

# Relatório - Programação Para Dispositivos Móveis

**Alunos:** Gustavo Chiossi, Gabriel Senkovski

O objetivo desse trabalho é implementar aplicativos Android usando 3 conjuntos de tecnologias diferentes: **Android Studio**, **Linha de Comando**, **Flutter e Dart**. O programa precisa ter entrada, processamento de dados e saída.

## Primeira etapa: Android Studio

Software necessário: [Android Studio](#).

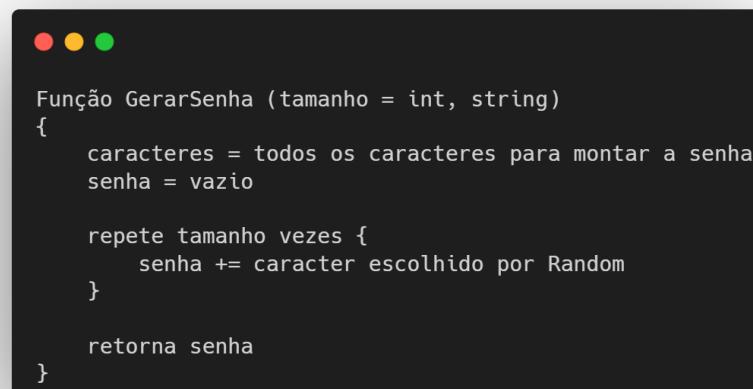
Para criar um projeto do zero, é necessário seguir as seguintes etapas no software do Android Studio: *New Project-> Empty Activity-> Nomear o aplicativo-> Finish*. O SDK usado foi a API 24 (“Nougat”, Android 7.0). A linguagem usada foi Kotlin. Após isso é necessário esperar o *Gradle project sync* finalizar, para configurar e baixar todas as dependências necessárias pro Android Studio compilar e rodar o aplicativo corretamente.

Como está sendo usado o Jetpack Compose, não é necessário mexer com arquivos .xml, deixando o processo de desenvolvimento mais simples. Foi escolhido desenvolver um aplicativo **gerador de senhas aleatórias**, onde a entrada será o tamanho de caracteres, processamento será a geração dos caracteres, e a saída será a senha final.

Principais *imports* feitos:

- *kotlin.random.Random*, para conseguir gerar os caracteres aleatórios.
- *androidx.compose.runtime.mutableStateOf*, para atualizar a UI.
- *androidx.compose.material3.OutlinedTextField*, para criar o campo que o usuário digita a entrada.
- entre outros *imports* feitos a fim de trabalhar com as dimensões e alinhamentos da UI.

O código pode ser dividido em 2 funções principais: a lógica (Figura 1) e a interface (Figura 2).

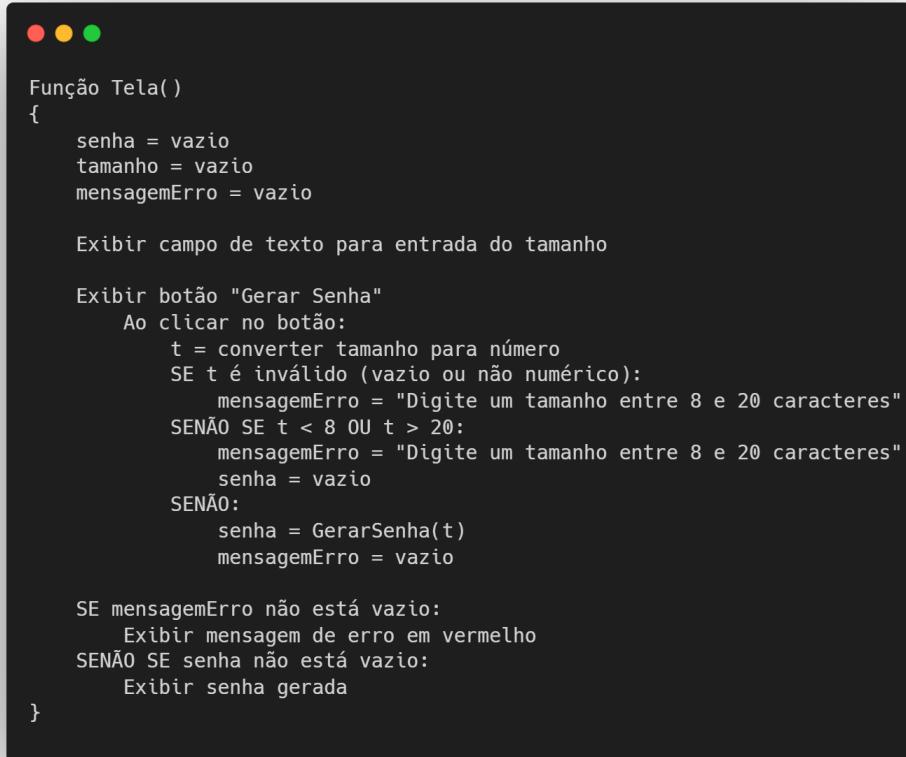


```
Função GerarSenha (tamanho = int, string)
{
    caracteres = todos os caracteres para montar a senha
    senha = vazio

    repete tamanho vezes {
        senha += caractere escolhido por Random
    }

    retorna senha
}
```

Figura 1: pseudocódigo da função que gera a senha aleatória no aplicativo.



```
Função Tela()
{
    senha = vazio
    tamanho = vazio
    mensagemErro = vazio

    Exibir campo de texto para entrada do tamanho

    Exibir botão "Gerar Senha"
        Ao clicar no botão:
            t = converter tamanho para número
            SE t é inválido (vazio ou não numérico):
                mensagemErro = "Digite um tamanho entre 8 e 20 caracteres"
            SENÃO SE t < 8 OU t > 20:
                mensagemErro = "Digite um tamanho entre 8 e 20 caracteres"
                senha = vazio
            SENÃO:
                senha = GerarSenha(t)
                mensagemErro = vazio

            SE mensagemErro não está vazio:
                Exibir mensagem de erro em vermelho
            SENÃO SE senha não está vazia:
                Exibir senha gerada
}
```

Figura 2: pseudocódigo da função que gerencia a UI do aplicativo.

A função de **gerar senha**, representada pelo pseudocódigo na Figura 1, tem 2 parâmetros: a variável *tamanho* (int), e retorna uma string como saída (a senha completa). A cada iteração no laço de repetição, é usado *kotlin.random.Random* para concatenar 1 caractere presente na variável *caracteres*, até a condição do laço ser rompida. Após isso, retorna a senha completa.

Já a função de **interface** recebe a entrada do usuário através de um campo de texto, armazenando o tamanho desejado da senha. Quando o botão “Gerar Senha” é pressionado, a função verifica se o valor digitado é numérico e se está dentro do intervalo permitido (8 a 20 caracteres). Caso a entrada seja inválida, uma mensagem de erro é exibida. Caso contrário, a função chama a outra função (*GerarSenha*) para gerar a senha aleatória e exibir na tela.

Para testar o programa, foi optado pelo emulador Android Virtual Device (AVD), disponível no próprio software do Android Studio. Foi necessário mudar a virtualização para *Software*, já que deixando no padrão (automático) estava causando erro. A API utilizada foi a 36.0. O resto da configuração foi deixada nos valores padrão. O resultado é mostrado na Figura 3:



Figura 3: nessa emulação de teste, o tamanho da senha é 8 caracteres.

## Segunda etapa: Linha de Comando

Foi desenvolvido no [WSL 2](#). É necessário instalar as seguintes ferramentas: [Java Development Kit \(JDK\)](#) e [Android Command line Tools](#). O passo a passo da instalação é visto nos tópicos seguintes.

1. No terminal do WSL 2, para instalação do JDK, use:

```
$ curl -LO https://download.oracle.com/java/25/latest/jdk-25_linux-x64_bin.tar.gz  
$ tar zxf jdk-25_linux-x64_bin.tar.gz  
$ export JAVA_HOME="$PWD/jdk-25.0.1"  
$ export PATH="$JAVA_HOME/bin:$PATH"  
$ java --version
```

Use `java --version` para confirmar a instalação. Se foi bem sucedido, deve retornar algo parecido com:

```
java 25.0.1 2025-10-21 LTS  
Java(TM) SE Runtime Environment (build 25.0.1+8-LTS-27)  
Java HotSpot(TM) 64-Bit Server VM (build 25.0.1+8-LTS-27, mixed mode, sharing)
```

2. No terminal, para instalação do Android CLI Tools, use:

```
$ mkdir android_sdk  
$ SDK="$PWD/android_sdk"
```

Os comandos acima são usados para definir o diretório do Android CLI Tools. Abaixo segue a continuação da instalação, com *sdkmanager*:

```
$ curl -LO  
https://dl.google.com/android/repository/commandlinetools-linux-13114758\_latest.zip  
$ unzip -q commandlinetools-linux-13114758_latest.zip -d "$SDK"  
$ "$SDK/cmdline-tools/bin/sdkmanager" --sdk_root="$SDK" --install \  
"platforms;android-36" "build-tools;35.0.0" "platform-tools" "ndk;27.3.13750724"
```

Agora é necessário criar e configurar os 3 arquivos principais do aplicativo. Primeiramente é necessário criar a pasta do projeto (nesse caso, *App/*). O arquivo *AndroidManifest.xml* fica na raiz dessa pasta. Depois criamos o arquivo que vai definir a UI (*App/res/layout/activity\_main.xml*), e o arquivo (*App/java/net/gustavo/app/MainActivity.java*) que controla a lógica do aplicativo. Não é possível reaproveitar o código criado na primeira etapa, pois nessa etapa, não usamos Gradle, ou seja, não é possível usar Jetpack Compose. Tendo isso em vista, o aplicativo desenvolvido nessa parte é bem simples: tem como entrada o nome do usuário, e a saída é a frase “Hello, <nome do usuário>”. O programa foi feito em java.

Após o desenvolvimento do aplicativo, é preciso fazer a build do mesmo. Em *App/*, foi usado o comando ***mkdir -p build/gen build/obj build/apk***, para criar a pasta de build. Após isso, para gerar *R.java*, foi usado os comandos:

```
$ "$SDK/cmdline-tools/bin/sdkmanager" --sdk_root="$SDK" --install  
"platforms;android-34"  
$ PLATFORM="$SDK/platforms/android-34"  
$ BUILD_TOOLS="$SDK/build-tools/34.0.0"  
$ "$BUILD_TOOLS/aapt" package -f -m -J build/gen/ -S res \  
-M AndroidManifest.xml -I "$PLATFORM/android.jar"
```

Compilar (javac):

```
$ javac --release 11 -classpath "$PLATFORM/android.jar" -d build/obj \  
build/gen/net/gustavo/app/R.java java/net/gustavo/app/MainActivity.java
```

Traduzir pra Dalvik (d8):

```
$ "$BUILD_TOOLS/d8" --release --lib "$PLATFORM/android.jar" \  
--output build/apk/build/obj/net/gustavo/app/*.class
```

Juntar em um APK (aapt):

```
$ "$BUILD_TOOLS/aapt" package -f -M AndroidManifest.xml -S res \  
-I "$PLATFORM/android.jar" -F build/apk/app.apk
```

```
-I "$PLATFORM/android.jar" \
-F build/App.unsigned.apk build/apk/
```

Aplicativo precisa ser *signed*:

Parte 1- zipalign:

```
$ "$BUILD_TOOLS/zipalign" -f -p 4 \
build/App.unsigned.apk build/App.aligned.apk

$ keytool -genkeypair -keystore keystore.jks -alias androidkey \
-validity 10000 -keyalg RSA -keysize 2048 \
-storepass android -keypass android
```

As perguntas foram deixadas em branco (Unknown), e no final é necessário aceitar digitando “yes”.

Parte 2- apksigner:

```
$ "$BUILD_TOOLS/apksigner" sign --ks keystore.jks \
--ks-key-alias androidkey \
--ks-pass pass:android \
--key-pass pass:android \
--out build/App.apk \
build/App.aligned.apk
```

A execução do programa foi feita no Android Studio. Os arquivos do aplicativo foram copiados para o Windows e executados com o mesmo AVD usado na etapa anterior. Os comandos utilizados no Powershell do Windows foram:

```
$ PS C:\Users\Chios\Documents\ESTUDOS\#1 FACUL\2
ANO\ANUAIS\TDS\TRAB3\CL\App\build> &
"C:\Users\Chios\AppData\Local\Android\Sdk\platform-tools\adb.exe" install -r
.\App.apk

$ PS C:\Users\Chios\Documents\ESTUDOS\#1 FACUL\2
ANO\ANUAIS\TDS\TRAB3\CL\App\build> &
"C:\Users\Chios\AppData\Local\Android\Sdk\platform-tools\adb.exe" shell am start -n
net.gustavo.app/.MainActivity
```

O resultado é mostrado na Figura 4:

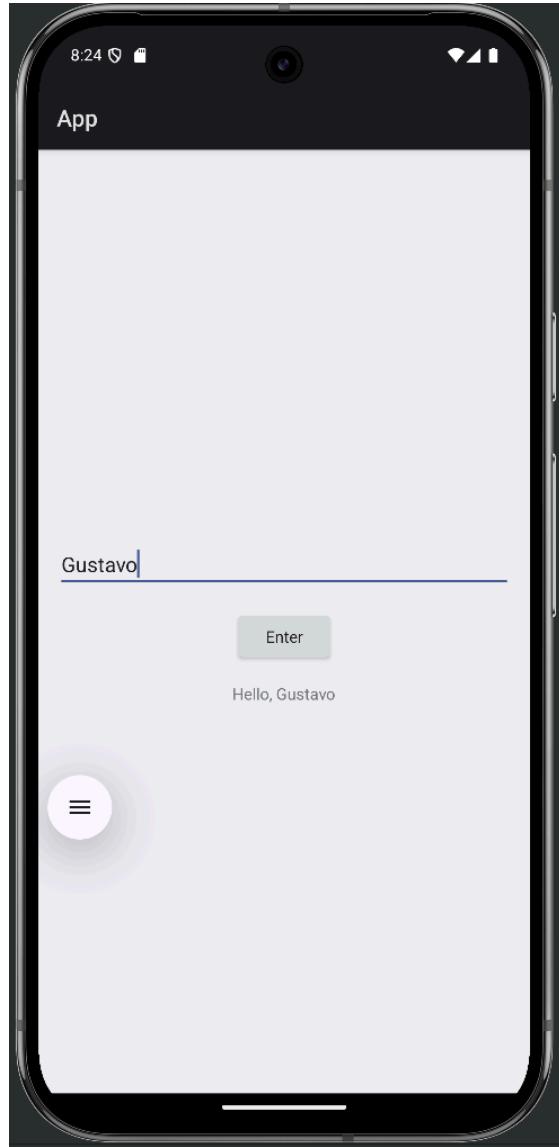


Figura 4: Teste do aplicativo com o usuário “Gustavo”.

## Terceira etapa: Flutter e Dart

### Instalação do SDK do Flutter

O primeiro passo foi baixar o SDK (Software Development Kit) do Flutter a partir do [site oficial](#). Após o download, o SDK foi descompactado em um local permanente no computador (ex: C:\flutter) e seu diretório bin foi adicionado às variáveis de ambiente (PATH) do sistema operacional.

### Instalação dos Plugins no Android Studio

Com o Android Studio já instalado (usado na primeira etapa do trabalho), foi necessário adicionar suporte ao Flutter:

1. Abrimos o Android Studio e navegamos até File > Settings > Plugins.

2. Procuramos e instalamos o plugin **Flutter**. Este plugin também instala automaticamente o plugin **Dart** como dependência.
3. Após a instalação, o Android Studio foi reiniciado.

## Verificação do Ambiente

Para garantir que toda a configuração estava correta, executamos o comando *flutter doctor* em um terminal, como mostra a Figura 5:

```
C:\Users\glsen>flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[✓] Flutter (Channel stable, 3.35.7, on Microsoft Windows [versão 10.0.19045.6456], locale pt-BR)
[✓] Windows Version (10 Home 64 bits, 22H2, 2009)
[!] Android toolchain - develop for Android devices (Android SDK version 36.1.0)
    ✘ cmdline-tools component is missing.
      Try installing or updating Android Studio.
      Alternatively, download the tools from https://developer.android.com/studio#command-line-tools-only and make sure
      to set the ANDROID_HOME environment variable.
      See https://developer.android.com/studio/command-line for more details.
    ✘ Android license status unknown.
      Run `flutter doctor --android-licenses` to accept the SDK licenses.
      See https://flutter.dev/to/windows-android-setup for more details.
[✓] Chrome - develop for the web
[✗] Visual Studio - develop Windows apps
    ✘ Visual Studio not installed; this is necessary to develop Windows apps.
      Download at https://visualstudio.microsoft.com/downloads/
      Please install the "Desktop development with C++" workload, including all of its default components
[✓] Android Studio (version 2025.1.4)
[✓] VS Code (version 1.91.1)
[✓] Connected device (3 available)
[✓] Network resources

! Doctor found issues in 2 categories.
```

Figura 5: verificando a integridade da instalação das ferramentas para uso do flutter.

## Criação do Novo Projeto Flutter

Com os plugins instalados, uma nova opção de projeto apareceu no Android Studio:

1. Fomos em File > New > New Flutter Project....
2. Selecioneamos o tipo "Flutter Application", definimos o nome do projeto (ex: tdsflutter) e o local para salvar, como mostra a Figura 6.
3. O Android Studio gerou uma estrutura de projeto completa, incluindo um aplicativo de exemplo (contador).

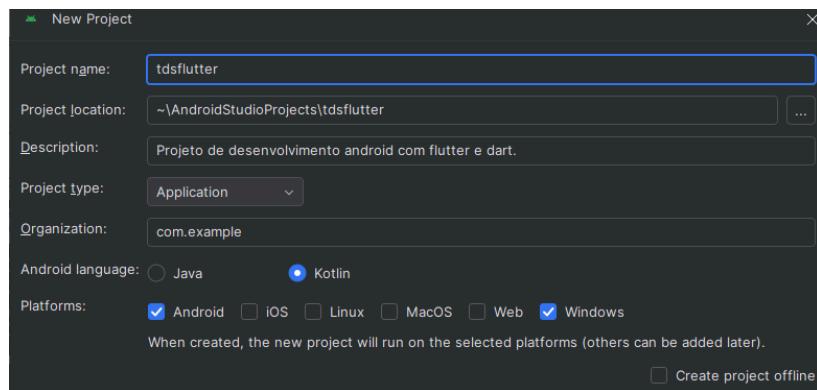


Figura 6: Demonstrando a criação do projeto usado no trabalho.

## Estrutura do Código

A ideia de aplicativo nesse caso foi a mesma ideia desenvolvida na primeira etapa (gerador de senha aleatória). A lógica principal de um aplicativo Flutter não reside nos arquivos *.kt* ou *.java* dentro da pasta *android/*, que servem apenas como "hospedeiros" nativos. Todo o

código da interface e da lógica do aplicativo foi implementado na linguagem Dart, dentro do arquivo **lib/main.dart**.

### Configuração do Emulador Android

Para testar o aplicativo, um dispositivo virtual Android (AVD) foi configurado:

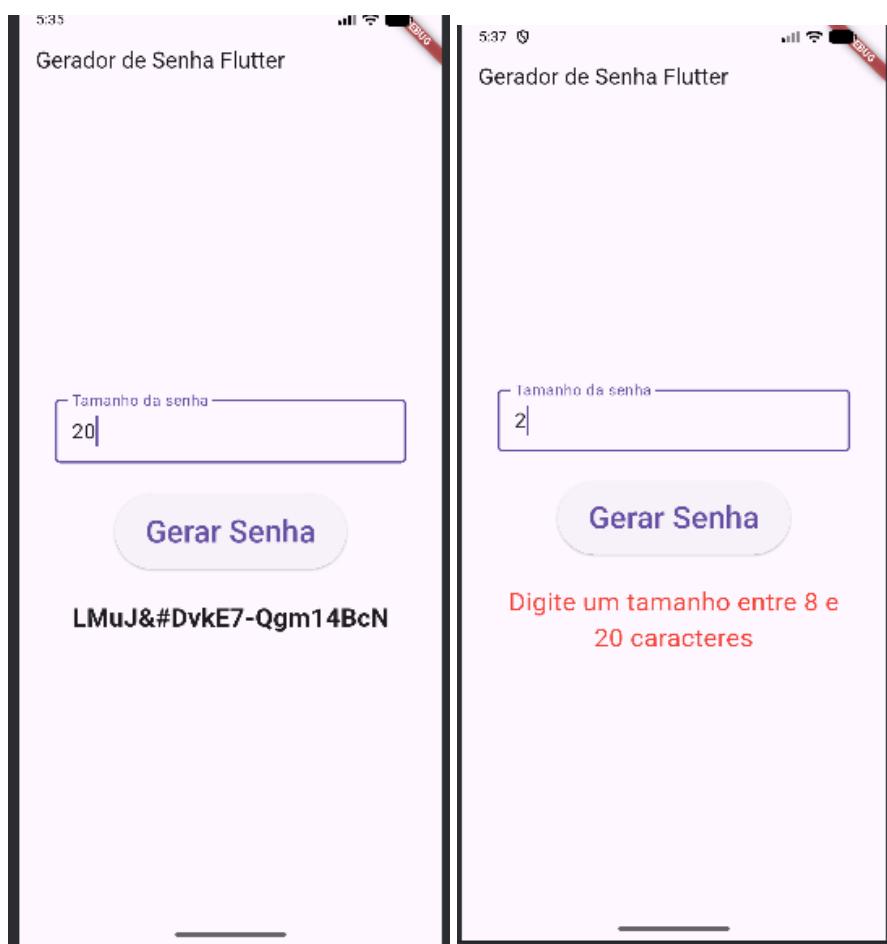
1. No Android Studio, fomos em Tools > Device Manager.
2. Um novo emulador foi criado (ex: "Medium Phone API 36.1").
3. O emulador foi iniciado e permaneceu rodando.

### Seleção do Dispositivo de Destino

Um ponto crucial foi selecionar o dispositivo correto para a execução. Na barra de ferramentas principal, o alvo de execução (que por padrão poderia estar como "Windows (desktop)" ou "Chrome") foi alterado para o emulador Android em execução (ex: Medium Phone API 36.1).

### Execução do Aplicativo

Com o dispositivo correto selecionado e o código em *lib/main.dart* pronto, o aplicativo foi executado, com os resultados exibidos nas Figuras 7 e 8:



Figuras 7 e 8: Teste da lógica de processamento do app, exibindo um caso de sucesso (esquerda) e um de validação de erro (direita).

