

Passo 1 = Premissa

A premissa do nosso app é ter inicialmente apenas um botão que executa uma função que gera uma notificação para o usuário. Após entender o funcionamento deste passo iremos integrá-lo com o Firebase para que recebamos notificações de forma remota. Sabendo que por padrão com o aplicativo aberto as notificações enviadas pelo FCM¹ não são exibidas, iremos definir um serviço para que a mesma seja criada e exibida mesmo com o app aberto.

Passo 2 = Definição

Vamos dividir a construção do nosso aplicativo em duas partes. Na primeira criaremos uma função para que uma notificação seja gerada e exibida ao clique de um botão. Na segunda vamos conectar o app com o firebase e adicionar o FCM¹, em seguida criaremos o serviço do firebase para que a notificação seja criada quando o app estiver em primeiro plano.

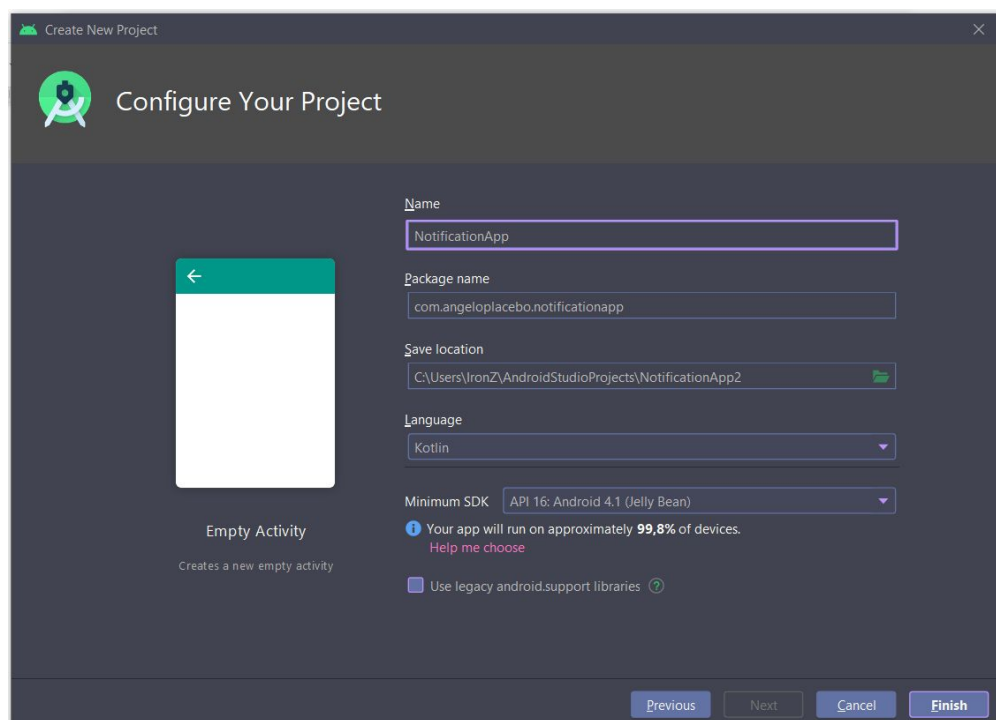
Passo 3 = Parte 1

Name: NotificationApp

Package name: com.angeloplacebo.notificationapp

Language: Kotlin

Minimum SDK: API 16 Android 4.1 (Jelly Bean)



¹ "Firebase Cloud Messaging - Google." <https://firebase.google.com/docs/cloud-messaging?hl=pt-br>. Acessado em 25 out.. 2020.

Figura 1. Tela de criação do projeto

Vamos alterar o layout para `LinearLayout`², definir orientação vertical e gravity³ como center e criar um botão com id `btNotificar` conforme a figura 2.

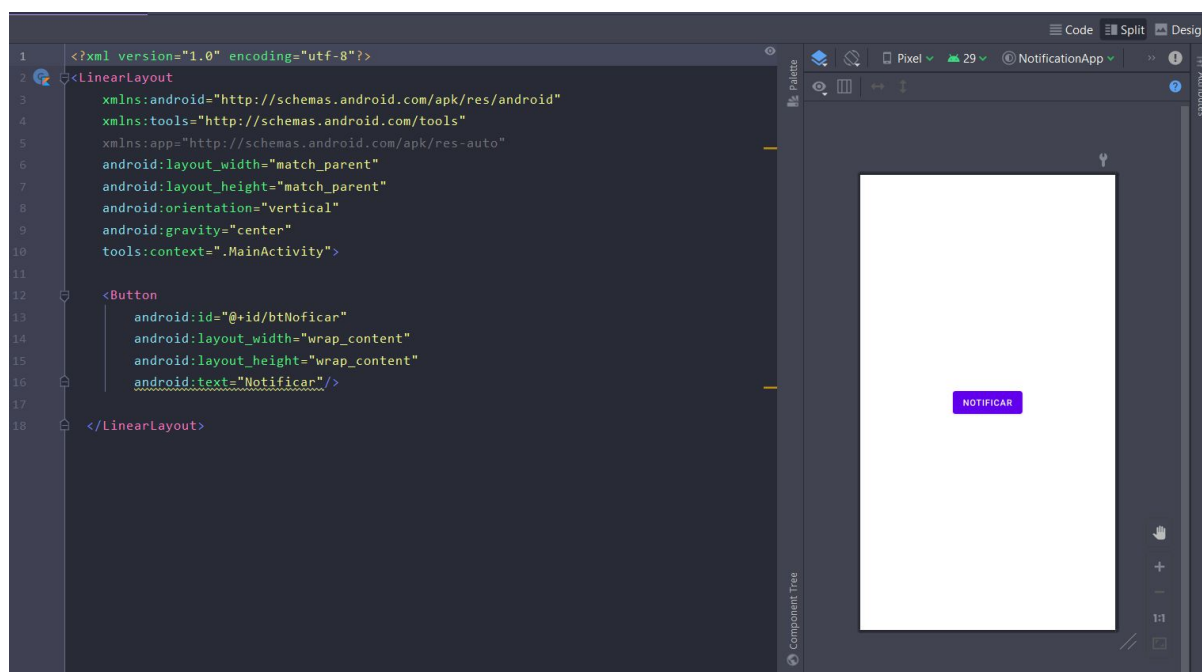


Figura 2. Layout do projeto

Em seguida iremos criar uma variável `lateinit`⁴ para este botão e resgatá-lo utilizando o método `findViewById` conforme Figura 3.

² "LinearLayout | Android Developers."

<https://developer.android.com/reference/android/widget/LinearLayout>. Acessado em 25 out.. 2020.

³ "Gravity | Android Developers." 4 ago.. 2020,

<https://developer.android.com/reference/kotlin/android/view/Gravity>. Acessado em 25 out.. 2020.

⁴ "Quando utilizar inicialização lazy do Kotlin | by Alex Felipe" 20 ago.. 2018,

<https://medium.com/android-dev-br/quando-utilizar-inicializa%C3%A7%C3%A3o-lazy-do-kotlin-45377c8b68ad>. Acessado em 25 out.. 2020.

```

class MainActivity : AppCompatActivity() {

    private lateinit var btNotificar : Button

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        btNotificar=findViewById(R.id.btNoficar)
    }
}

```

Figura 3. MainActivity

Em seguida vamos criar a notificação através de uma função que chamaremos de *criarNotificacao*, nela definiremos primeiramente as variáveis título, descrição e corpo.

Para que o título fique negrito usaremos a classe *htmlCompat*.(Figura 4.).

Precisamos criar o canal de notificação que por sua vez necessita de um id para o canal que será nada mais que o nome do nosso pacote, definiremos também a variável builder que é um objeto *NotificationCompat.Builder*. Um gerenciador de notificações também é necessário nele passamos o serviço de notificações do sistema, além disso uma notificação precisa de uma *pendingIntent* para ser iniciada no evento de clique bem como uma id única para cada notificação gerada assim um número aleatório entre 1 e 100 será gerado através da função *nextInt* da classe *Random*.(Figura 4.)

```

fun criarNotificacao(){
    val titulo :Spanned = HtmlCompat.fromHtml(
        source: "<strong> Titulo da notificação</strong>",
        HtmlCompat.FROM_HTML_MODE_LEGACY)

    val descricao = "Descricao da notificação"
    val corpo = "Corpo da notificação"

    val notificationChannel : NotificationChannel
    val builder : NotificationCompat.Builder
    val channelId = "com.angeloplacebo.notificationapp"

    val notificationManager : NotificationManager = applicationContext.getSystemService(
        Context.NOTIFICATION_SERVICE) as NotificationManager

    val id :Int = Random.nextInt( from: 1, until: 100)

    val intent = Intent( packageContext: this, MainActivity::class.java)
    val pendingIntent :PendingIntent! = PendingIntent.getActivity(
        context: this, requestCode: 0,intent, flags: 0)
}

```

Figura 4. Variáveis da função

Antes de uma notificação ser entregue é preciso registrar o canal de notificação sendo assim o trecho a seguir é bloqueado pela condição do SDK_INT pois este canal deverá ser criado apenas se estivermos utilizando a API de level 26 ou superior devido a esta classe ser nova versões anteriores não a suportam, assim versões anteriores ignoram o channelId do método Builder da classe NotificationCompat.

Para criar o canal alguns parâmetros são exigidos no construtor do NotificationChannel passamos o channelId a descrição e a importância através de NotificationManager.IMPORTANCE_DEFAULT) em seguida passamos este objeto para o gerenciador de notificações no método createNotificationChannel.(Figura 5.).

Utilizando nosso objeto Builder da classe NotificationCompat.Builder vamos definir o conteúdo da nossa notificação. Na figura 5 você verá os seguintes métodos:

- **setAutoCancel** = usado para fechar a notificação no evento de clique.
- **setSmallIcon** = define o ícone pequeno que é mostrado no cabeçalho da notificação.
- **setContentTitle** = define o título da notificação.
- **setContentText** = define o conteúdo(corpo) da notificação.
- **setPriority** = A prioridade determina se a notificação será intrusiva ou não no Android 7.1 e versões anteriores.
- **setContentIntent** = define a activity que será executada no evento de clique.

seguidamente iremos exibir a notificação, para isso chamaremos nosso objeto NotificationManager.notify, passando o id que geramos e o resultado de builder.build() conforme mostrado na Figura 5.

```
if(Build.VERSION.SDK_INT >= Build.VERSION_CODES.O){
    notificationChannel = NotificationChannel(
        channelId,descricao,NotificationManager.IMPORTANCE_DEFAULT)

    notificationManager.createNotificationChannel(notificationChannel)

    builder = NotificationCompat.Builder( context: this, channelId)
        .setAutoCancel(true)
        .setSmallIcon(R.drawable.ic_launcher_foreground)
        .setContentTitle(titulo)
        .setContentText(corpo)
        .setPriority(NotificationCompat.PRIORITY_DEFAULT)
        .setContentIntent(pendingIntent)
}else{
    builder = NotificationCompat.Builder( context: this)
        .setAutoCancel(true)
        .setSmallIcon(R.drawable.ic_launcher_foreground)
        .setContentTitle(titulo)
        .setContentText(corpo)
        .setPriority(NotificationCompat.PRIORITY_DEFAULT)
        .setContentIntent(pendingIntent)
}

notificationManager.notify(id,builder.build())
```

Figura 5. Definir notificação

Desta forma temos o resultado mostrado na figura 6.

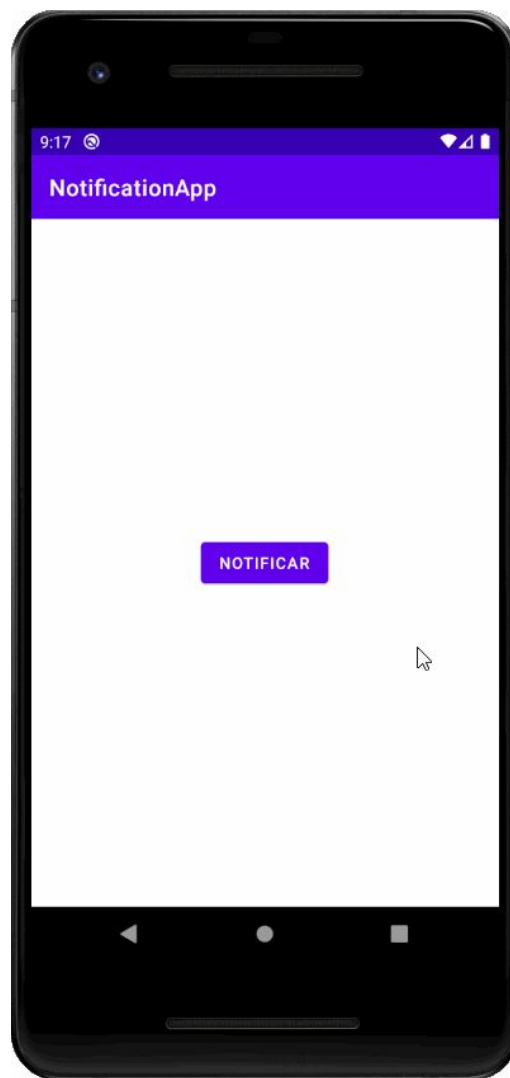


Figura 6. Funcionamento

Passo 4 = Parte 2

Neste passo vamos integrar nosso app com o serviço do firebase para que possamos receber notificações de forma remota utilizando o FCM.

Primeiramente autorize o uso de internet no arquivo de manifesto do seu aplicativo, para isso insira antes da abertura da tag application a seguinte linha de código:

```
<uses-permission android:name="android.permission.INTERNET"/>
```

Agora acesse e faça login no console do [firebase](#) no seu navegador padrão, e crie um projeto. Com ele criado podemos adicionar um app a ele que pode ser iOS, Android, Web ou Unity. No nosso caso a opção Android, logo podemos inserir os dados do nosso app como nome do pacote, e um apelido e a partir disso fazer o download do arquivo de configurações, entretanto existe uma maneira mais simples que é utilizando o Firebase Assistente, para isso basta acessar o menu ferramentas em seguida selecionar Firebase, Com ele aberto diversas opções serão mostradas, selecionaremos Cloud Messaging.

Neste momento temos o passo a passo automatizado para configurar nosso aplicativo, basta clicar em conectar ao firebase, uma nova janela será aberta em seu navegador, nela selecione o projeto recém criado, e automaticamente a conexão entre o app e o firebase será estabelecida.


De volta ao Android Studio no Assistente do firebase vamos clicar em adicionar fcm ao seu app para que todas as dependências sejam instaladas, aceite as mudanças que ele mostrará e aguarde o processo de sincronização finalizar. presentemente podemos enviar uma notificação remota, na página do projeto navegue até a opção Cloud Messaging e selecione enviar sua primeira mensagem, deste modo temos cinco passos sendo 2 desses opcionais. Em notificação preenchemos as informações da mensagem que será enviada (Figura 7). No menu Segmentação selecione seu aplicativo, o menu programação refere se a quando a notificação será enviada. Após a publicação a notificação deverá ser exibida instantaneamente. Lembrando que o aplicativo deve estar em background ou fechado para que a notificação seja entregue uma vez que por padrão a notificação só é exibida nestas condições.

1 Notificação

Título da notificação ?

Texto da notificação

Imagem de notificação (opcional) ?



Nome da notificação (opcional) ?

Figura 7. Menu Notificação

2 Segmentação

Segmento do usuário

Tópico

Segmentar usuário se...


 com.angeloplacebo.notificationapp

Figura 8. Menu Segmentação

3 Programação

Enviar para usuários qualificados

Agora

Próxima

Figura 9. Menu Programação

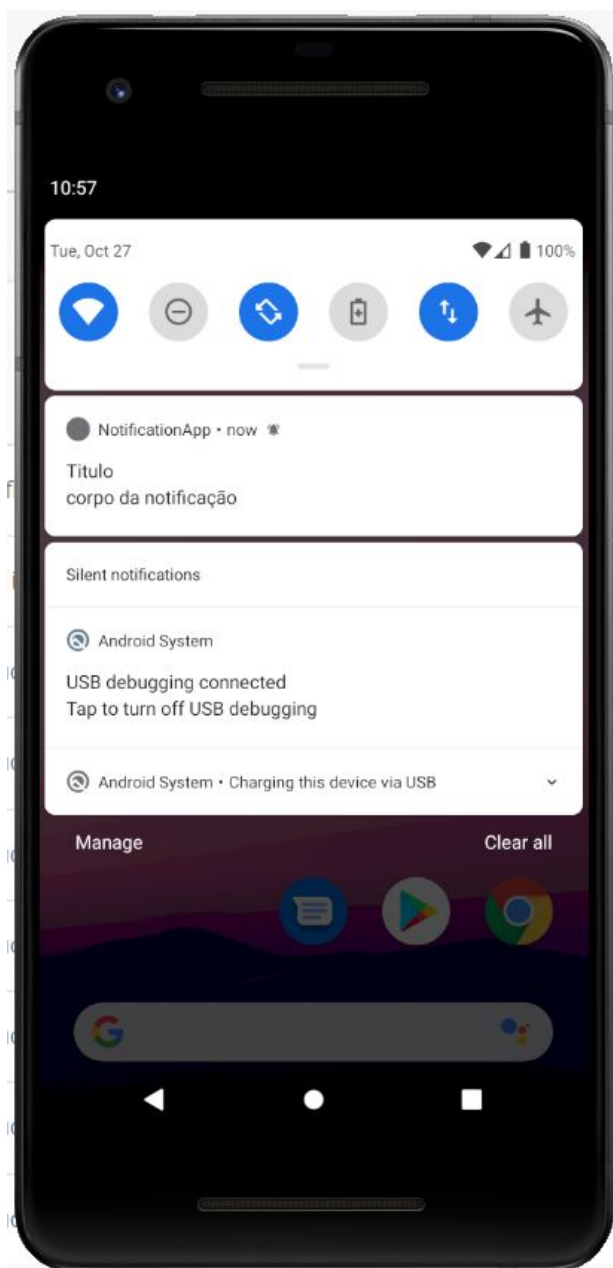


Figura 10. Notificação entregue

Com todos os passos anteriores concluídos podemos começar a configurar o serviço e a classe que chamaremos de **MyFirebaseMessagingService**. De volta ao arquivo de manifesto adicionaremos conforme o assistente nos indica no passo 4 o serviço.(Figura 11). Após isso criamos a classe Kotlin com o mesmo nome e indicamos que ela é do tipo `FirebaseMessagingService`, logo declaramos a sobrescrita da função **onMessageReceive** e **onNewToken** e nela utilizando o log mostraremos quando a mensagem for recebida.(Figura 12.).

Declare the following in your application's manifest:

```
<service
    android:name=".MyFirebaseMessagingService"
    <intent-filter>
        <action android:name="com.google.firebase.MESSAGING_EVENT"/>
    </intent-filter>
</service>
```

Figura 11. My Firebase Messaging Service

```
class MyFirebaseMessagingService : FirebaseMessagingService() {

    override fun onMessageReceived(remoteMessage: RemoteMessage) {
        super.onMessageReceived(remoteMessage)
        Log.e( tag: "Notification_APP", msg: "Mensagem Recebida")
    }

    override fun onNewToken(remoteMessage: String) {
        super.onNewToken(remoteMessage)
        Log.e( tag: "Notification_APP", msg: "Novo Token Recebido")
    }
}
```

Figura 12. Classe MyFirebaseMessagingService

Agora que estamos recebendo a notificação com o app em primeiro plano, vamos adaptar nossa função feita na parte um, desta vez iremos pegar os dados vindos da mensagem remota e iremos passar por parâmetro para nossa função (Figura 13.),dentro da função **onMessageReceived** para armazenar o título e o corpo teremos duas variáveis, chamaremos nossa nova função de **handleShowNotification** que receberá o contexto da aplicação e duas variáveis do tipo `String` que poderão ser nulas.(Figuras 14 e 15.).


```

override fun onMessageReceived(remoteMessage: RemoteMessage) {
    super.onMessageReceived(remoteMessage)
    val title :String? = remoteMessage.notification?.title
    val body :String? = remoteMessage.notification?.body
    handleShowNotification(applicationContext,title,body)
}

```

Figura 13. Função OnMessageReceived

```

fun handleShowNotification(context: Context,title: String?, message: String?){

    val titulo :Spanned = HtmlCompat.fromHtml( source: "<strong>${title}</strong>",
        HtmlCompat.FROM_HTML_MODE_LEGACY)

    val notificationChannel : NotificationChannel
    val builder : NotificationCompat.Builder
    val channelId = "com.angeloplacebo.notificationapp"

    val notificationManager : NotificationManager = applicationContext.getSystemService(
        Context.NOTIFICATION_SERVICE) as NotificationManager

    val id :Int = Random.nextInt( from: 1, until: 100)

    val intent = Intent( packageContext: this, MainActivity::class.java)
    val pendingIntent :PendingIntent! = PendingIntent.getActivity(
        context: this, requestCode: 0,intent, flags: 0)
}

```

Figura 14. Função handleShowNotification

```

if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
    notificationChannel = NotificationChannel(channelId, message,
        NotificationManager.IMPORTANCE_DEFAULT)
    notificationManager.createNotificationChannel(notificationChannel)

    builder = NotificationCompat.Builder(context: this, channelId)
        .setAutoCancel(true)
        .setSmallIcon(R.drawable.ic_launcher_foreground)
        .setContentTitle(titulo)
        .setContentText(message)
        .setPriority(NotificationCompat.PRIORITY_DEFAULT)
        .setContentIntent(pendingIntent)
} else {
    builder = NotificationCompat.Builder(context: this)
        .setAutoCancel(true)
        .setSmallIcon(R.drawable.ic_launcher_foreground)
        .setContentTitle(titulo)
        .setContentText(message)
        .setPriority(NotificationCompat.PRIORITY_DEFAULT)
        .setContentIntent(pendingIntent)
}

notificationManager.notify(id, builder.build())

```

Figura 15. Função handleShowNotification

Assim concluímos a parte 2, exceto que desta forma quando a notificação tiver um corpo muito longo não teremos como exibir completamente, pois por padrão o corpo é truncado para que seja exibido em apenas uma linha, para resolver isso iremos inserir no nosso builder os métodos **.setCategory** definindo como mensagem **.setStyle** definido como **bigTextStyle** e passaremos o corpo da mensagem para nossa notificação, definindo também um novo título uma vez que ao expandir ela pode ter um novo título, e um sumário conforme o código na figura 16. e a exibição final na figura 17.

```

builder = NotificationCompat.Builder( context: this, channelId)
    .setCategory(NotificationCompat.CATEGORY_MESSAGE)
    .setAutoCancel(true)
    .setSmallIcon(R.drawable.ic_launcher_foreground)
    .setContentTitle(titulo)
    .setStyle(
        NotificationCompat.BigTextStyle()
            .bigText(message)
            .setSummaryText("Summary")
            .setBigContentTitle(
                HtmlCompat.fromHtml( source: "<strong>Big Text</strong>",
                    HtmlCompat.FROM_HTML_MODE_LEGACY))
    )
    .setPriority(NotificationCompat.PRIORITY_DEFAULT)
    .setContentIntent(pendingIntent)

```

Figura 15. Builder com BigTextStyle

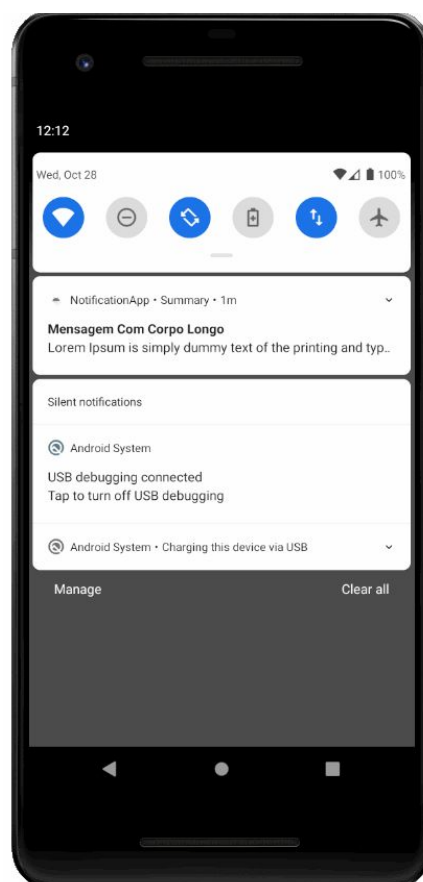


Figura 17. Notificação Expandida