluxos e incluir algumas funcionalidades.
- versões diferentes para diversos ambientes

Utilizando flavors você não precisaria lidar com código duplicado, múltiplos versionamentos, possíveis erros humanos na hora de copiar códigos e resources, testar duas vezes, basta adicionar a configuração para isso acontecer.



PRODUCT FLAVORS - COMO FAZ?

A diferenciação de um aplicativo de outro é o seu package, também conhecido como **applicationId**. Este package não são os pacotes java da sua aplicação e sim a assinatura do seu aplicativo. Através dele Google identifica o aplicativo no Google Play e o Android no aparelho. Portanto, se houver dois (ou mais) aplicativos onde a diferença entre eles é mínima deverão ter package diferente.

Iremos criar 3 aplicativos através do nosso projeto. São eles:

- **dev**: que será a versão apontando para o ambiente de desenvolvimento.
- **hml**: que será a versão apontando para o ambiente de homologação.
- **prd**: que será a versão apontando para o ambiente de produção.

PRODUCT FLAVORS - COMO FAZ?

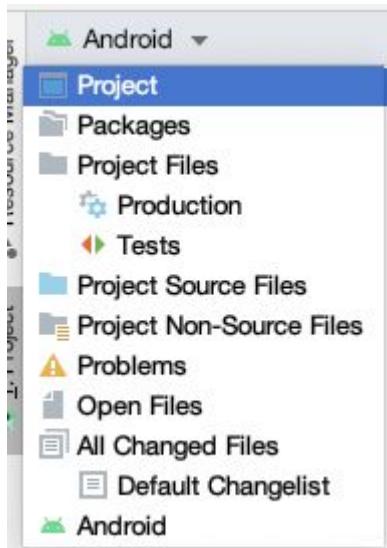
A primeira coisa que devemos configurar são os packages no **build.gradle** do seu aplicativo (aquele da pasta app/). Adicione as seguintes códigos dentro da tag android

flavorDimensions "app"

```
productFlavors {  
    dev {  
        applicationIdSuffix ".dev"  
        buildConfigField 'String', 'CHAVE', "\"VALOR\""  
    }  
  
    hml {  
        applicationIdSuffix ".hml"  
        buildConfigField 'String', 'CHAVE', "\"VALOR\""  
    }  
  
    prd {  
        buildConfigField 'String', 'CHAVE', "\"VALOR\""  
    }  
}
```

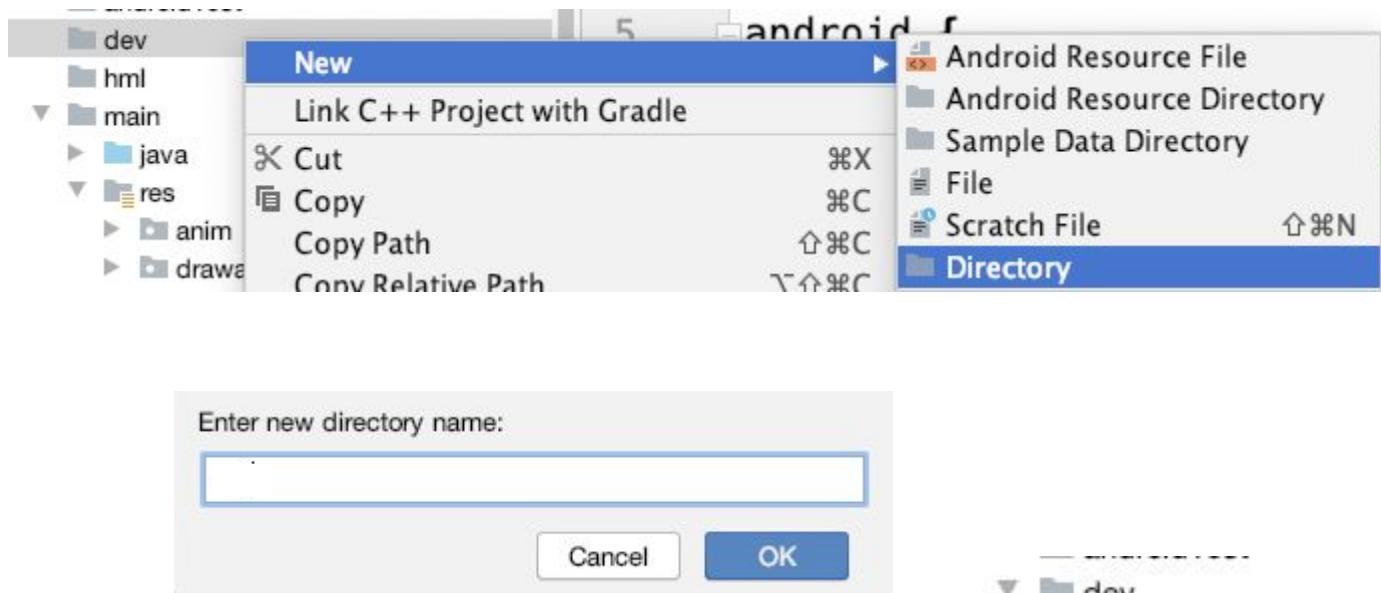
PRODUCT FLAVORS - COMO FAZ?

Altere a visão do app para **Project**:



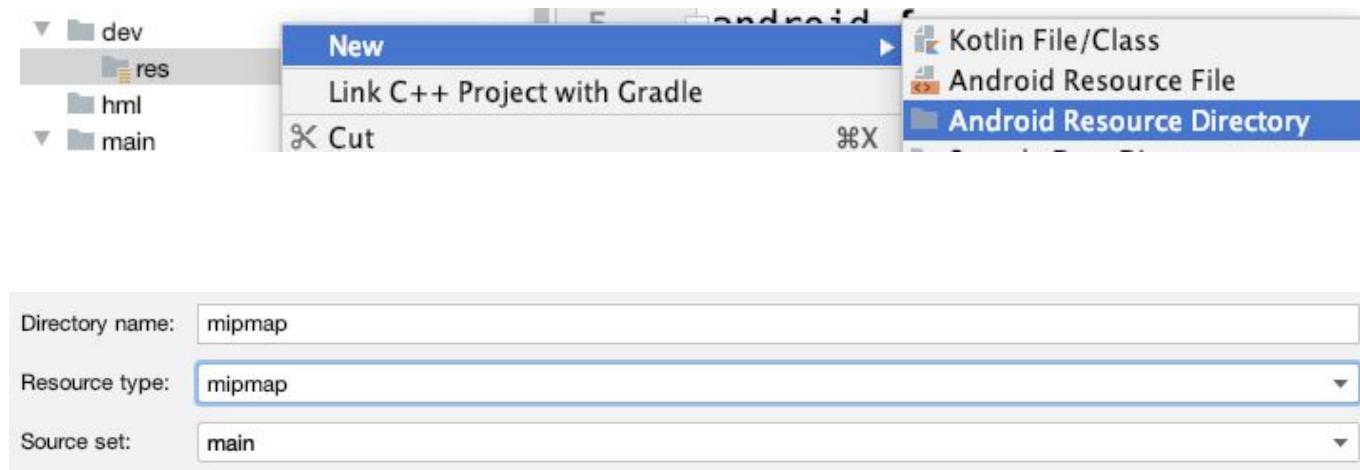
PRODUCT FLAVORS - COMO FAZ?

Crie os diretórios **dev** e **hml**:



PRODUCT FLAVORS - COMO FAZ?

Clique com o **botão direito** → **New** → **Android Resource Directory**



PRODUCT FLAVORS - COMO FAZ?

Adicione os recursos de acordo com a necessidades dos ambientes:

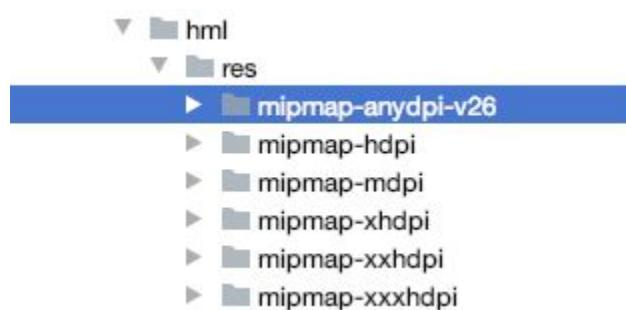


CRIE E ADICIONE AS CONFIGURAÇÕES
PARA O AMBIENTE DE HOMOLOGAÇÃO



PRODUCT FLAVORS - COMO FAZ?

Os arquivos de hml



ADICIONANDO UMA LABEL DE AMBIENTE NAS TELAS DO APP



PRODUCT FLAVORS - COMO FAZ?

Dentro da pasta layout do projeto principal, crie um arquivo chamado **include_flavour.xml** e adicione:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/flavourScreen"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:gravity="end"
    android:background="#FFC107"
    android:orientation="vertical"
    android:visibility="gone">
    <TextView
        android:id="@+id/tvEnvironment"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginEnd="8dp"
        android:layout_marginRight="8dp"
        android:gravity="end"
        android:padding="4dp"
        tools:text="Ambiente"
        android:textColor="#FFFFFF" />
</LinearLayout>
```

PRODUCT FLAVORS - COMO FAZ?

Abra o arquivo **BaseFragment** e adicione o código **em negrito**:

```
override fun onCreateView(
    inflater: LayoutInflater,
    container: ViewGroup?,
    savedInstanceState: Bundle?
): View? {

    val screenRootView = FrameLayout(requireContext())

    val screenView = inflater.inflate(layout, container, false)

    loadingView = inflater.inflate(R.layout.include_loading,
    container, false)
```

PRODUCT FLAVORS - COMO FAZ?

Abra o arquivo **BaseFragment** e adicione o código **em negrito**:

```
val flavourScreen = inflater.inflate(R.layout.include_flavour,  
container, false)  
flavourView = flavourScreen.findViewById(R.id.flavourScreen)  
  
configureEnvironment(  
    flavourView,  
    flavourScreen.findViewById(R.id.tvEnvironment) as TextView  
)  
  
screenRootView.addView(screenView)  
screenRootView.addView(loadingView)  
screenRootView.addView(flavourView)  
  
registerObserver()  
return screenRootView  
}
```

PRODUCT FLAVORS - COMO FAZ?

Abra o arquivo **BaseFragment** e adicione o código **em negrito**:

```
private fun configureEnvironment(container: View, tvEnvironment: TextView) {  
  
    when (BuildConfig.FLAVOR) {  
        "dev" -> {  
            container.visibility = View.VISIBLE  
            tvEnvironment.text = "Desenvolvimento"  
        }  
        "qa" -> {  
            container.visibility = View.VISIBLE  
            tvEnvironment.text = "Homologação"  
        }  
        "main" -> {  
            container.visibility = View.GONE  
            tvEnvironment.text = ""  
        }  
    }  
}
```

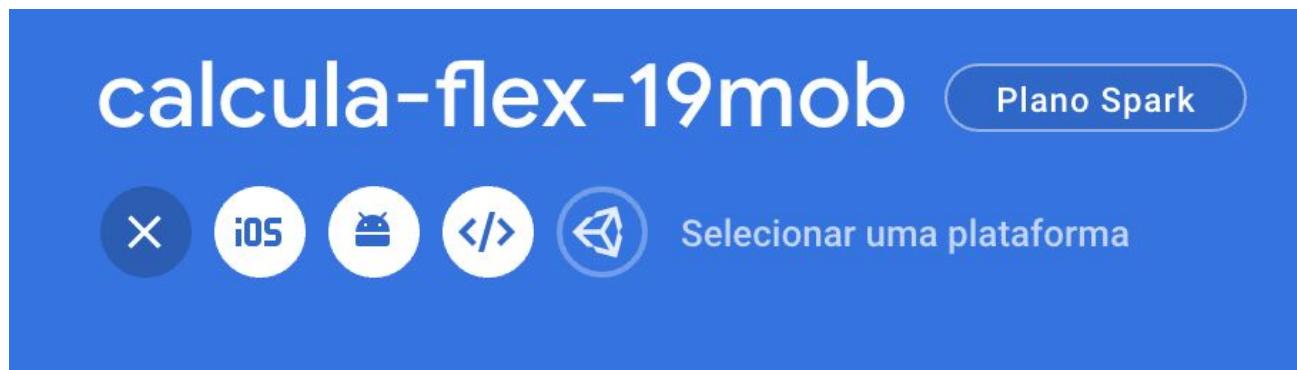
CONFIGURANDO O FIREBASE

Abra a configuração do projeto e clique em **+ Adicionar app**

The screenshot shows the Firebase console interface for the project 'calcula-flex-19mob'. The left sidebar contains navigation links for 'Desenvolver' (Authentication, Database, Storage, Hosting, Functions, Machine Learning), 'Qualidade' (Crashlytics, Performance, Test Lab), and 'Analytics' (Dashboard, Events, Conversions, Audiences, Funnels). The main dashboard displays two apps: 'calcula-flex-19mob' (Piano Spark) and 'br.com.heiderlopes...'. A prominent button '+ Adicionar app' is visible. Key metrics shown include 'Usuários nos últimos 30 minutos' (0), 'Usuários ativos por dia' (0, -100%), and 'Retenção no primeiro dia' (0%, -100%). A call-to-action 'Track your revenue!' with links to 'Link para a AdMob' and 'Vincular ao Google Play' is present. The bottom of the screen features a 'Desenvolver' button.

CONFIGURANDO O FIREBASE

Selecione a plataforma **Android**:



CONFIGURANDO O FIREBASE

Adicione as configurações o do flavour e clique em **Registrar app**

× Adicionar o Firebase ao seu app para Android

CONFIGURANDO O FIREBASE

Faça o download do arquivo **google-services.json** e adicione no projeto conforme indicado:

2

Fazer o download do arquivo de configuração

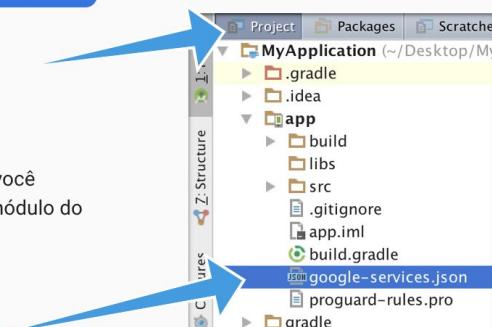
 Fazer o download de google-services.json

Mude para a visualização do Projeto no Android Studio para ver o diretório raiz.

Mova o arquivo google-services.json que você acabou de salvar para o diretório raiz do módulo do app para Android.



google-services.json

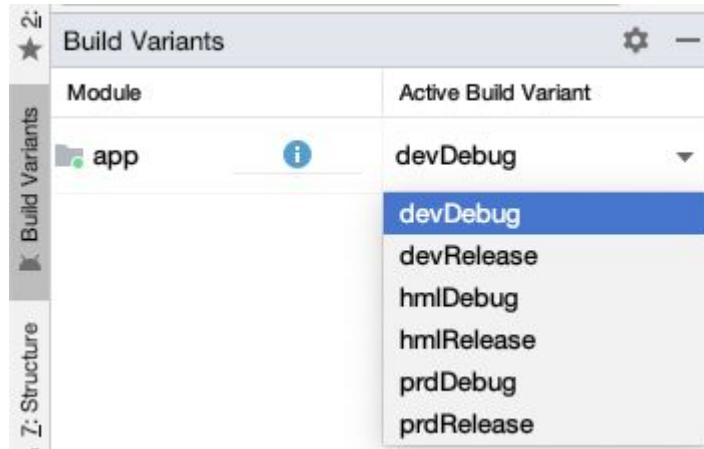


Anterior

Próxima

CONFIGURANDO O FIREBASE

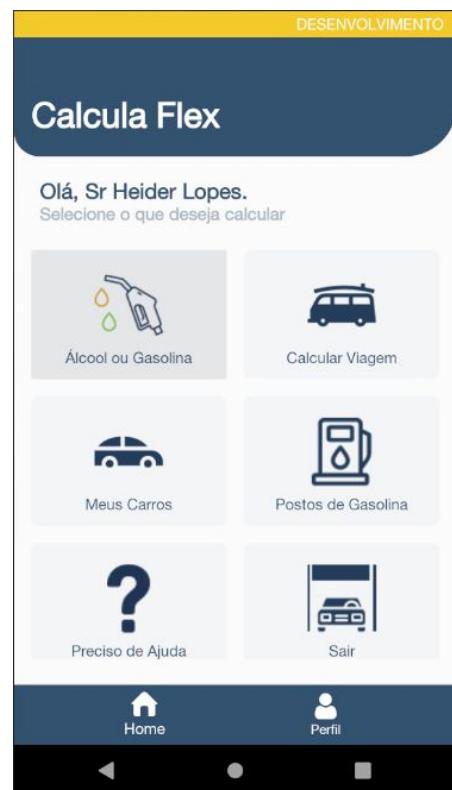
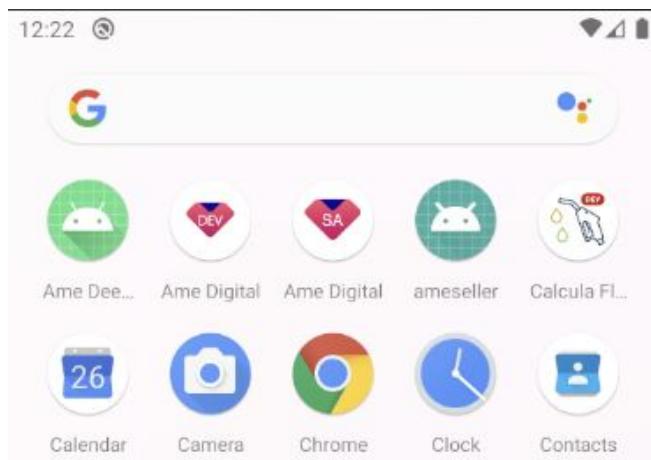
Em **Build Variants** altera para **devDebug** e rode o aplicativo:



CONFIGURANDO O FIREBASE

Rode a aplicação e veja o resultado.

Obs: é necessário rodar o aplicativo para registrar no Firebase e efetivar a configuração.



EXERCÍCIO:

CONFIGURE O AMBIENTE DE HOMOLOGAÇÃO NO FIREBASE



RESULTADO ATÉ O MOMENTO



APP PARA OS TRÊS AMBIENTES

DESENVOLVIMENTO

HOMOLOGAÇÃO

Calcula Flex

Olá, Sr Heider Lopes.

Selecione o que deseja calcular



Álcool ou Gasolina



Calcular Viagem



Meus Carros



Postos de Gasolina



Preciso de Ajuda



Sair



Home



Perfil

HOMOLOGAÇÃO

Calcula Flex

Olá, Sr Heider Lopes.

Selecione o que deseja calcular



Álcool ou Gasolina



Calcular Viagem



Álcool ou Gasolina



Calcular Viagem



Meus Carros



Postos de Gasolina



Postos de Gasolina



Meus Carros



Postos de Gasolina



Preciso de Ajuda



Sair



Home



Perfil

DISTRIBUINDO O APLICATIVO PARA EQUIPE INTERNA



FIREBASE APP DISTRIBUTION

O **Firebase App Distribution** facilita a distribuição dos seus aplicativos para testadores confiáveis. Ao disponibilizar seus apps para os testadores rapidamente, é possível receber feedback com antecedência e frequência.

Além disso, se você usar o **Crashlytics** nos seus aplicativos, receberá automaticamente métricas de estabilidade para todas as suas versões, para saber quando está tudo pronto para o envio.



FIREBASE APP DISTRIBUTION

Selecione **App Distribution** e defina qual pacote você irá distribuir, por exemplo, **dev**, **hml** e **prd**.

The screenshot shows the Firebase console interface for app distribution. On the left, there's a sidebar with icons for Hosting, Functions, Machine Learning, Qualidade, Crashlytics, Performance, Test Lab, and App Distribution (which is selected and highlighted in blue). The main area has a blue background with a woman holding a yellow box. It displays the project name "calcula-flex-19mob" and a "Beta" button. The central part is titled "App Distribution" with the sub-instruction "Distribua versões de pré-lançamento para seus apps para trusted testers". A "Primeiros passos" button is visible. A callout box lists three distribution packages: "br.com.heiderlopes.calculaflex", "Calcula Flex DEV", "Calcula Flex HML", and "br.com.heiderlopes.calculaflex2". At the bottom, there's a checkbox for accepting terms and conditions.

Firebase

Hosting

Functions

Machine Learning

Qualidade

Crashlytics

Performance

Test Lab

App Distribution

calcula-flex-19mob ▾

Beta

App Distribution

Distribua versões de pré-lançamento para seus apps para trusted testers

Primeiros passos

Aceito os [Termos de Serviço do Firebase Crashlytics e Firebase App Distribution](#)

- br.com.heiderlopes.calculaflex
- Calcula Flex DEV
- Calcula Flex HML
- br.com.heiderlopes.calculaflex2

FIREBASE APP DISTRIBUTION

Selecione o pacote e aceite os termos de uso.

Após isso, clique em **Primeiros Passos**

The screenshot shows the Firebase console interface for 'App Distribution'. On the left, there's a sidebar with icons for Hosting, Functions, Machine Learning, Quality (Crashlytics, Performance, Test Lab), and Analytics (Dashboard). The main area has a blue background with a cartoon character holding a yellow box. It displays the project name 'calcula-flex-19mob' and a 'Beta' status. The central heading is 'App Distribution' with the sub-instruction 'Distribua versões de pré-lançamento dos seus apps para trusted testers'. Below this is a 'Primeiros passos' button and a checked checkbox for accepting terms of service. The URL 'br.com.heiderlopes.calculaflex' is also visible.

FIREBASE APP DISTRIBUTION

Selecione a aba **Testadores e grupos** e adicione seus testers/

The screenshot shows the 'App Distribution' interface. At the top, there are navigation links: 'calcula-flex-19mob ▾', 'Acessar a documentação', a bell icon, and a user profile icon with the letter 'H'. Below the header, the main title 'App Distribution' is displayed. Underneath it, there are three tabs: 'Lançamentos', 'Links de convite', and 'Testadores e grupos', with 'Testadores e grupos' being the active tab. On the left side, there is a sidebar with two sections: 'Testadores' and 'Grupos'. The 'Testadores' section contains a button 'Ver todos os testadores' which is highlighted with a blue background and white text, indicating it is selected. Below this, it says '0 testador'. To the right of the sidebar, the main content area has a heading 'Todos os testadores' with a user icon and a 'Mais' button. Below this, there is a button 'Adicionar testadores'. At the bottom of the sidebar, there is a link 'Adicionar grupo'.

FIREBASE APP DISTRIBUTION

Após isso, selecione **Lançamentos** e adicione o apk gerado:

calculaflex-19mob ▾ Acessar a documentação  

App Distribution

br.com.heiderlopes.calculaflex ▾ 

[Lançamentos](#) [Links de convite](#) [Testadores e grupos](#)

 Arraste qualquer arquivo .apk para cá caso queira uma nova versão. [Saiba mais](#)  Procurar

[Lançamentos](#) E-mail de contato  heidertreinamentos@gmail.com 

 [Search versions and notes](#)



Nenhuma versão foi criada ainda.

FIREBASE APP DISTRIBUTION

Adicione os testadores e clique em **Próxima.**

The screenshot shows the Firebase App Distribution interface. At the top, there's a navigation bar with 'calcula-flex-19mob' (dropdown), 'Acessar a documentação' (link), a bell icon, and a user profile icon. Below the navigation, the page title is 'App Distribution' with a dropdown set to 'br.com.heiderlopes.calculaflex'. There are three tabs: 'Lançamentos' (selected), 'Links de convite', and 'Testadores e grupos'. The main content area has a dashed blue border. Inside, there's a message: 'Arraste qualquer arquivo .apk para cá caso queira uma nova versão.' followed by a 'Saiba mais' link and a 'Procurar' button. Below this is a 'Lançamentos' section with an 'E-mail de contato' field containing 'heidertreinamentos@gmail.com' with a pencil icon. A search bar says 'Search versions and notes'. A card for version '1.0 (1)' is shown, stating 'Não distribuída – .apk (upload concluído)'. It has two steps: '1 Adicionar testadores' and '2 Adicione as notas da versão (opcional)'. A 'Excluir' button is also present. At the bottom, there's a button to 'Adicionar testadores ou grupos' with a user icon.

FIREBASE APP DISTRIBUTION

Adicione o **Release Note** e clique em **Distribuir...**

1.0 (1)

Não distribuída – .apk (upload concluído)

1 Adicionar testadores

2 Adicione as notas da versão
(opcional)

[Excluir](#)

Adicione suas notas da versão aqui

Visível nos e-mails de versão e no app para testadores do App Distribution

OBRIGADO



/heider.lopes



/in/heider-lopes-a06b2869/

FIAP

Copyright © 2020 | Professor (a) Heider Lopes

Todos os direitos reservados. Reprodução ou divulgação total ou parcial deste documento, é expressamente proibido sem consentimento formal, por escrito, do professor/autor.

FIAP