

ENGINEERING SOFTWARE & DEVELOPMENT

PUSH NOTIFICATION COM FIREBASE E ANDROID

HEIDER PINHOLI LOPES



4

LISTA DE FIGURAS

Figura 4.1 - Dependência do FCM	7
Figura 4.2 - Painel do Firebase Cloud Messaging na Plataforma do Google	8
Figura 4.3 - Criação do pacote fcm	9
Figura 4.4 - Nomeando o pacote fcm	9
Figura 4.5 - Criação da classe CalculaFlexFCMService	10
Figura 4.6 - Nomeação da classe CalculaFlexFCMService	10
Figura 4.7 – Exemplo de notificações na barra de status.....	14
Figura 4.8 – Exemplo de notificações ao deslizar a barra de notificações	15
Figura 4.9 - Painel para ativar/desativar notificações	16
Figura 4.10 - Notificação com actions	17
Figura 4.11 - Criação do pacote utils.....	19
Figura 4.12 - Nomeando o pacote utils.....	20
Figura 4.13 - Criação da classe NotificationUtils	20
Figura 4.14 - Nomeando a NotificationUtils	20
Figura 4.15 - Exemplo de notificação simples	22
Figura 4.16 - Visualizar os canais da notificação/silenciar notificação	23
Figura 4.17 - Canais de notificação de um aplicativo e seu status.....	23
Figura 4.18 – Notificação com texto longo	26
Figura 4.19 - Notificação com botões de ação	27
Figura 4.20 - Notificação com digitação de texto botão de ação	28
Figura 4.21 - Notificação com digitação de texto resposta	29
Figura 4.22 - Notificações agrupadas.....	30
Figura 4.23 - Painel Ampliar do Firebase	32
Figura 4.24 - Enviando a primeira mensagem 1	33
Figura 4.25 - Enviando a primeira mensagem 2.....	34
Figura 4.26 - Segmentação de usuários para receber a notificação	35
Figura 4.27 - Programando o envio da mensagem	35
Figura 4.28 - Dados opcionais da notificação 1	36
Figura 4.29 - Dados opcionais da notificação 2.....	37
Figura 4.30 - Exibição da notificação	37
Figura 4.31 - Token do usuário exibido no Logcat do Android Studio	38
Figura 4.32 - Acessando as configurações do projeto	39
Figura 4.33 - Obtendo a chave do servidor	39
Figura 4.34 - Adicionando os headers na request.....	39
Figura 4.35 - Configuração para enviar a mensagem	40
Figura 4.36 - Notificação enviada.....	41
Figura 4.37 – Selecionando o BetterFuelFragment.....	43
Figura 4.38 – Criando o deep link para tela de Melhor Combustível.....	43
Figura 4.39 – Criando o deep link para tela de SignUp.....	43
Figura 4.40 - Criando uma configuração	44
Figura 4.41 - Criando uma configuração	44
Figura 4.42 - Criando uma configuração	45
Figura 4.43 - Configurando Launch de Deeplink para tela de melhor combustível ...	45
Figura 4.44 - Criando uma configuração	45
Figura 4.45 - Criando uma configuração	46
Figura 4.46 - Criando uma configuração	46
Figura 4.47 - Configurando Launch de Deeplink para tela de criação de conta	46

Figura 4.48 - Aplicativo sendo executado direto para tela do Deeplink.....	47
Figura 4.49 - Adicionando os headers na request.....	48
Figura 4.50 - Configuração para enviar a mensagem	48
Figura 4.51 - Notificação enviada.....	49
Figura 4.52 - Exibição da notificação com deep link	51

EMANIP

LISTA DE TABELAS

Tabela 4.1 – AndroidManifest.xml com os metadados adicionais	17
Tabela 4.2 – Criação de um canal de notificação.....	24

EXEMPLO

LISTA DE CÓDIGOS-FONTE

Código-fonte 4.1 – Dependência do FCM	8
Código-fonte 4.2 – Atualização do Google Services	9
Código-fonte 4.3 – Estrutura básica da implementação do FCM	10
Código-fonte 4.4 – Declaração da classe CalculaFlexFCMService no AndroidManifest.xml	11
Código-fonte 4.5 – Opcional elementos metadados para notificação	12
Código-fonte 4.6 – Opcional canal de notificação	12
Código-fonte 4.7 – Adição da string referente ao canal das notificações	13
Código-fonte 4.8 – AndroidManifest.xml com os metadados adicionais.....	14
Código-fonte 4.9 – Classe NotificationUtils	21
Código-fonte 4.10 – Criação de um canal de notificação	24
Código-fonte 4.11 – Criação de um canal de notificação	25
Código-fonte 4.12 – Notificação com Texto Longo.....	27
Código-fonte 4.13 – Notificação com botões de ação	28
Código-fonte 4.14 – Notificação com digitação de texto	30
Código-fonte 4.15 – Notificações agrupadas.....	31
Código-fonte 4.16 – Classe para exibir a notificação enviada pelo FCM	32
Código-fonte 4.17 – Payload do FCM	40
Código-fonte 4.18 – Adição do deeplink.....	44
Código-fonte 4.19 – Payload do FCM	48
Código-fonte 4.20 – Notificação enviada 1	50
Código-fonte 4.21 – Notificação enviada 2.....	51

SUMÁRIO

4 PUSH NOTIFICATION COM FIREBASE E ANDROID	7
4.1 Implementando o FCM no projeto	8
4.2 Notificações no Android.....	14
4.2.1 Criando a classe de Notificação	19
4.2.2 Canais de notificação	22
4.2.3 Visibilidade das Notificações	25
4.2.4 Notificação com texto longo	26
4.2.5 Notificação com botões de ação.....	27
4.2.6 Notificação com digitação de texto	28
4.2.7 Notificações agrupadas	30
4.3 Projeto de notificações	31
4.3.1 Envio de Notificações pelo Firebase	32
4.3.2 Disparando a mensagem via API	37
4.4 Deeplinks.....	41
4.4.1 Adicionando deeplink no projeto	42
4.4.2 Testando o deep link pelo Android Studio	44
4.4.3 Abrindo deeplink via Push Notification	47
CONCLUSÃO.....	52
REFERÊNCIAS.....	53

4 PUSH NOTIFICATION COM FIREBASE E ANDROID

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

Usando o FCM (Firebase Cloud Messaging), é possível notificar um app cliente de que novos e-mails ou outros dados estão disponíveis para sincronização, divulgar promoções, emitir alertas, entre outros.

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.

Por meio do FCM, as mensagens podem ser enviadas por meio do console fornecido pelo site do Firebase (console.firebase.google.com), por meio da implementação da API no seu backend ou da chamada do serviço via HTTP.

O uso do FCM pode ser feito por aplicações Android, iOS e WEB.



Figura 4.1 - Dependência do FCM
Fonte: Elaborada pelo autor (2020)

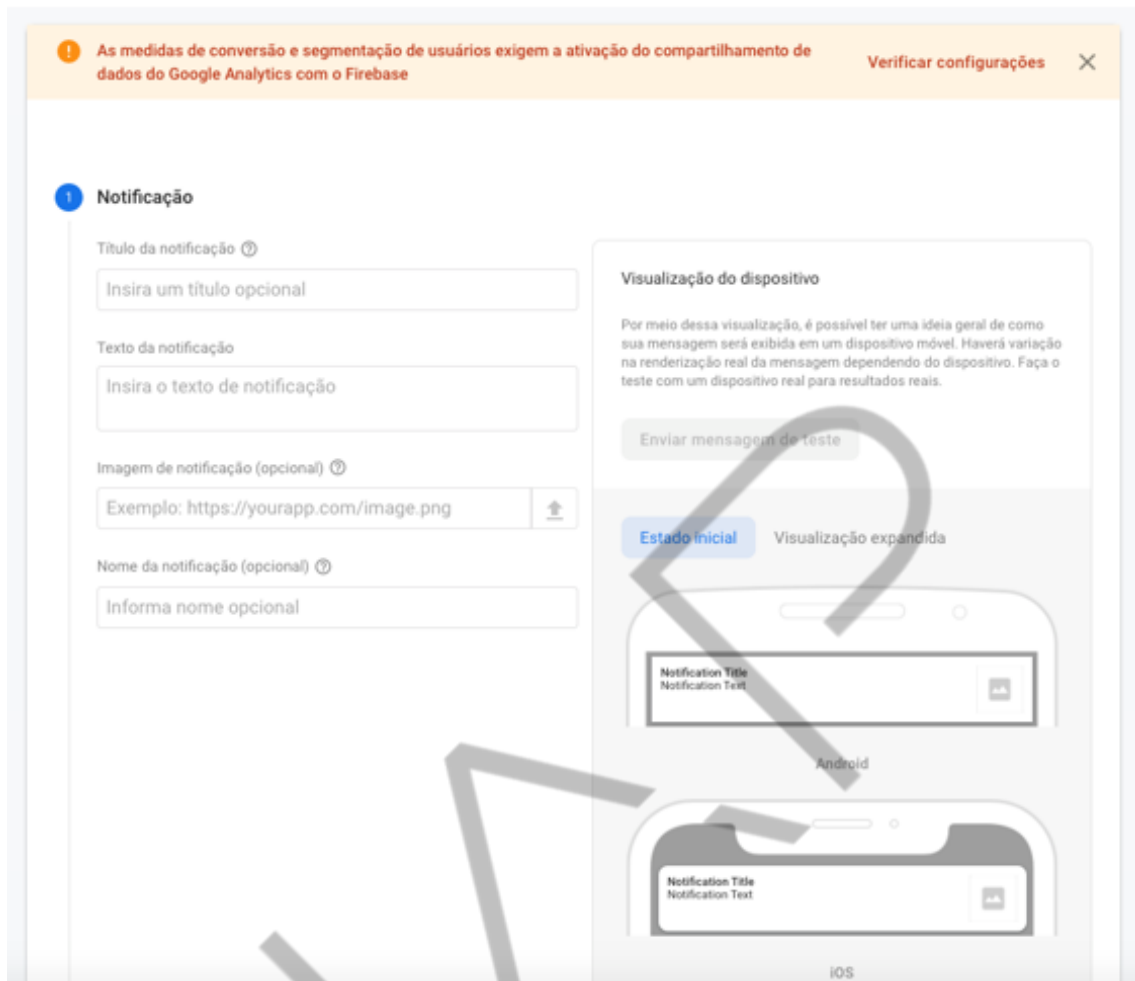


Figura 4.2 - Painel do Firebase Cloud Messaging na Plataforma do Google
Fonte: Elaborado pelo autor (2020)

Veja o link da documentação completa do FCM:
<<https://firebase.google.com/docs/cloud-messaging?hl=pt>>.

4.1 Implementando o FCM no projeto

Vamos começar adicionando a dependência do FCM em nosso projeto. Abra o arquivo **build.gradle (app)** e adicione a seguinte dependência.

```
implementation 'com.google.firebase:firebase-messaging:20.2.4'
```

Código-fonte 4.1 – Dependência do FCM
Fonte: Elaborado pelo autor (2020)

Abra o arquivo **build.gradle (project)** e atualize a versão do plugin do **google-services**:

```
classpath 'com.google.gms:google-services:4.3.3'
```

Código-fonte 4.2 – Atualização do Google Services
Fonte: Elaborado pelo autor (2020)

Continuando com o desenvolvimento do aplicativo Calcula Flex, crie um package chamado **fcm** na raiz do projeto.

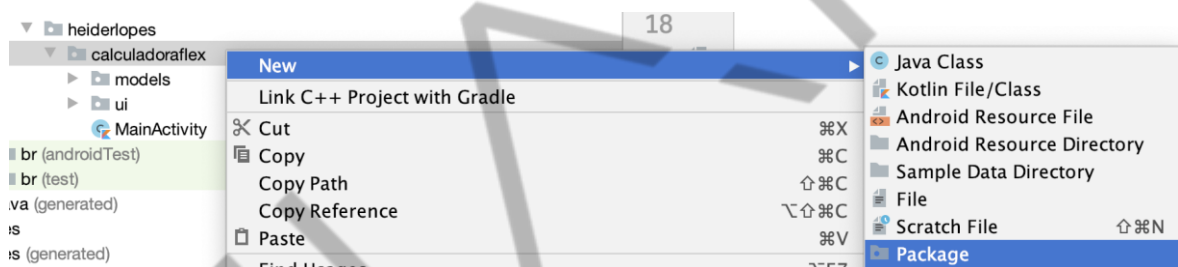


Figura 4.3 - Criação do pacote fcm
Fonte: Elaborado pelo autor (2020)

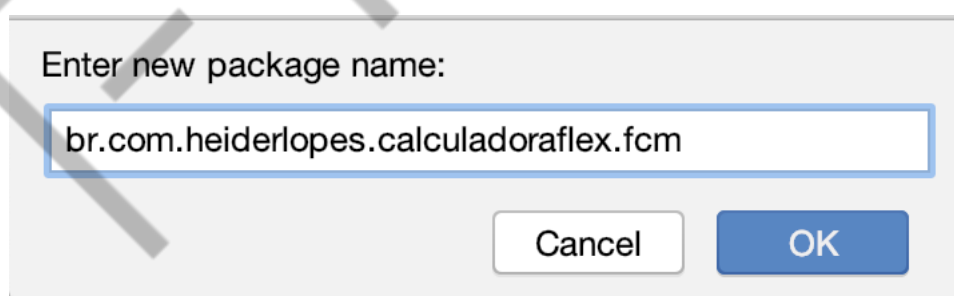


Figura 4.4 - Nomeando o pacote fcm
Fonte: Elaborado pelo autor (2020)

Dentro desse pacote será criada a classe **CalculaFlexFCMService**. Essa classe estenderá a classe **FirebaseMessagingService** do Firebase. Isso é necessário para processar qualquer mensagem, além de simplesmente receber notificações em segundo plano do app. Para receber notificações em apps em

primeiro plano ou payload de dados, enviar mensagens upstream e assim por diante, esta classe precisa estender esse serviço.

Para criá-la no projeto, clique com botão direito do mouse sobre o package **fcm** → **New** → **Kotlin File/Class**.

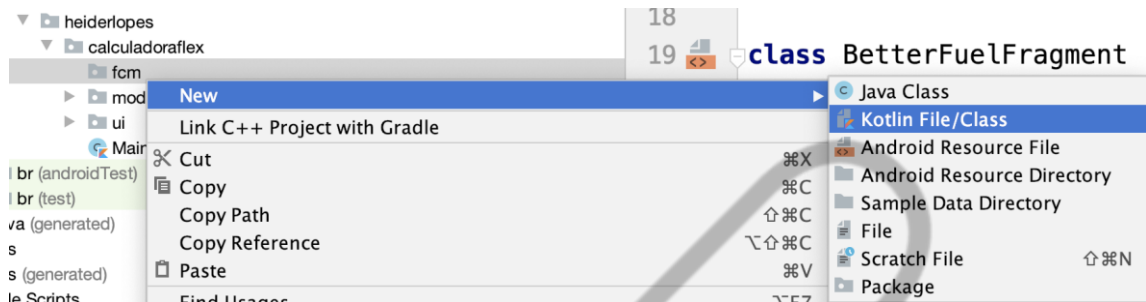


Figura 4.5 - Criação da classe CalculaFlexFCMService

Fonte: Elaborado pelo autor (2020)

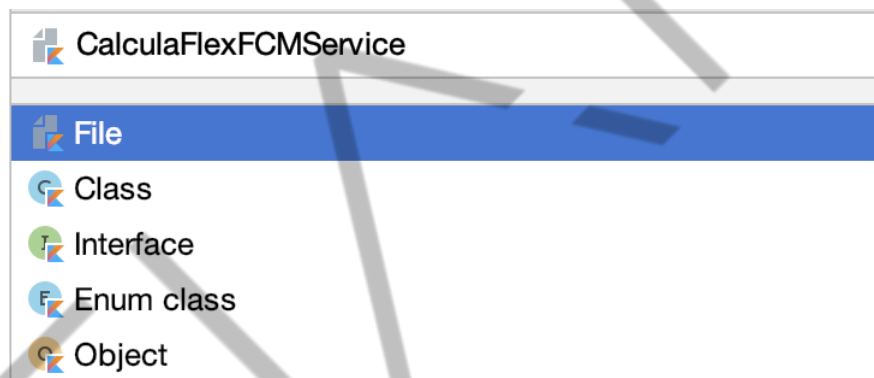


Figura 4.6 - Nomeação da classe CalculaFlexFCMService

Fonte: Elaborado pelo autor (2020)

```
class CalculaFlexFCMService : FirebaseMessagingService() {

    override fun onMessageReceived(p0: RemoteMessage) {
        super.onMessageReceived(p0)
    }

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

Código-fonte 4.3 – Estrutura básica da implementação do FCM

Fonte: Elaborado pelo autor (2020)

Como a classe **CalculaFlexFCMService** estende a classe **FirebaseMessagingService** (que, por sua vez, estende a classe **Service** do Android), é necessário adicionar a declaração desse componente ao arquivo **AndroidManifest.xml**.

Abra o arquivo **AndroidManifest.xml** e realize a declaração do serviço conforme o código em negrito no exemplo apresentado no código-fonte “Declaração da classe **CalculaFlexFCMService** no **AndroidManifest.xml**”.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="br.com.heiderlopes.calculadoraflex">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

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

</manifest>
```

Código-fonte 4.4 – Declaração da classe **CalculaFlexFCMService** no **AndroidManifest.xml**
Fonte: Elaborado pelo autor (2020)

Abra o arquivo **AndroidManifest.xml** e realize a declaração do serviço conforme o código em negrito no Código-fonte “Opcional elementos metadados para notificação”.

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.firebase.messaging.default_notification_icon"
    android:resource="@drawable/ic_launcher_foreground" />

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

Código-fonte 4.5 – Opcional elementos metadados para notificação
Fonte: Elaborado pelo autor (2020)

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.firebase.messaging.default_notification_channel_id"
    android:value="@string/default_notification_channel_id" />
```

Código-fonte 4.6 – Opcional canal de notificação
Fonte: Elaborado pelo autor (2020)

Para implementá-los, abra o arquivo **strings.xml** e adicione a string referente ao canal (caso precise, altere/adicione uma nova cor também no arquivo colors.xml).

```
<string name="default_notification_channel_id">default</string>
<string name="default_notification_channel_name">Padrão</string>
<string name="default_notification_channel_description">Canal padrão de
notificações</string>
```

Código-fonte 4.7 – Adição da string referente ao canal das notificações
Fonte: Elaborado pelo autor (2020)

Abra novamente o arquivo **AndroidManifest.xml** e adicione os metadados conforme o exemplo no código-fonte “AndroidManifest.xml com os metadados adicionais”.

```
<?xml version="1.0" encoding="utf-8" ?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="br.com.heiderlopes.calculadoraflex">
    <uses-permission android:name="android.permission.INTERNET"/>
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

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

        <meta-data
            android:name="com.google.firebase.messaging.default_notification_icon"
            android:resource="@drawable/ic_launcher_foreground" />

        <meta-data
            android:name="com.google.firebase.messaging.default_notification_color"
            android:resource="@color/colorAccent" />

        <meta-data
```

```
android:name="com.google.firebase.messaging.default_notification_channel_id"
    android:value="@string/default_notification_channel_id" />
</application>

</manifest>
```

Código-fonte 4.8 – AndroidManifest.xml com os metadados adicionais
Fonte: Elaborado pelo autor (2020)

4.2 Notificações 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.

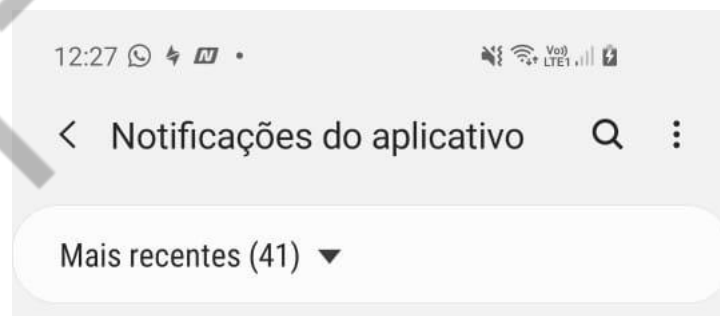


Figura 4.7 – Exemplo de notificações na barra de status
Fonte: Elaborado pelo autor (2020)

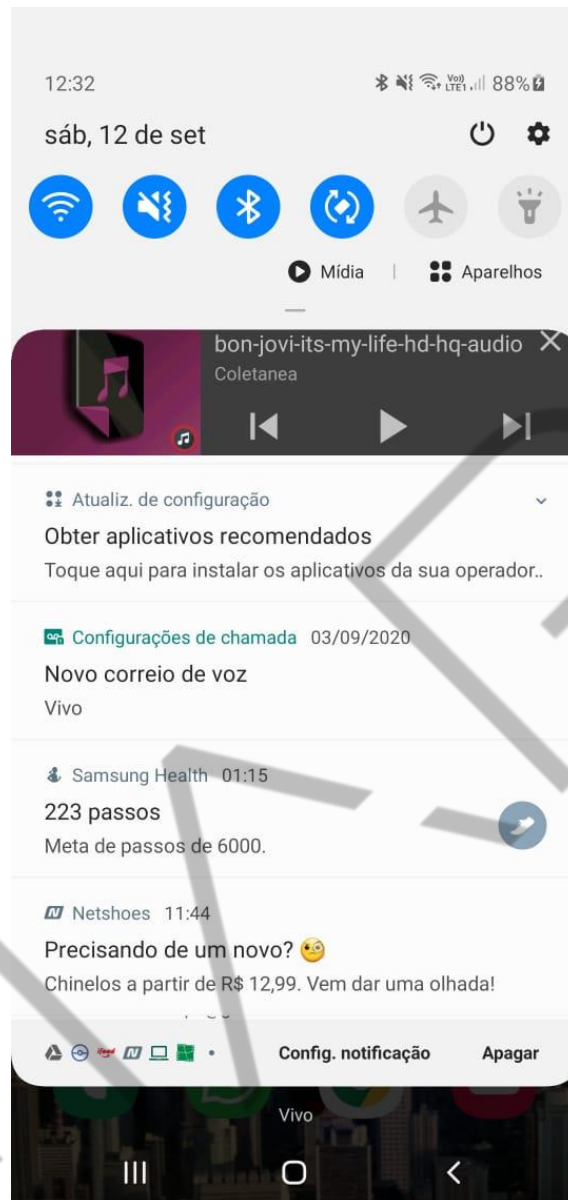


Figura 4.8 – Exemplo de notificações ao deslizar a barra de notificações
Fonte: Elaborado pelo autor (2020)

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.

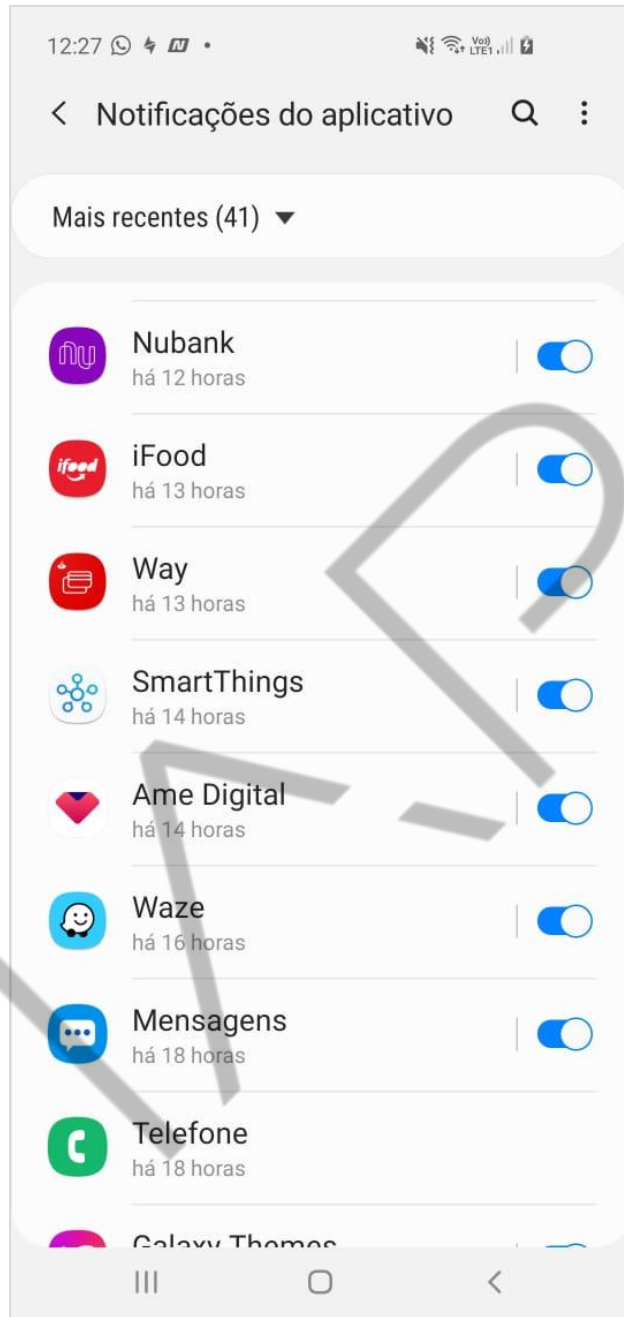


Figura 4.9 - Painel para ativar/desativar notificações
Fonte: Elaborado pelo autor (2020)

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.

Por meio 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 “Notificação com actions”, a notificação pode, além de exibir mensagem, controlar o aplicativo, tocar/pausar e avançar/voltar a música.

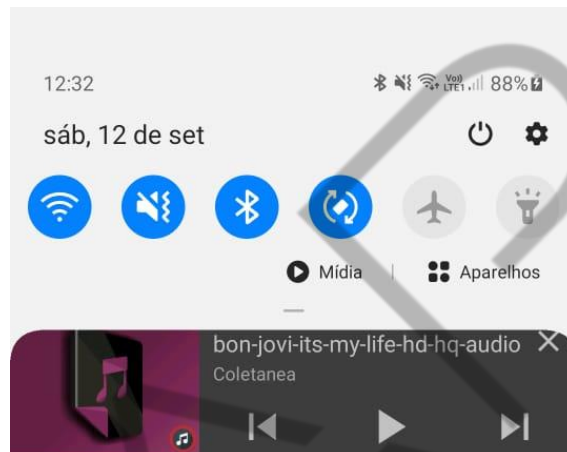


Figura 4.10 - Notificação com actions

Fonte: Elaborado pelo autor (2020)

Veja a tabela contendo as principais características, que podem ser adicionadas a uma notificação, seu respectivo método da classe **NotificationCompat.Builder** e a menor versão na qual o recurso está disponível.

Tabela 4.1 – AndroidManifest.xml com os metadados adicionais

Método	Propósito	API Level
setSmallIcon(int)	Define um ícone para notificação (obrigatório). Deve apontar para uma imagem preferencialmente de 24 dp x 24 dp. Esse ícone ficará em destaque, caso não tenha sido usado o método setLargeIcon(Bitmap) em aparelhos com Android 2.3 ou inferior.	1
setContentTitle(CharSequence)	Texto principal da notificação (obrigatório).	1
setContentText(CharSequence)	Texto que aparecerá abaixo do título (obrigatório).	1

setTicker(CharSequence)	Texto exibido assim que a notificação chega ao aparelho. No Lollipop, foi substituído pela heads-up.	1
setWhen(long)	Hora em que será exibida a notificação.	1
setAutoCancel(boolean)	Define se a notificação deve ser removida automaticamente quando for clicada.	1
setOngoing(boolean)	Informa que o evento relativo a essa notificação está acontecendo. Nesse caso, a notificação não poderá ser removida pelo usuário.	1
setContentIntent(PendingIntent)	PendingIntent que contém a Intent que será disparada ao clicar na notificação.	1
setDeleteIntent(PendingIntent)	PendingIntent que contém a Intent que será disparada ao remover uma notificação.	1
setLights(int, long, long)	Define a cor do led de notificações do aparelho (em RGB) e quantos milissegundos ele ficará aceso e apagado.	1
setSound(Uri)	Atribui um som personalizado para a notificação.	1
setVibrate(long[])	Define uma sequência com os milissegundos de delay para iniciar a vibração e o tempo que o aparelho vai vibrar. Necessita da permissão android.permission.VIBRATE.	1
setDefaults(int)	Permite habilitar o som, vibração e luz padrão do aparelho para notificação.	1
addAction(int, CharSequence, PendingIntent)	Permite adicionar até três ações que ficarão abaixo da notificação e que podem realizar uma tarefa diferente da que é realizada ao clicar na notificação.	16
setContentInfo(CharSequence) setNumber(int)	Texto/número que aparecerá do lado direito da notificação.	11

setSubText(CharSequence)	Texto opcional para terceira linha da notificação.	11
setLargeIcon(Bitmap)	Imagem para ser exibida em destaque no lado esquerdo da notificação.	11
setStyle(NotificationCompat.Style)	Define um estilo para notificação.	16
setPriority(int)	Sugere a prioridade da notificação no notification drawer. Pode assumir os valores: PRIORITY_MIN, PRIORITY_LOW, PRIORITY_DEFAULT, PRIORITY_HIGH ou PRIORITY_MAX.	16
setVisibility(int)	Determina a visibilidade da notificação da tela de bloqueio.	22
setColor(int)	Define uma cor para o ícone e ações da notificação.	22
setBadgeIconType(int)	Define qual tipo de ícone deve ser exibido como badge da notificação. Esse ícone ficará na parte superior direita do ícone do aplicativo.	26
setTimeoutAfter(long)	Define um timeout para que a notificação seja excluída automaticamente.	26
build()	Constrói um objeto Notification.	1

Fonte: GLAUBER (2019)

4.2.1 Criando a classe de Notificação

Crie um pacote chamado de **utils** na raiz do projeto.

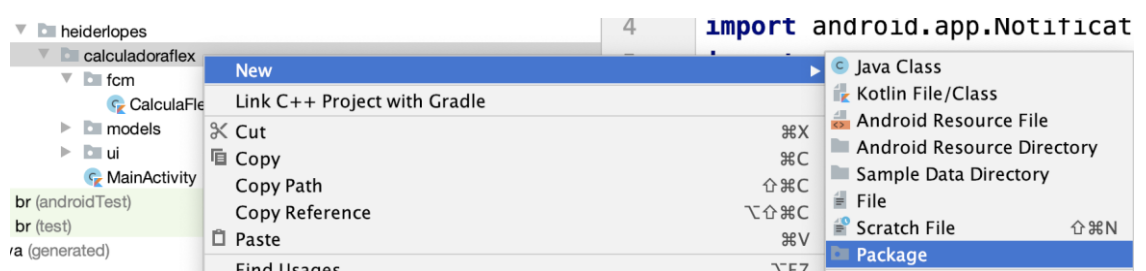


Figura 4.11 - Criação do pacote utils

Fonte: Elaborado pelo autor (2020)

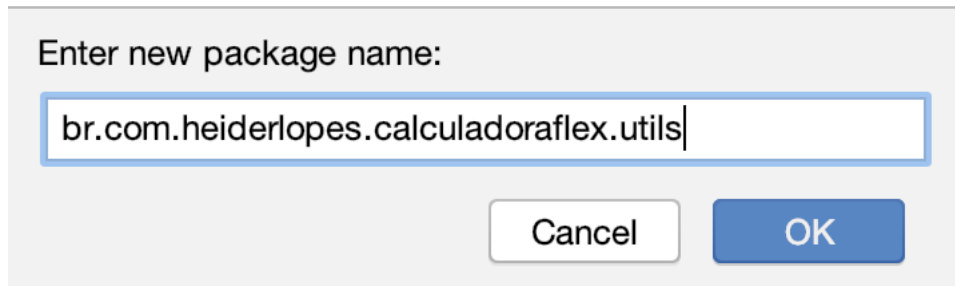


Figura 4.12 - Nomeando o pacote utils
Fonte: Elaborado pelo autor (2020)

Agora, dentro do pacote **utils** criado, adicione uma nova classe chamada **NotificationUtils**.

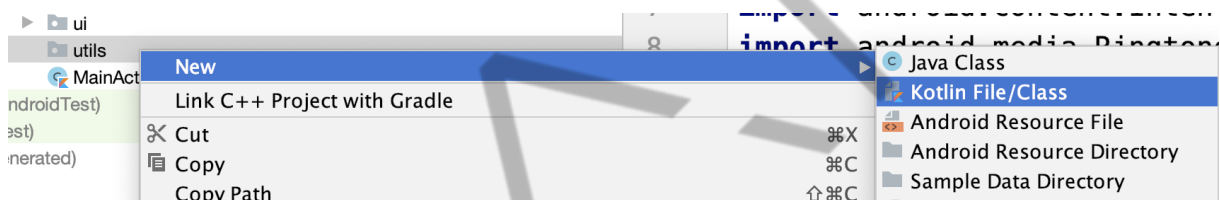


Figura 4.13 - Criação da classe NotificationUtils
Fonte: Elaborado pelo autor (2020)

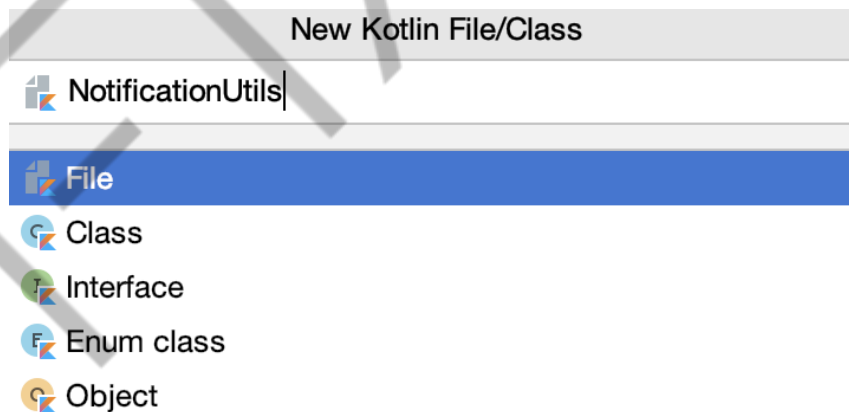


Figura 4.14 - Nomeando a NotificationUtils
Fonte: Elaborado pelo autor (2020)

Para exibir uma notificação simples, adicione o seguinte código:

```
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)
}

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.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())
}
```

Código-fonte 4.9 – Classe NotificationUtils
Fonte: Elaborado pelo autor (2020)

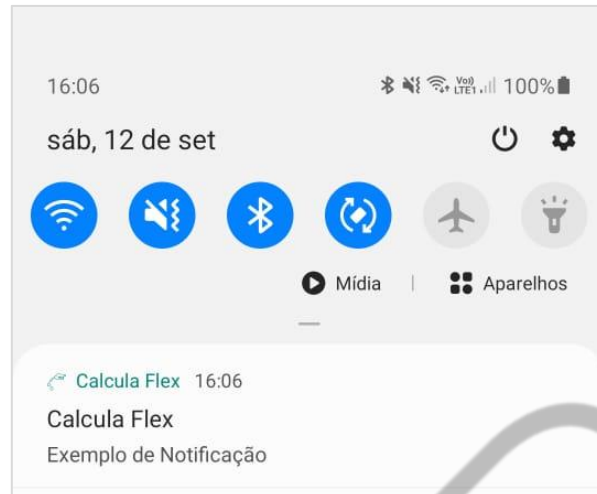


Figura 4.15 - Exemplo de notificação simples
Fonte: Elaborado pelo autor (2020)

O código do “Código-fonte “Classe NotificationUtils” é responsável por exibir uma notificação que poderá ser reaproveitada em toda a aplicação. Ela foi feita para ser genérica, com isso, será possível exibir notificação tanto vinda do FCM como qualquer outra notificação.

4.2.2 Canais de notificação

Introduzido no Android Oreo (api 26), é possível categorizar e agrupar as notificações do aplicativo e, dessa forma, em vez de o 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, no qual o usuário poderia desligar o de promoções, caso não quisesse receber esse tipo de mensagem, porém, continuaria recebendo as notificações referentes 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 nesse ícone, é possível visualizar o canal do qual essa notificação faz parte. Em algumas versões, será possível silenciar a notificação fazendo o mesmo movimento, clicando no ícone do sino que aparece.

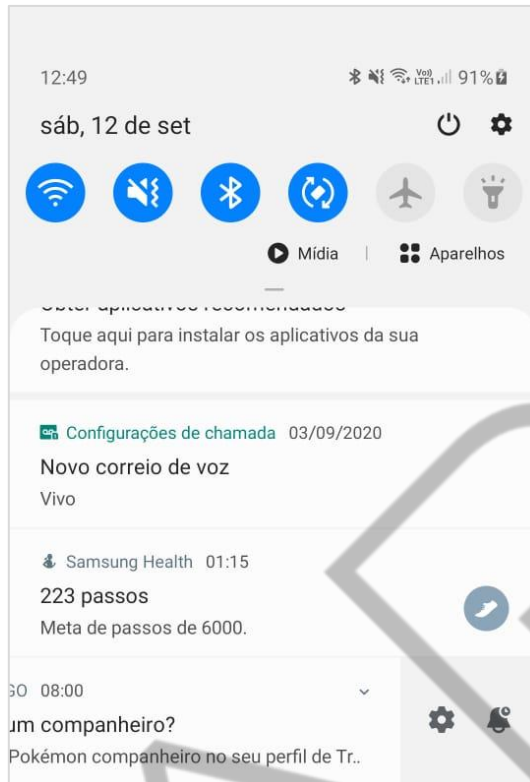


Figura 4.16 - Visualizar os canais da notificação/silenciar notificação
Fonte: Elaborado pelo autor (2020)

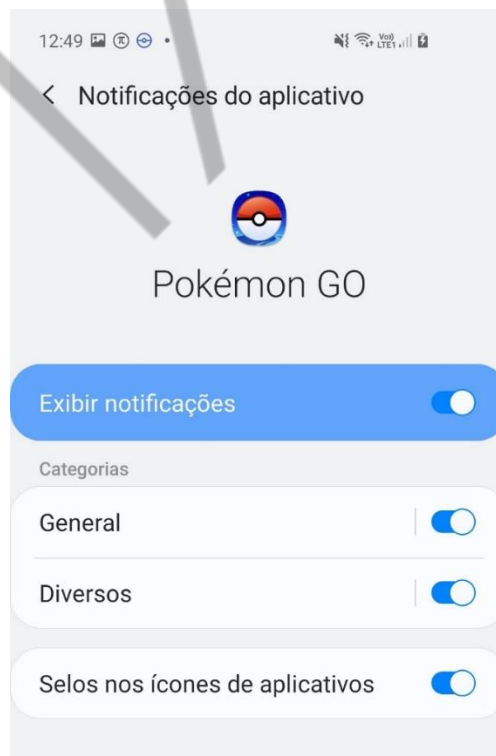


Figura 4.17 - Canais de notificação de um aplicativo e seu status
Fonte: Elaborado pelo autor (2020)

No código do tópico anterior foi criado um canal por meio 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)
}

```

Código-fonte 4.10 – Criação de um canal de notificação
 Fonte: Elaborado pelo autor (2020)

Esse método só será chamado se a aplicação estiver sendo executada em aparelhos com o Android Oreo (API 26) ou superior. Nesse método, será criado o canal por meio da classe **NotificationChannel** por meio dos parâmetros id, nome e importância. Veja a tabela com os níveis de importância de um canal de notificações:

Tabela 4.2 – Criação 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.

Fonte: GLAUBER (2019)

A classe **NotificationManager** fornece métodos para gerenciar os canais de notificação, em que 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)**, cujo parâmetro enviado é o canal que deseja apagar.

4.2.3 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 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.

Exemplo de utilização:

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

Código-fonte 4.11 – Criação de um canal de notificação
Fonte: Elaborado pelo autor (2020)

É possível definir uma notificação para ser exibida na tela de bloqueio e outra para ser exibida na barra de notificações, para isso, utilize o método **setPublicVersion**.

As notificações são fortes aliadas para engajar o usuário, para uma melhor experiência, existem alguns tipos que podem ser utilizados.

4.2.4 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.

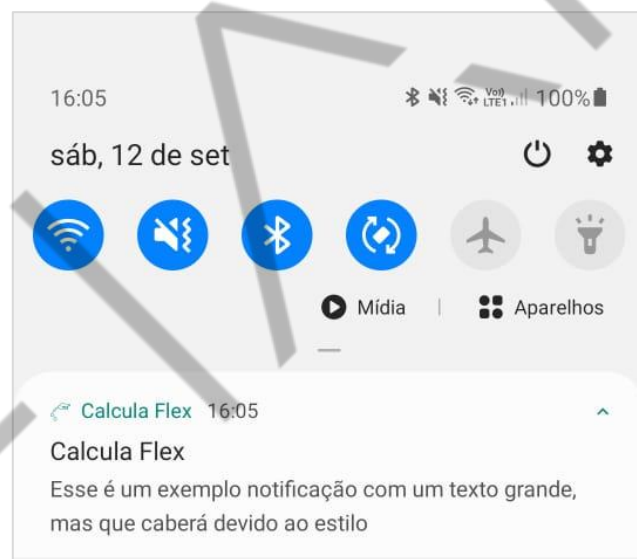


Figura 4.18 – Notificação com texto longo
Fonte: Elaborado pelo autor (2020)

Veja um exemplo no código-fonte “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())
}

```

Código-fonte 4.12 – Notificação com Texto Longo
Fonte: GLAUBER (2019)

4.2.5 Notificação com botões de ação

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

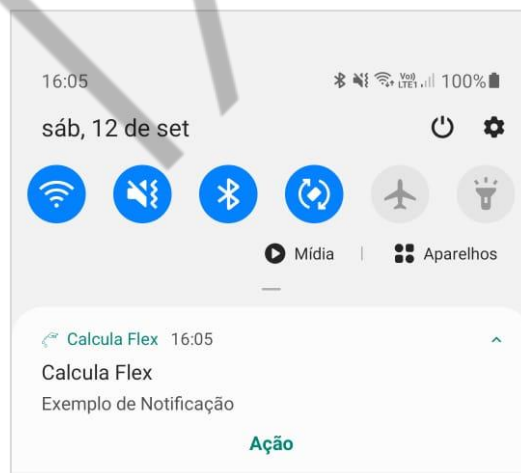


Figura 4.19 - Notificação com botões de ação
Fonte: Elaborado pelo autor (2020)

Veja um exemplo de implementação no Código-fonte “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())  
}
```

Código-fonte 4.13 – Notificação com botões de ação
Fonte: GLAUBER (2019)

4.2.6 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.

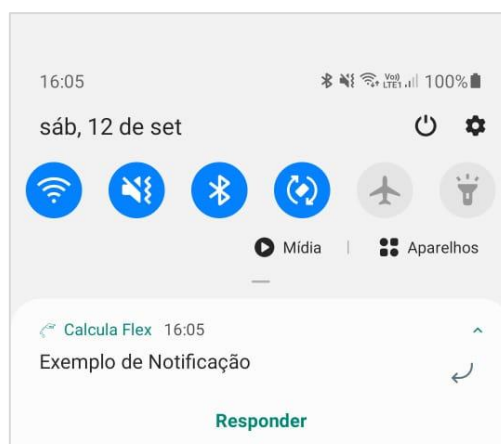


Figura 4.20 - Notificação com digitação de texto botão de ação

Fonte: Elaborado pelo autor (2020)

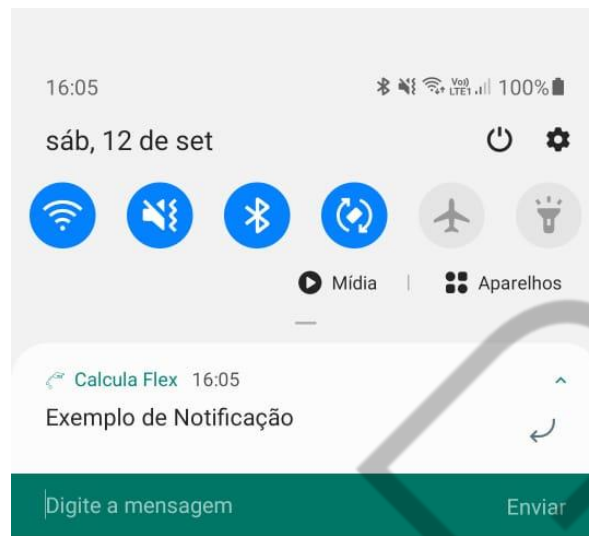


Figura 4.21 - Notificação com digitação de texto resposta
Fonte: Elaborado pelo autor (2020)

Exemplo de implementação:

```
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)
    }
}
```

Código-fonte 4.14 – Notificação com digitação de texto
Fonte: GLAUBER (2019)

4.2.7 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.

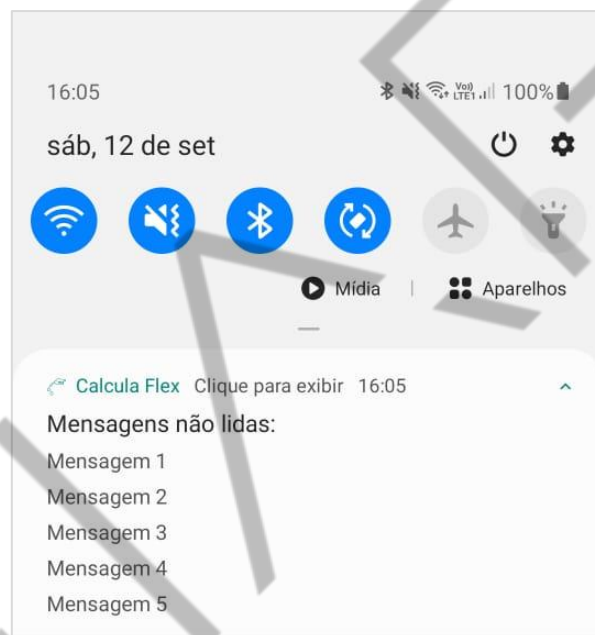


Figura 4.22 - Notificações agrupadas
Fonte: Elaborado pelo autor (2020)

Veja um exemplo de implementação no Código-fonte “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))  
    }  
}
```

```

    }

    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())
}

```

Código-fonte 4.15 – Notificações agrupadas
 Fonte: GLAUBER (2019)

4.3 Projeto de notificações

Neste repositório <https://github.com/FIAPON/TiposNotificacoes> está disponível um exemplo de implementação com os tipos de notificações.

Agora, vamos implementar as notificações em nosso projeto. Abra o arquivo **CalculaFlexFCMService** e adicione o seguinte código:

```

class CalculaFlexFCMService : FirebaseMessagingService() {

    override fun onMessageReceived(p0: RemoteMessage) {
        val intent = Intent(this, MainActivity::class.java)

        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)
}
}
```

Código-fonte 4.16 – Classe para exibir a notificação enviada pelo FCM
Fonte: Elaborado pelo autor (2020)

4.3.1 Envio de Notificações pelo Firebase

Abra o painel do Firebase do seu projeto (<https://console.firebase.google.com>).
Selecione a opção **Cloud Messaging** (Menu Ampliar).

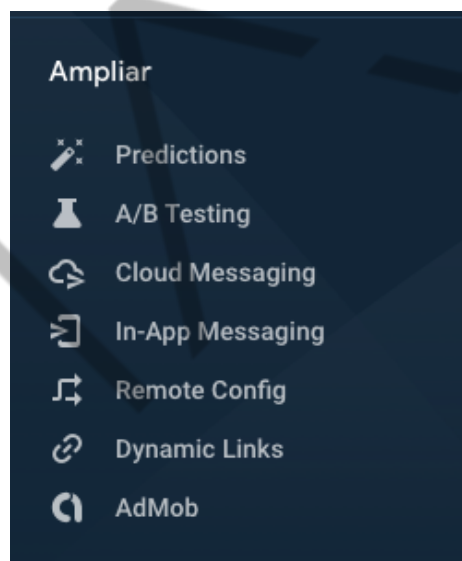


Figura 4.23 - Painel Ampliar do Firebase
Fonte: Elaborado pelo autor (2020)

Clique no botão **Send your first message**.

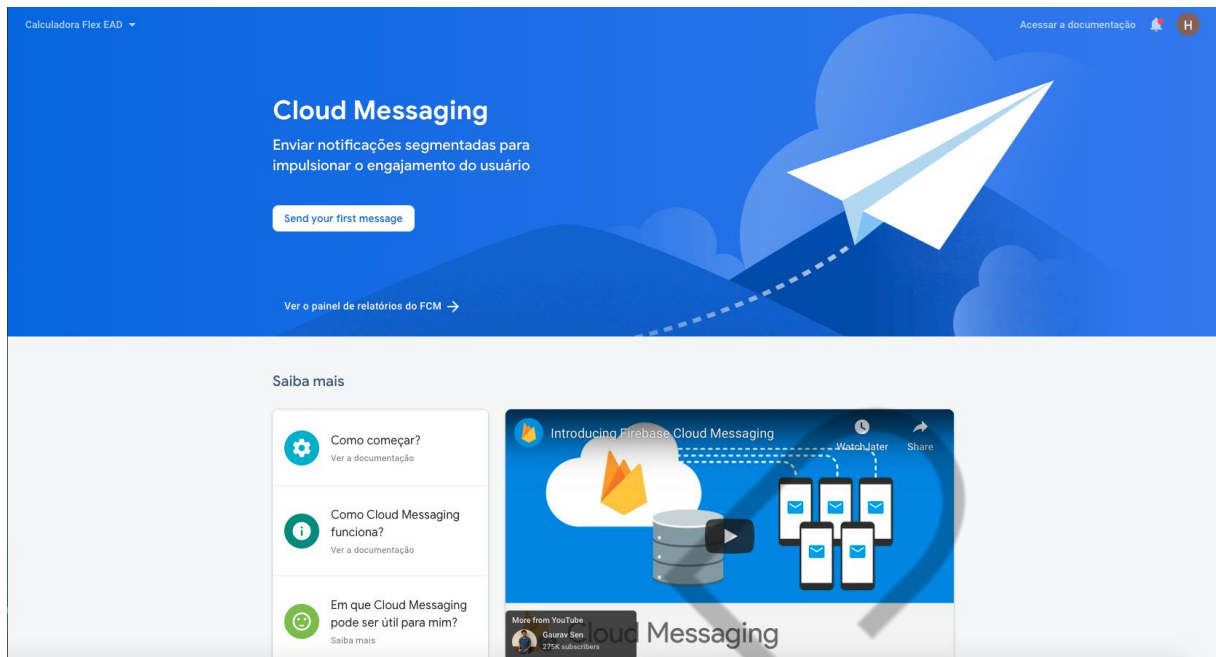


Figura 4.24 - Enviando a primeira mensagem 1

Fonte: Elaborado pelo autor (2020)

Preencha os campos que serão enviados na notificação:

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

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

Imagem 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.

1

Notificação

Título da notificação ?

Calcula Flex

Texto da notificação

Você foi sorteado para ganhar um abastecimento gratuito no seu veículo

Imagem de notificação (opcional) ?

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

Nome da notificação (opcional) ?

Informa nome opcional

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.

Enviar mensagem de teste

Estado inicial

Visualização expandida

Calcula Flex

Você foi sorteado para ganhar um abastecimento gratuit...

Android

Calcula Flex

Você foi sorteado para ganhar um abastecimento grat...

iOS

Figura 4.25 - Enviando a primeira mensagem 2
Fonte: Elaborado pelo autor (2020)

Após preencher os campos da notificação, clique em **Próxima** para preencher a segmentação. Nesse ponto, você poderá selecionar a qual público se destina essa notificação. No exemplo da figura “Segmentação de usuários para receber a notificação”, a notificação será enviada para todos os usuários da aplicação.

The screenshot shows the 'Segmentação' (Segmentation) step in the Firebase console. It features a progress bar with four steps: 1. Notificação (Notification), 2. Segmentação (Segmentation), 3. Programação (Scheduling), and 4. Outras opções (opcional) (Other options (optional)). The 'Segmentação' step is currently active. Below the progress bar, there is a section titled 'Segmentar usuário se...' (Segment user if...). It includes a dropdown menu labeled 'App' with the value 'br.com.heiderlopes.calculadoraflex' selected. There is also a button labeled 'Segmentar outro app' (Segment another app). A blue 'Próxima' (Next) button is located below the segmentation options.

Figura 4.26 - Segmentação de usuários para receber a notificação
Fonte: Elaborado pelo autor (2020)

Após definir a segmentação, o próximo passo é definir o momento do envio. Você pode enviar imediatamente ou agendar.

The screenshot shows the 'Programação' (Scheduling) step in the Firebase console. It features a progress bar with four steps: 1. Notificação (Notification), 2. Segmentação (Segmentation), 3. Programação (Scheduling), and 4. Outras opções (opcional) (Other options (optional)). The 'Programação' step is currently active. Below the progress bar, there is a section titled 'Enviar para usuários qualificados' (Send to qualified users). It includes a dropdown menu with the value 'Agora' (Now) selected. A blue 'Próxima' (Next) button is located below the scheduling options.

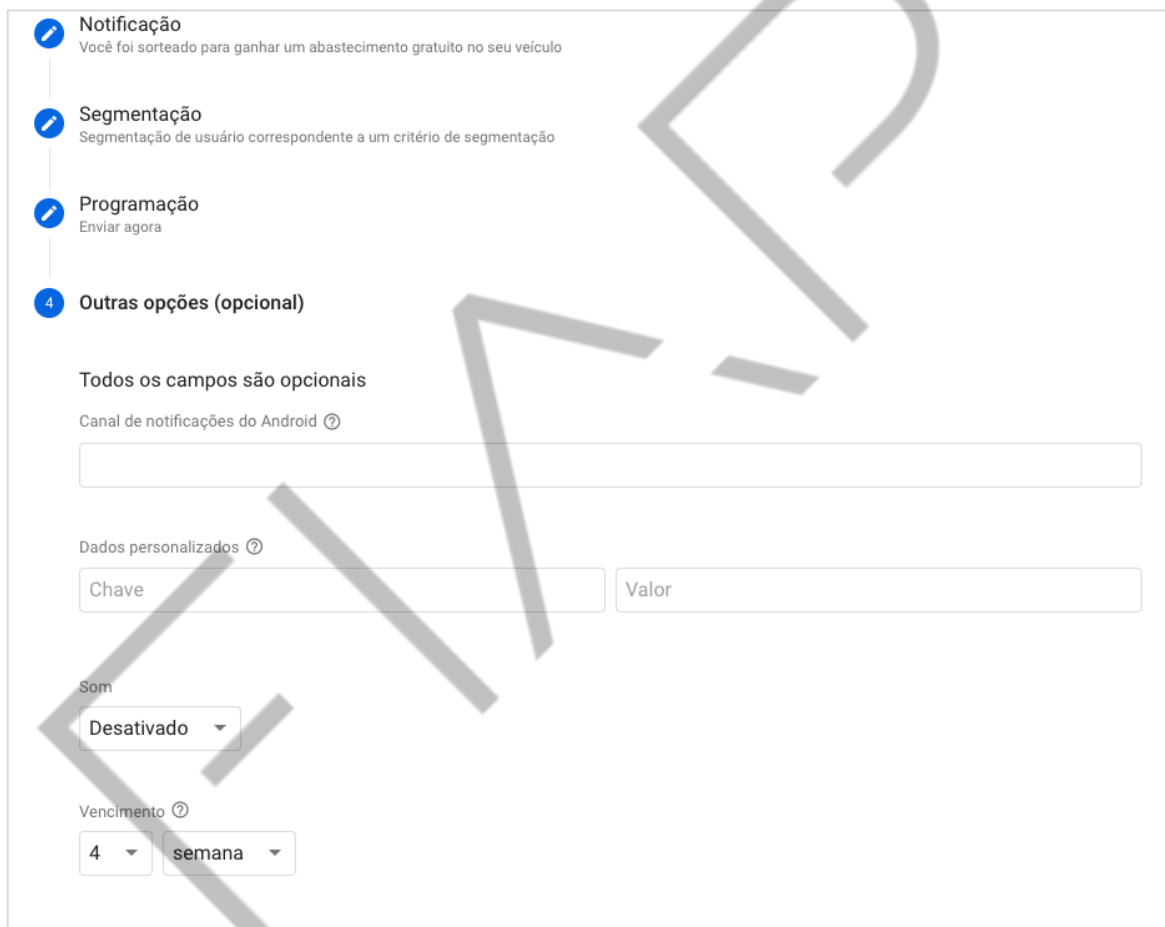
Figura 4.27 - Programando o envio da mensagem
Fonte: Elaborado pelo autor (2020)

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.



O formulário apresenta uma barra lateral com quatro itens: 'Notificação' (com ícone de lápis), 'Segmentação' (com ícone de lápis), 'Programação' (com ícone de lápis) e 'Outras opções (opcional)' (com ícone de círculo contendo o número 4). Abaixo da barra lateral, há o texto 'Todos os campos são opcionais'. O formulário contém os seguintes campos: 'Canal de notificações do Android' (com um ícone de ajuda), um campo de texto vazio; 'Dados personalizados' (com um ícone de ajuda), seguido por uma tabela com duas colunas: 'Chave' e 'Valor', ambas com campos de texto vazios; 'Som', com um menu suspenso selecionando 'Desativado'; e 'Vencimento' (com um ícone de ajuda), com dois menus suspensos selecionando '4' e 'semana'.

Figura 4.28 - Dados opcionais da notificação 1
Fonte: Elaborado pelo autor (2020)

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**

Figura 4.29 - Dados opcionais da notificação 2
Fonte: Elaborado pelo autor (2020)

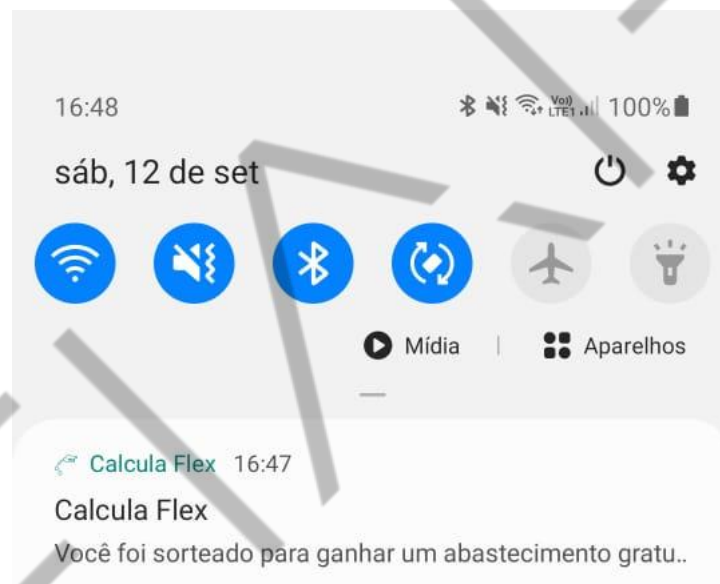


Figura 4.30 - Exibição da notificação
Fonte: Elaborado pelo autor (2020)

4.3.2 Disparando a mensagem via API

Por meio do Postman é possível realizar a chamada da API Rest. Dessa mesma forma, é possível integrar seu backend com o **Firebase** e disparar uma notificação para o usuário diretamente do seu back-end.

Postman: é uma ferramenta para enviar requisições web (Restful) para qualquer API. Permite configurar parâmetros, autenticação e escolher métodos diferentes para envio (GET, POST, PATCH, DELETE, etc.). É bastante utilizado em testes

de integração. Você pode baixar gratuitamente por este link:
<<https://www.postman.com/>..

No método **onNewToken** da classe **CalculaFlexFCMService**, você terá o token sempre que for atualizado. Nesse 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 ter 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)  
}
```

Código-fonte 4.17 – Exibição da notificação
Fonte: Elaborado pelo autor (2020)

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 imagem da figura “Token do usuário exibido no Logcat do Android Studio”.

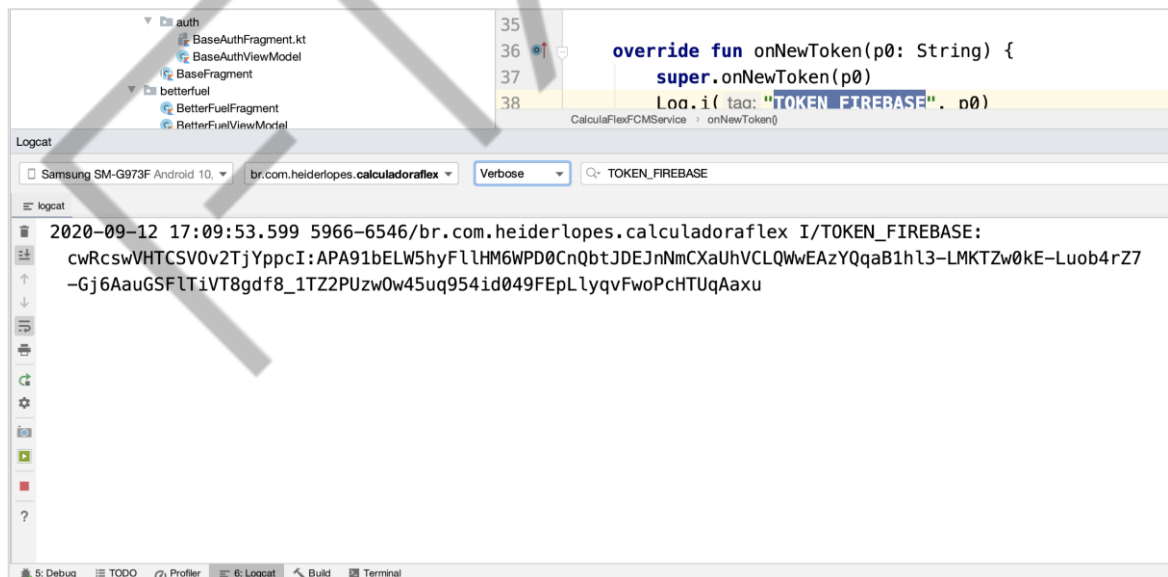


Figura 4.31 - Token do usuário exibido no Logcat do Android Studio
Fonte: Elaborado pelo autor (2020)

Agora, vá até o console do Firebase e acesse a configuração do projeto. Clique na engrenagem ao lado da **Visão geral do projeto**.

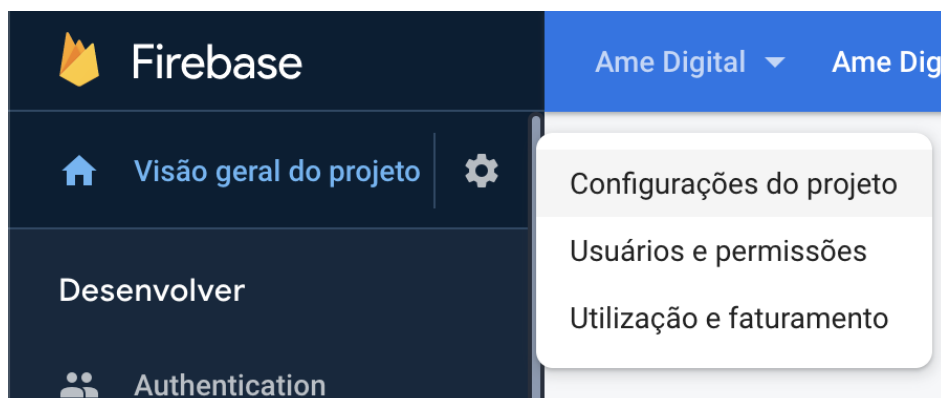


Figura 4.32 - Acessando as configurações do projeto
Fonte: Elaborado pelo autor (2020)

Clique sobre a opção **Cloud Messaging** e copie a chave do Servidor.



Figura 4.33 - Obtendo a chave do servidor
Fonte: Elaborado pelo autor (2020)

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**.

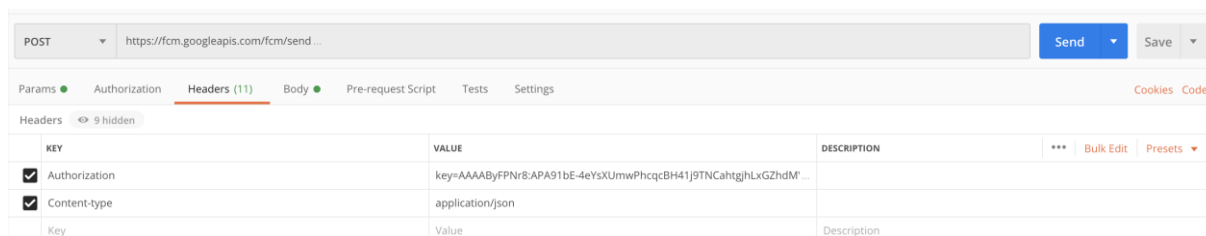


Figura 4.34 - Adicionando os headers na request
Fonte: Elaborado pelo autor (2020)

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"  
  }  
}
```

Código-fonte 4.17 – Payload do FCM
Fonte: Elaborado pelo autor (2020)

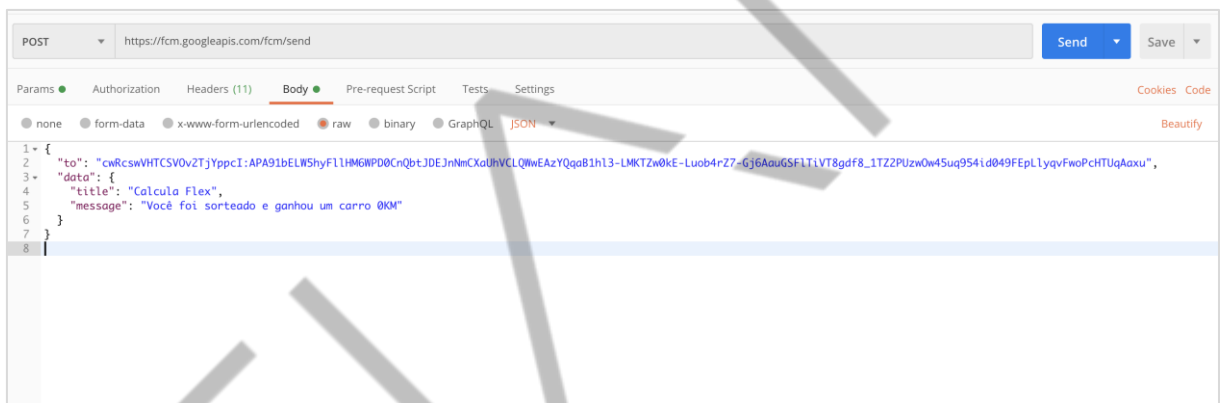


Figura 4.35 - Configuração para enviar a mensagem
Fonte: Elaborado pelo autor (2020)

Após a configuração da notificação, clique em **Send** para enviar a mensagem.

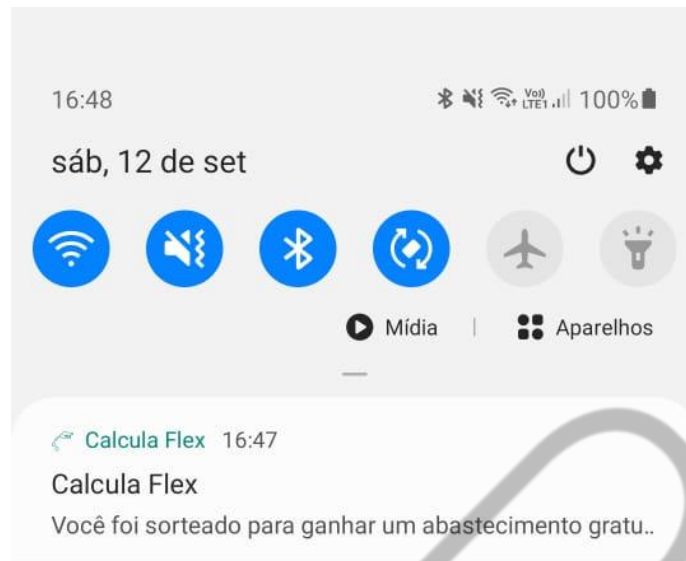


Figura 4.36 - Notificação enviada
Fonte: Elaborado pelo autor (2020)

4.4 Deeplinks

O Deep Link é usado para aplicativos móveis, possibilita que os usuários explorem outras áreas do app e naveguem por lugares específicos por um comando, no caso, os links direcionados.

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

Outro exemplo muito recorrente é o 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 pelos quais 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 vêm 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.

Os usuários podem personalizar a experiência que têm dentro do app, já que o desenvolvedor com o deep link pode fazer um convite personalizado, criando 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, a pessoa entrará na página certa já para consumir o produto.

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

Além de trazer mais visibilidade para o app, traz também usuários orgânicos, ou seja, com muito potencial de engajamento.

4.4.1 Adicionando deeplink no projeto

Abra o arquivo **main_nav_grap.xml** no modo design. Selecione o 42ragmente **betterFuelFragment**:

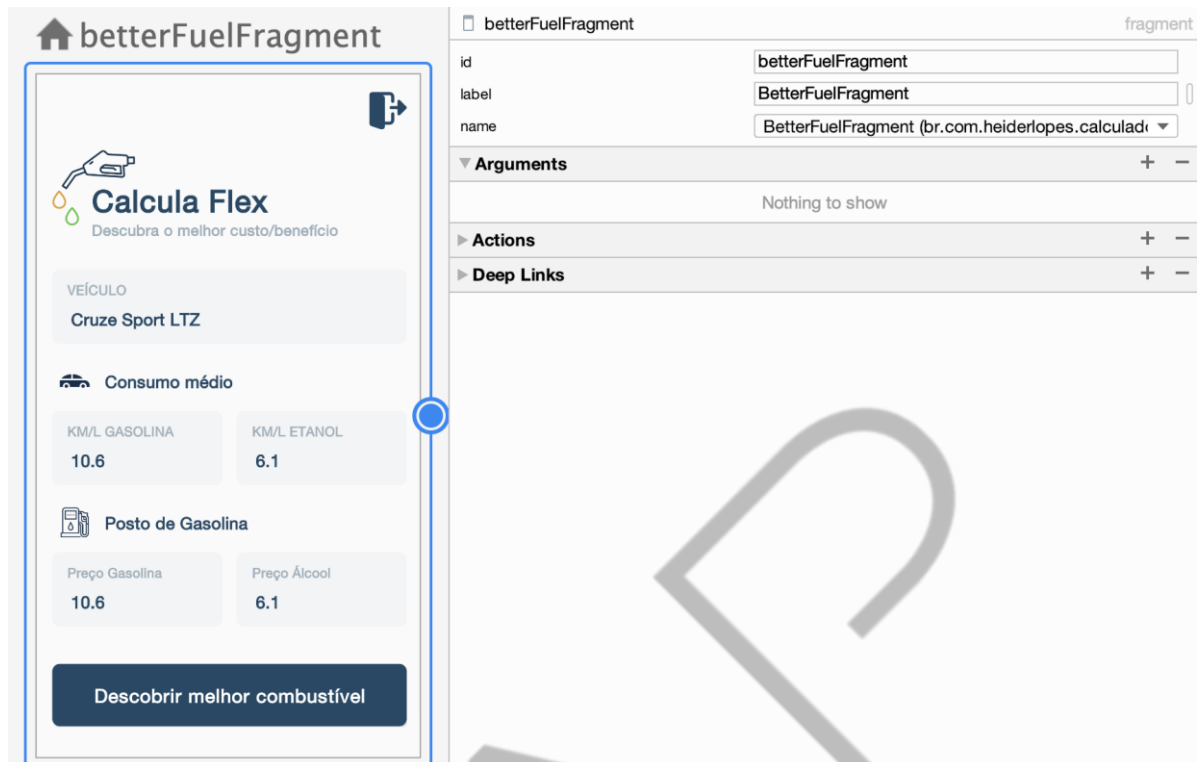


Figura 4.37 – Selecionando o BetterFuelFragment
Fonte: Elaborado pelo autor (2020)

Selecione o botão **+** da opção **Deep Links**. Adicione a URI que será utilizada como deep link:

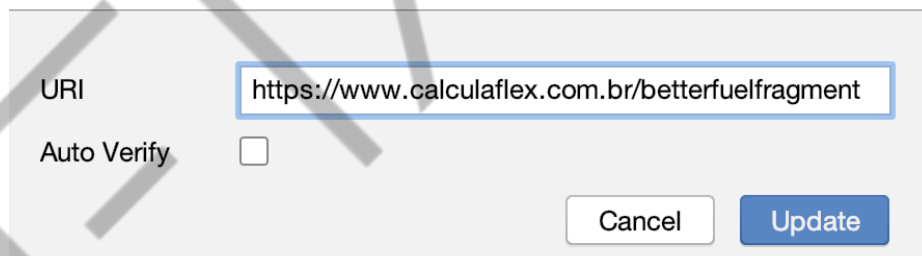


Figura 4.38 – Criando o deep link para tela de Melhor Combustível
Fonte: Elaborado pelo autor (2020)

Agora, selecione o SignUpFragment e adicione o seguinte deep link para ele:

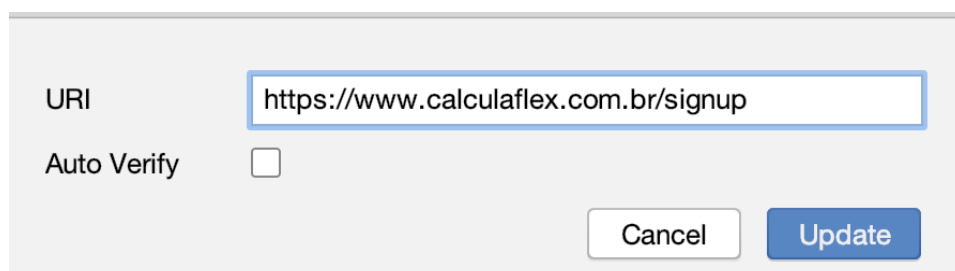


Figura 4.39 – Criando o deep link para tela de SignUp
Fonte: Elaborado pelo autor (2020)

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

Adicione o elemento **<nav-graph>** a uma activity que aponta para um gráfico de navegação existente, como mostrado no código-fonte “Adição do deep link”:

```
<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>
```

Código-fonte 4.18 – Adição do deeplink
Fonte: Elaborado pelo autor (2020)

4.4.2 Testando o deep link pelo Android Studio

Crie uma configuração de execução. Primeiro, selecione a opção **Edit Configurations ...**

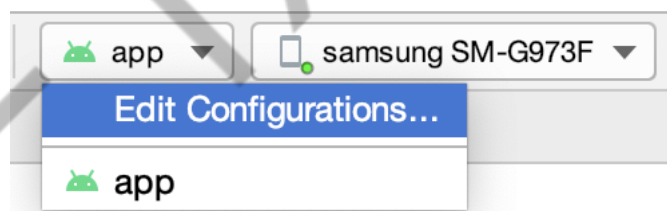


Figura 4.40 - Criando uma configuração
Fonte: Elaborado pelo autor (2020)

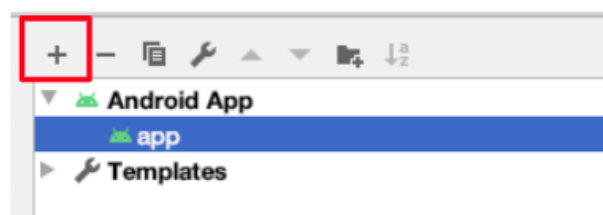


Figura 4.41 - Criando uma configuração
Fonte: Elaborado pelo autor (2020)

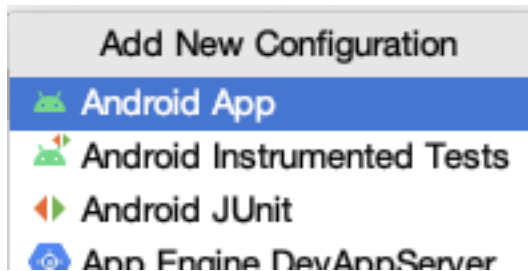


Figura 4.42 - Criando uma configuração
Fonte: Elaborado pelo autor (2020)

Em Launch Options, altere o modo Launch para URL. Em seguida, adicione a seguinte URL: **<https://www.calculaflex.com.br/betterfuel>**.

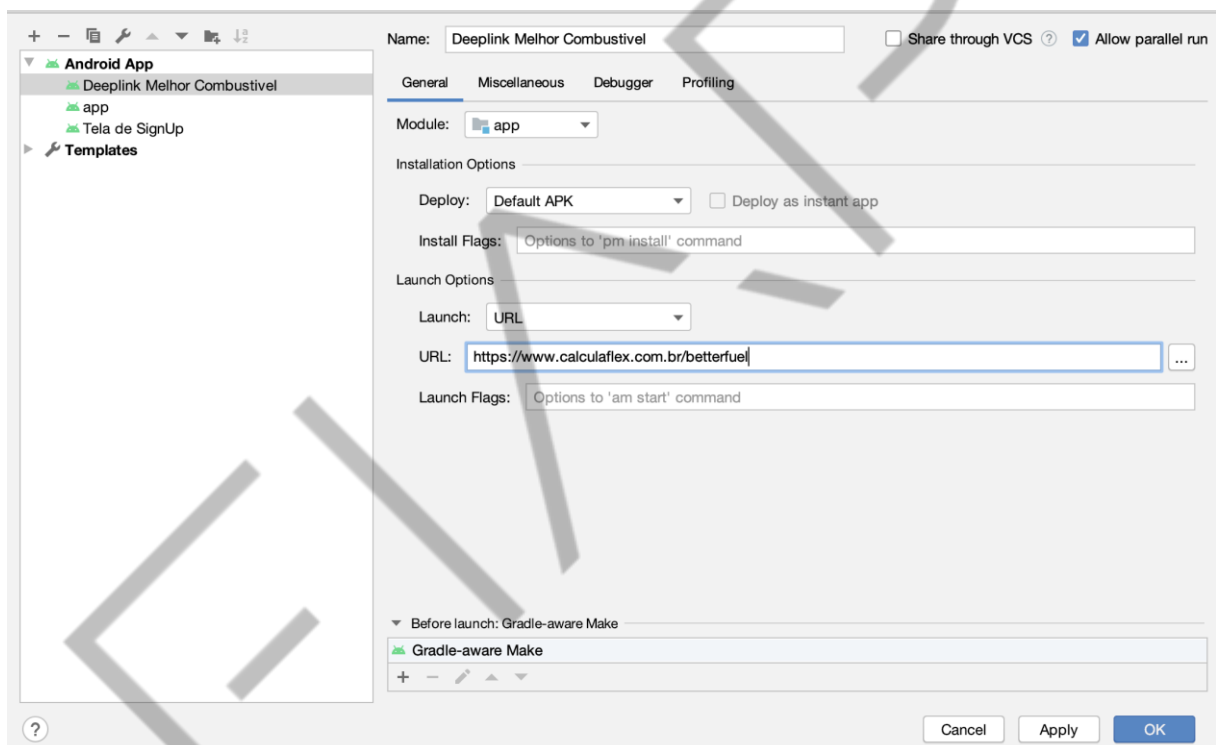


Figura 4.43 - Configurando Launch de Deeplink para tela de melhor combustível
Fonte: Elaborado pelo autor (2020)

Crie nova configuração de execução para a tela de SignUp. Primeiro, selecione a opção **Edit Configurations ...**

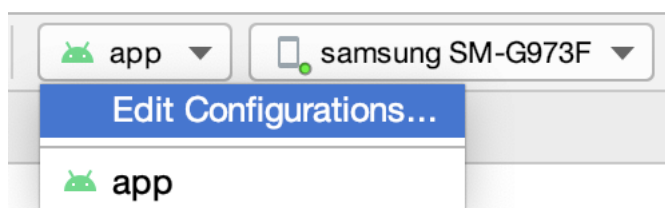


Figura 4.44 - Criando uma configuração
Fonte: Elaborado pelo autor (2020)

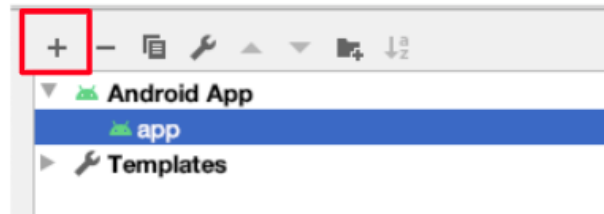


Figura 4.45 - Criando uma configuração
Fonte: Elaborado pelo autor (2020)

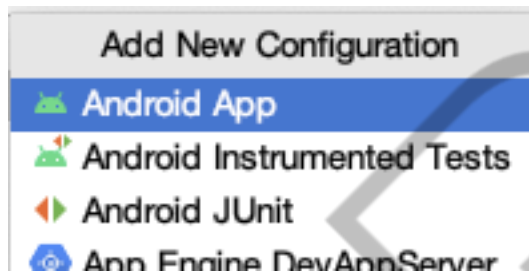


Figura 4.46 - Criando uma configuração
Fonte: Elaborado pelo autor (2020)

Em Launch Options, altere o modo Launch para URL. Em seguida, adicione a seguinte URL: <https://www.calculaflex.com.br/signup>.

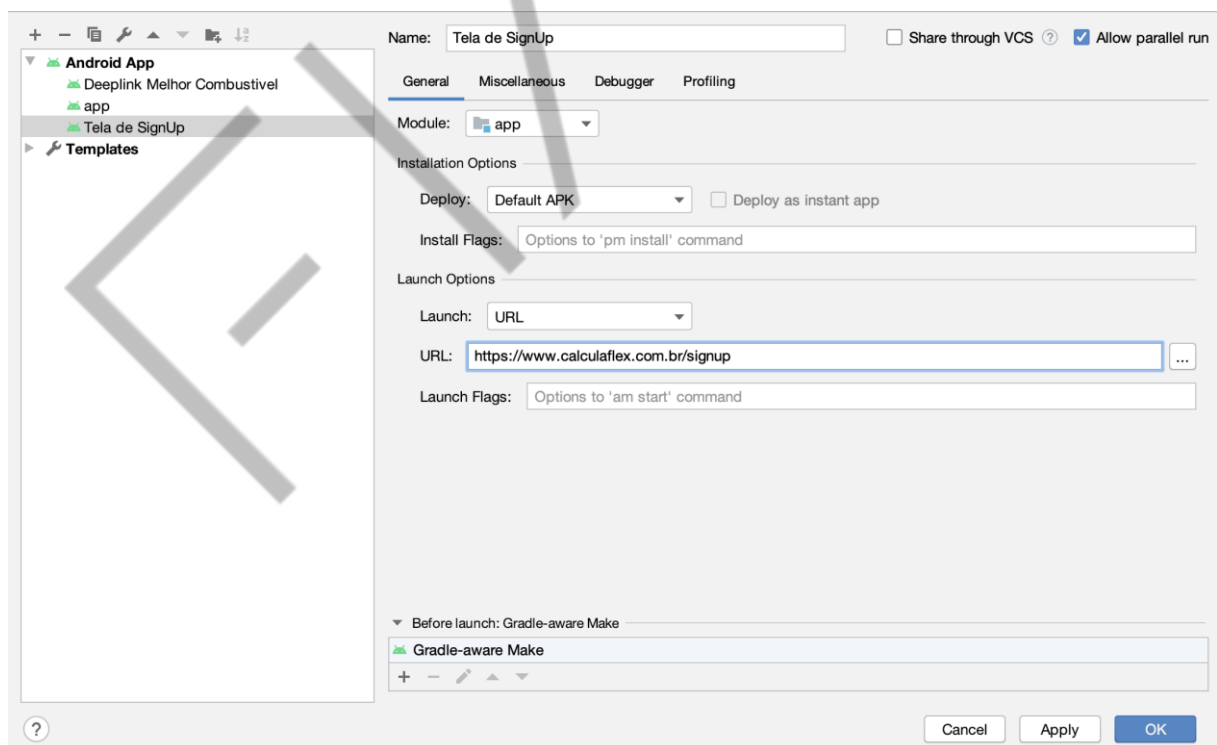


Figura 4.47 - Configurando Launch de Deeplink para tela de criação de conta
Fonte: Elaborado pelo autor (2020)

Execute a aplicação utilizando as configurações criadas e observe que a aplicação abriu de acordo com o deep link informado.

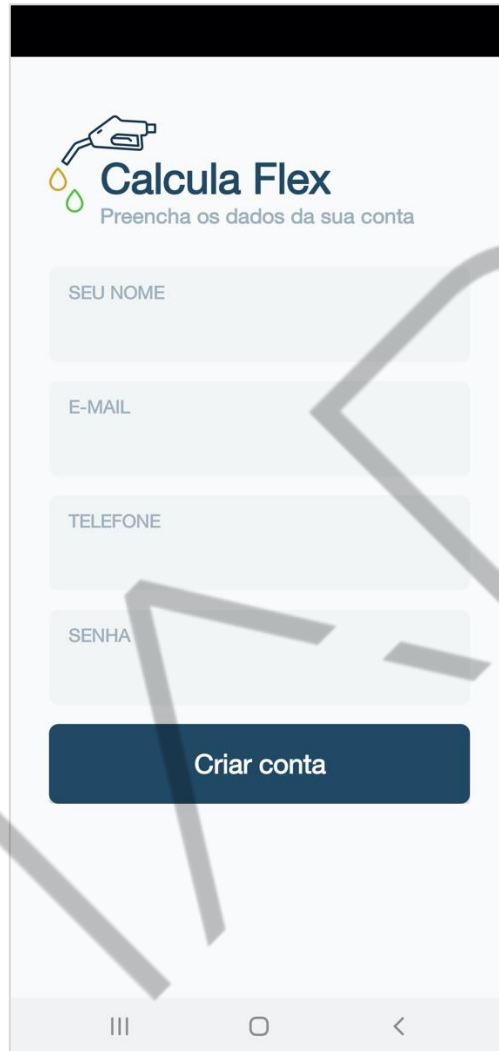


Figura 4.48 - Aplicativo sendo executado direto para tela do Deeplink
Fonte: Elaborado pelo autor (2020)

4.4.3 Abrindo deeplink via Push Notification

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**.

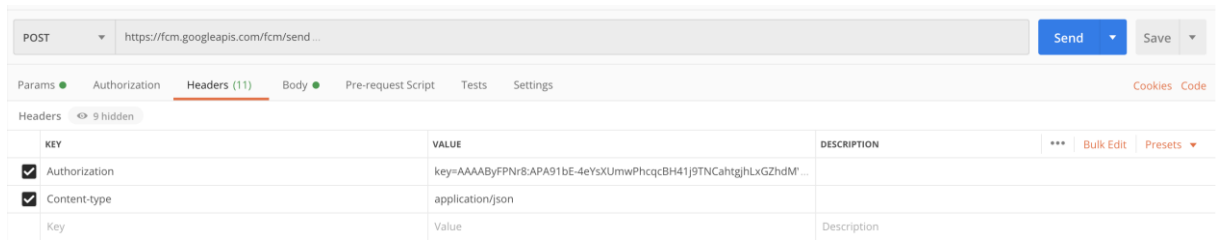
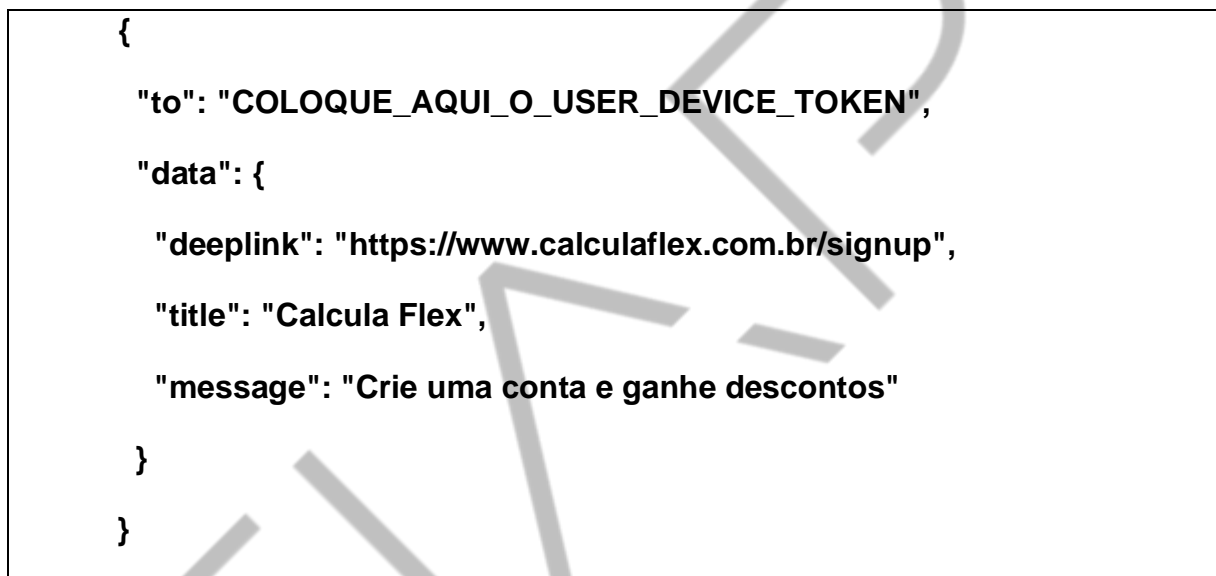


Figura 4.49 - Adicionando os headers na request
Fonte: Elaborado pelo autor (2020)

Clique agora sobre a aba **Body** e adicione o objeto que será enviado para o usuário:



Código-fonte 4.19 – Payload do FCM
Fonte: Elaborado pelo autor (2020)

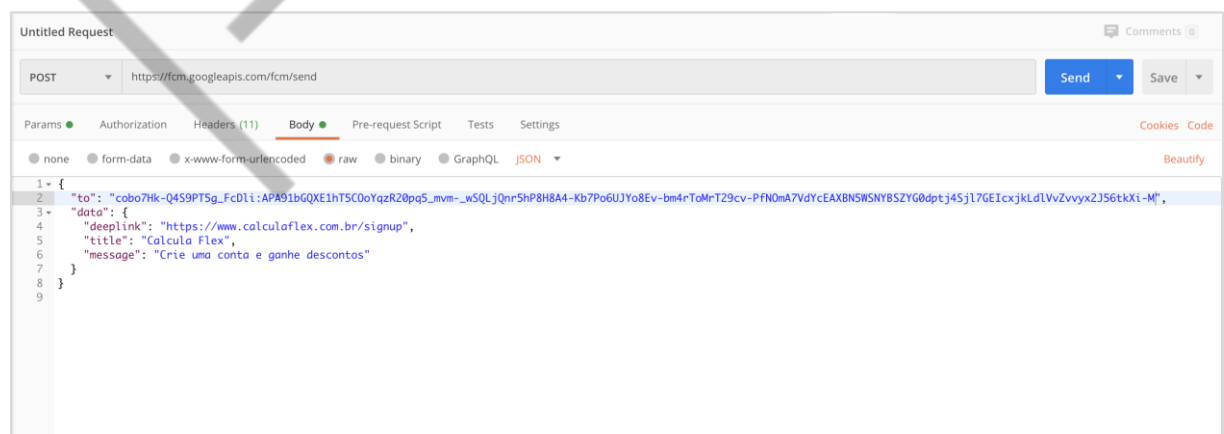


Figura 4.50 - Configuração para enviar a mensagem
Fonte: Elaborado pelo autor (2020)

Após a configuração da notificação, clique em **Send** para enviar a mensagem.

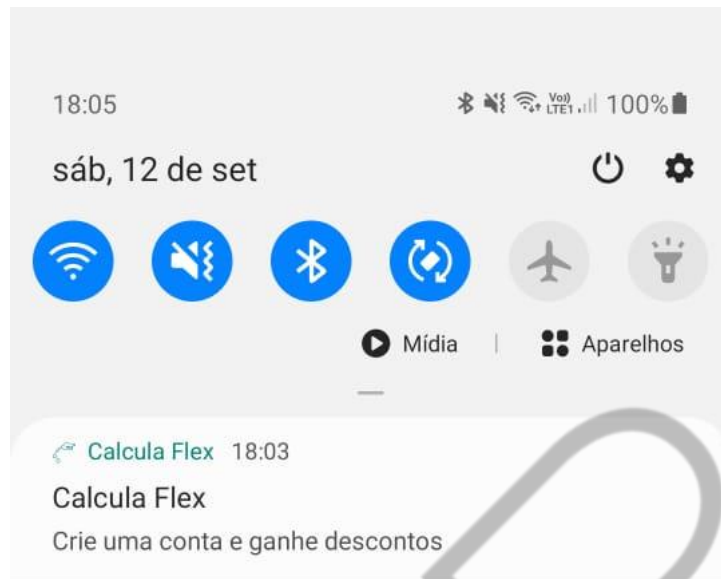


Figura 4.51 - Notificação enviada
Fonte: Elaborado pelo autor (2020)

Porém, se, ao clicar na notificação, e a aplicação não abrir, o que aconteceu? É necessário configurar a ação da notificação.

Para isso, abra a classe **NotificationUtils** e receba a intent que deverá ser executada via parâmetro e adicione ao 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)  
    }  
    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())
}

```

Código-fonte 4.20 – Notificação enviada 1
 Fonte: Elaborado pelo autor (2020)

Feito isso, adicione a verificação na classe **CalculaFlexFCMService**, conforme o código abaixo:

```

class CalculaFlexFCMService : FirebaseMessagingService() {

    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
        )

        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)
}

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

Código-fonte 4.21 – Notificação enviada 2
Fonte: Elaborado pelo autor (2020)

Pronto. Feita a alteração, envie novamente a notificação e, ao clicá-la, você será redirecionado para a tela de criação de conta.

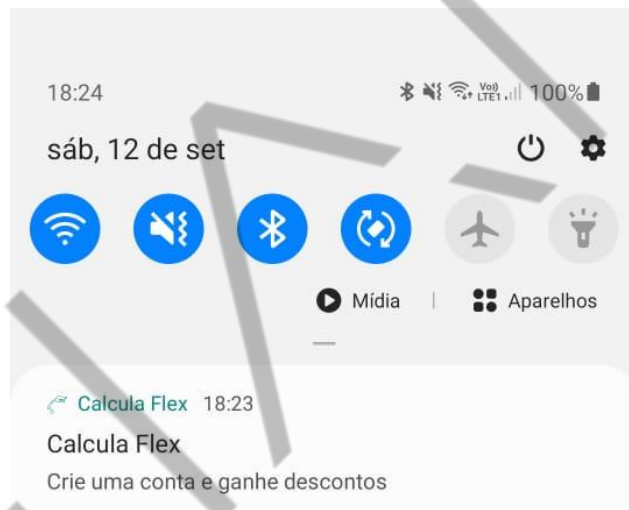


Figura 4.52 - Exibição da notificação com deep link
Fonte: Elaborado pelo autor (2020)

O projeto completo pode ser baixado neste repositório
<<https://github.com/FIAPON/CalculaFlexAndroid>>.

CONCLUSÃO

Neste capítulo, foi possível conhecer o Firebase Cloud Messaging e o poder que ele agrega para engajar o usuário no seu aplicativo. Lembre-se: usando com moderação, seu usuário poderá aproveitar cada vez mais as funcionalidades que seu aplicativo pode fornecer. Utilizando o deep link com FCM, você consegue direcionar o seu usuário para telas que talvez ele desconheça e, assim, estimular o uso, pois ele pode não estar recebendo nenhum estímulo ou pode não ter conhecimento de determinadas funcionalidades.

REFERÊNCIAS

GLAUBER, N. **Dominando Android com Kotlin**. São Paulo: Novatec, 2019.

GOOGLE. **Documentação do Firebase**. 2020. Disponível em:
<<https://firebase.google.com/docs/cloud-messaging/android/receive?hl=pt-b>>.
Acesso em: 10/09/2020.

EXEMPLO