

PdAM-I

Programação de Aplicativos Mobile - I

Passo a Passo

Criar um novo projeto

Antes de criar um novo projeto, verifique se você está usando o Android Studio 4.1 ou mais recente.

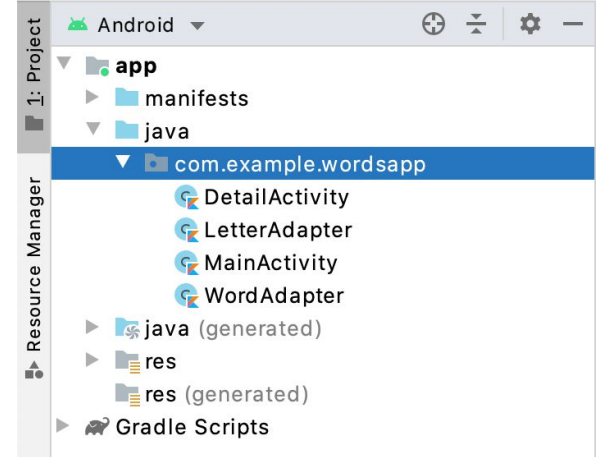
1. Inicie um novo projeto Kotlin no Android Studio usando o modelo **Empty Activity**.
2. Insira **Words** no campo **Name** do app, **com.example.words** no campo **Package name** e escolha **API Level 19** no campo **Minimum SDK**.
3. Clique em **Finish** para criar o projeto.

Para esse projeto trabalharemos com 4 arquivos básicos para produzir o funcionamento do app.

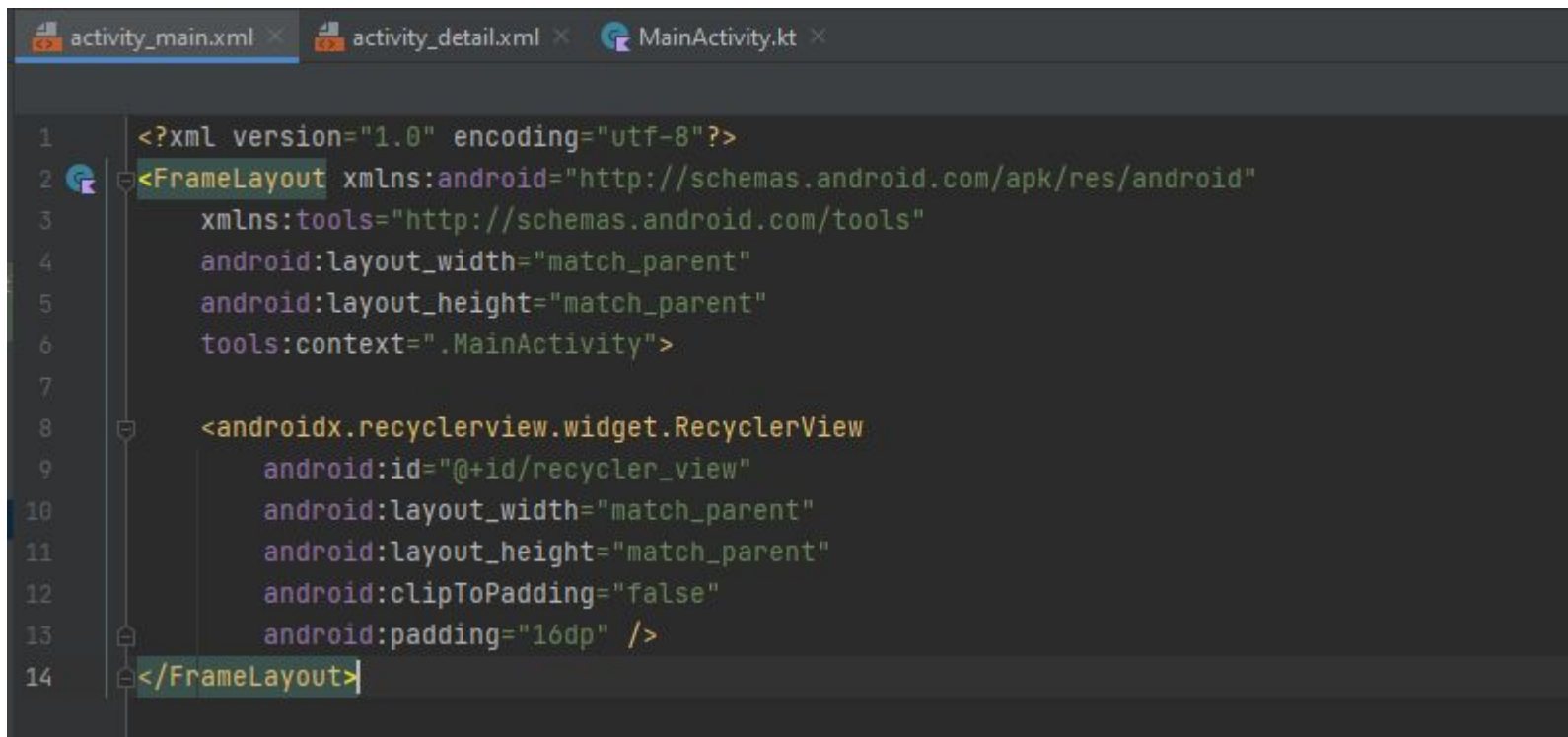
Detail Activity: que mostra os detalhes da palavra selecionada

Letter Adapter: que é o adaptador para essa activity

Main activity que mostra a lista de letras.



Criação dos arquivos de Layout



The screenshot shows an IDE with three tabs: activity_main.xml, activity_detail.xml, and MainActivity.kt. The activity_main.xml tab is active, displaying the following XML code:

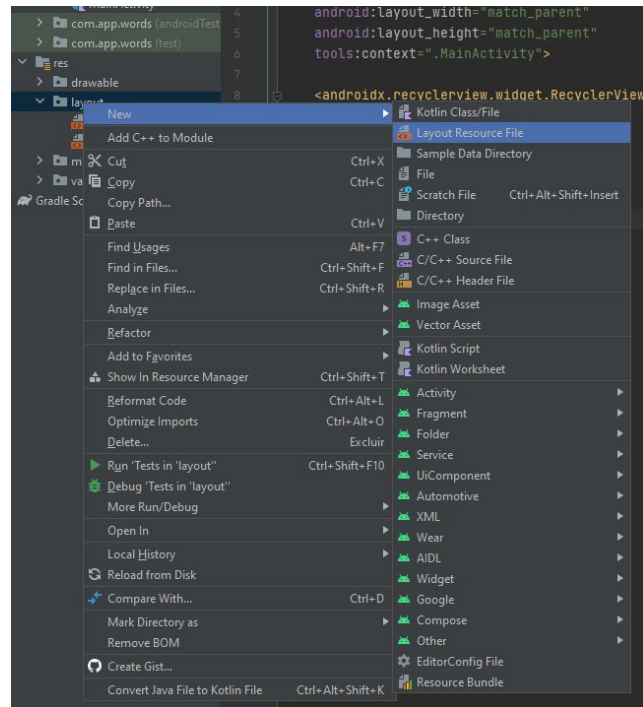
```
1  <?xml version="1.0" encoding="utf-8"?>
2  <FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:tools="http://schemas.android.com/tools"
4      android:layout_width="match_parent"
5      android:layout_height="match_parent"
6      tools:context=".MainActivity">
7
8      <androidx.recyclerview.widget.RecyclerView
9          android:id="@+id/recycler_view"
10         android:layout_width="match_parent"
11         android:layout_height="match_parent"
12         android:clipToPadding="false"
13         android:padding="16dp" />
14 </FrameLayout>
```

Criação dos arquivos de Layout

O Objetivo dessa Mainactivity é listar as letras para que possamos criar o dicionário.

Esse layout é um recycler_view e sendo assim é necessário criar outros arquivos de layout, um activity_detail e um item_view.

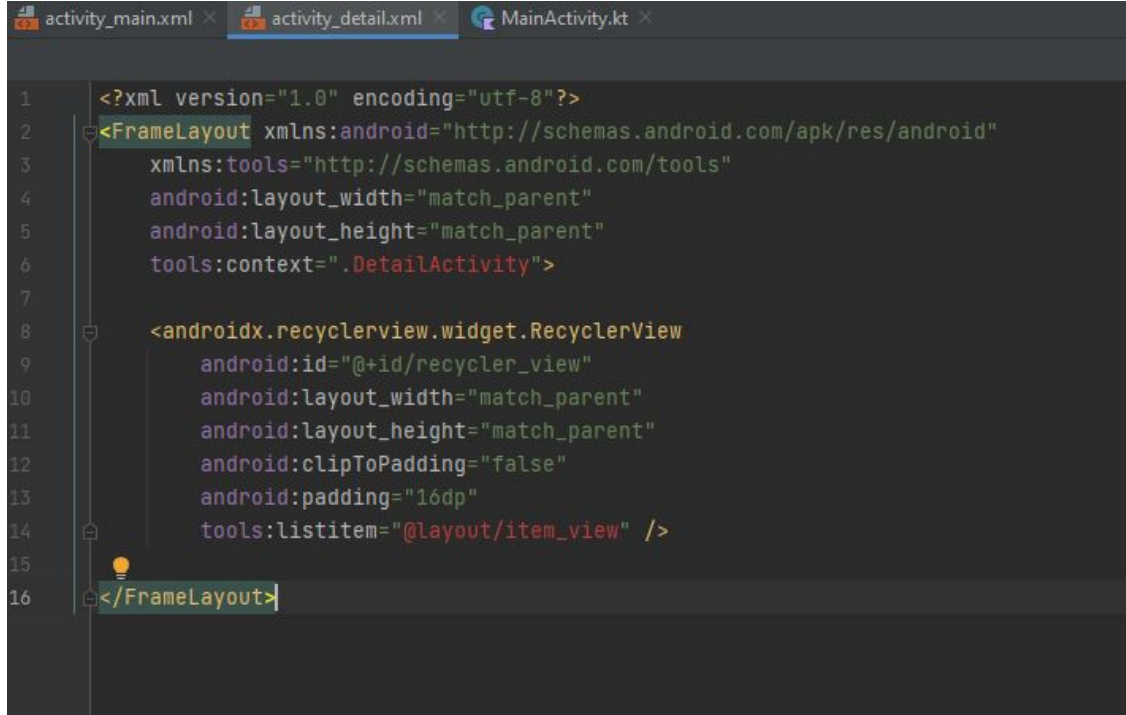
Para a criação de um novo arquivo de layout clique com o botão direito em cima de layout e em novo LayoutResourceFile.



Criação dos arquivos de Layout

A criação de `activity_detail` exige que esse outro `FrameLayout` tenha como contexto o `.DetailActivity` que ainda não existe.

Esse context se refere a activity que coordena essa View.

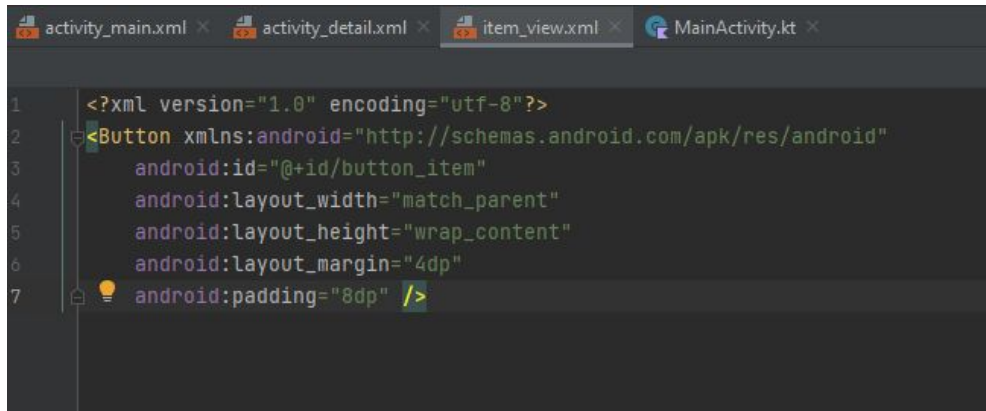


```
1  <?xml version="1.0" encoding="utf-8"?>
2  <FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
3              xmlns:tools="http://schemas.android.com/tools"
4              android:layout_width="match_parent"
5              android:layout_height="match_parent"
6              tools:context=".DetailActivity">
7
8      <androidx.recyclerview.widget.RecyclerView
9          android:id="@+id/recycler_view"
10         android:layout_width="match_parent"
11         android:layout_height="match_parent"
12         android:clipToPadding="false"
13         android:padding="16dp"
14         tools:listitem="@layout/item_view" />
15
16 </FrameLayout>
```

Criação dos arquivos de Layout

O próximo arquivo de layout é justamente o `item_view` que representa cada um dos itens da lista de palavras.

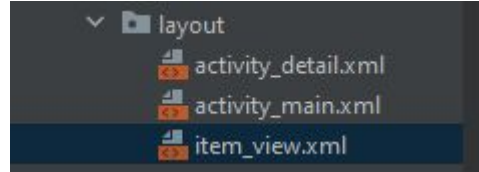
Ele é basicamente um **botão**.



```
1 <?xml version="1.0" encoding="utf-8"?>
2 <Button xmlns:android="http://schemas.android.com/apk/res/android"
3     android:id="@+id/button_item"
4     android:layout_width="match_parent"
5     android:layout_height="wrap_content"
6     android:layout_margin="4dp"
7     android:padding="8dp" />
```


Criação dos arquivos de Layout

Assim teremos 3 arquivos de layout:



Criação de um array para as palavras

Para a busca das palavras será necessário criar uma lista de palavras, um

Configurando uma lista de dados

Adicionar strings de afirmação

1. Criar um arquivo Arrays.xml **Project**, abra **app > res > values > arrays.xml**.

O objetivo é popular esse arquivo, como exemplo pegamos linguagens e termos de programação e TI.

O formato do arquivo deve ser algo do tipo:

```
?xml version="1.0" encoding="utf-8"?>
```

```
<resources>
```

```
  <string-array name="words">
```

```
    <item>Assembly</item>
```

```
    <item>ASP.net</item>
```

```
    <item>Basic</item>
```

```
    <item>C</item>
```

```
    <item>C++</item>
```

```
    <item>C#</item>
```

```
    <item>COBOL</item>
```

```
    <item>Clojure</item>
```

```
  .
```

```
  .
```

```
  .
```

Criação das Activities e dos Adapters

A Main activity ficará algo como o próximo Slide.

O Objetivo é que ela seja capaz de gerenciar o RecyclerView.

Ela ainda não faz muita coisa, mas trabalharemos isso mais a frente.

```
package com.example.wordsapp

import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity
import androidx.recyclerview.widget.LinearLayoutManager
import androidx.recyclerview.widget.RecyclerView
import com.example.wordsapp.databinding.ActivityMainBinding

class MainActivity : AppCompatActivity() {
    private lateinit var recyclerView: RecyclerView

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        val binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        recyclerView = binding.recyclerView
        // Sets the LinearLayoutManager of the recyclerView
        recyclerView.layoutManager = LinearLayoutManager(this)
        recyclerView.adapter = LetterAdapter()
    }
}
```

Criaremos também uma `DetailActivity` para representar o layout que construímos anteriormente.

O Objetivo desta `Activity` é mostrar a lista de palavras de cada letra.

```
package com.example.wordsapp

import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity
import androidx.recyclerview.widget.DividerItemDecoration
import androidx.recyclerview.widget.LinearLayoutManager
import com.example.wordsapp.databinding.ActivityDetailBinding

class DetailActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        val binding = ActivityDetailBinding.inflate(layoutInflater)
        setContentView(binding.root)

        val letterId = "A"

        val recyclerView = binding.recyclerView
        recyclerView.layoutManager = LinearLayoutManager(this)
        recyclerView.adapter = WordAdapter(letterId, this)

        recyclerView.addItemDecoration(
            DividerItemDecoration(this, DividerItemDecoration.VERTICAL)
        )

        title = getString(R.string.detail_prefix) + " " + letterId
    }
}
```


Para a construção dos adapters o processo é um pouco mais complicado.

O código disponibilizado está no github:

<https://github.com/google-developer-training/android-basics-kotlin-words-app/blob/starter/app/src/main/java/com/example/wordsapp/LetterAdapter.kt>

<https://github.com/google-developer-training/android-basics-kotlin-words-app/blob/starter/app/src/main/java/com/example/wordsapp/WordAdapter.kt>

Adicionando uma intent Explícita

Na primeira tela, quando o usuário tocar em uma letra, ele será levado para uma segunda tela com uma lista de palavras. A `DetailActivity` já está implementada. Portanto, basta inicializá-la com uma intent. Como seu app sabe exatamente qual atividade precisa ser iniciada, você usará uma intent explícita.

A criação e o uso de intents podem ser feitos seguindo apenas algumas etapas.

1. Abra o arquivo `LetterAdapter.kt` e role para baixo até o `onBindViewHolder()`. Abaixo da linha que define o texto do botão, configure o `onClickListener` como `holder.button`.

```
holder.button.setOnClickListener {  
  
}
```

2. Em seguida, acesse uma referência ao `context`.

```
val context = holder.view.context
```

3. Crie uma `Intent`, transmitindo o contexto e o nome da classe da atividade de destino.

```
val intent = Intent(context, DetailActivity::class.java)
```

O nome da atividade que você quer mostrar é especificado por `DetailActivity::class.java`. Um objeto `DetailActivity` é criado internamente.

4. Chame o método `putExtra`, transmitindo "letter" como o primeiro argumento e o texto do botão como o segundo argumento.

```
intent.putExtra("letter", holder.button.text.toString())
```

Adicionando uma intent

O que é um extra? Lembre-se de que uma intent é simplesmente um conjunto de instruções. Ela ainda não tem uma instância da atividade de destino. Em vez disso, um extra é um dado, como um número ou uma string, que recebe um nome para ser recuperado depois. Isso é semelhante a transmitir um argumento quando você chama uma função. Como uma `DetailActivity` pode ser exibida para qualquer letra, você precisa informar qual letra será exibida.

Além disso, por que você acha que é necessário chamar `toString()`? O texto do botão já é uma string, certo?

Mais ou menos. Na verdade, o texto é do tipo `CharSequence`, que é chamado de *interface*. No momento, você não precisa saber nada sobre as interfaces do Kotlin, além de que elas são uma forma de garantir que um tipo, como uma `String`, implemente funções e propriedades específicas. Pense em uma `CharSequence` como uma representação mais genérica de uma classe semelhante a uma string. A propriedade `text` de um botão pode ser uma string ou um objeto qualquer que também seja uma `CharSequence`. No entanto, o método `putExtra()` aceita uma `String`, e não qualquer `CharSequence`. Portanto, você precisa chamar `toString()`.

5. Chame o método `startActivity()` no objeto de contexto, transmitindo a `intent`.

```
context.startActivity(intent)
```

Agora, execute o app e tente tocar em uma letra. A tela de detalhes será exibida. Mas, independentemente da letra tocada pelo usuário, a tela de detalhes sempre mostrará palavras para a letra "A". Ainda há algumas tarefas a serem realizadas na atividade detalhada para que ela mostre palavras para qualquer letra transmitida como o extra da `intent`.

Configurando a DetailActivity

Você acabou de criar sua primeira intent explícita. Agora, falaremos sobre a tela de detalhes.

No método `onCreate` da `DetailActivity`, após a chamada para `setContentView`, substitua a letra codificada pelo código para receber o `letterId` transmitido pela `intent`.

```
val letterId = intent?.extras?.getString("letter").toString()
```

Como tem muita coisa acontecendo ao mesmo tempo, vamos dar uma olhada em cada parte:

Primeiro, de onde vem a propriedade `intent`? Ela não é uma propriedade da `DetailActivity`, mas uma propriedade de qualquer atividade. Ela mantém uma referência à intent usada para iniciar a atividade.

A propriedade "extras" é do tipo `Bundle` e, como você pode imaginar, fornece uma maneira de acessar todos os extras transmitidos à intent.

Essas duas propriedades estão marcadas com um ponto de interrogação. Por quê? A razão é que as propriedades `intent` e `extras` são anuláveis, o que significa que elas podem ter um valor ou não. Às vezes, você quer que uma variável seja `null`. A propriedade `intent` pode não ser uma `Intent` de fato (se a atividade não tiver sido iniciada por uma intent) e a propriedade "extras" pode não ser um `Bundle`, mas um valor chamado `null`.

Em Kotlin, `null` significa a ausência de um valor. O objeto pode existir ou pode ser `null`. Se o app tentar acessar uma propriedade ou chamar uma função em um objeto `null`, ele falhará. Para acessar esse valor com segurança, coloque um "?" após o nome dele. Se a `intent` for `null`, o app não tentará acessar a propriedade "extras". Se a `extras` for nula, seu código não tentará chamar `getString()`.

Configurando a intent Implícita

Na maioria dos casos, você exibirá atividades específicas no seu app. No entanto, em algumas situações, você pode não saber qual atividade ou app quer iniciar. Na nossa tela de detalhes, cada palavra é um botão que mostra a definição da palavra ao usuário.

Em nosso exemplo, você usará a funcionalidade do dicionário fornecida pela Pesquisa Google. No entanto, em vez de adicionar uma nova atividade ao seu app, você iniciará o navegador do dispositivo para exibir a página de pesquisa.

É possível que alguns usuários prefiram usar um navegador de terceiros. O smartphone deles pode vir com um navegador pré-instalado pelo fabricante. Não é possível saber com certeza quais os apps o usuário instalou. E você não pode presumir a forma como eles querem procurar uma palavra. Esse é um exemplo perfeito de quando usar uma intent implícita. O app fornece informações ao sistema sobre qual ação deve ser tomada, e o sistema descobre o que fazer com essa ação, **solicitando ao usuário mais informações conforme necessário.**

Configurando a intent Implícita

1. Para este app, você fará uma Pesquisa Google pelo assunto selecionado. Como o mesmo URL base é usado para todas as pesquisas, é uma boa ideia defini-lo como uma constante. Na `DetailActivity`, modifique o objeto complementar para adicionar uma nova constante, `SEARCH_PREFIX`. Esse é o URL base da Pesquisa Google.

```
companion object {  
    const val LETTER = "letter"  
    const val SEARCH_PREFIX = "https://www.google.com/search?q="  
}
```

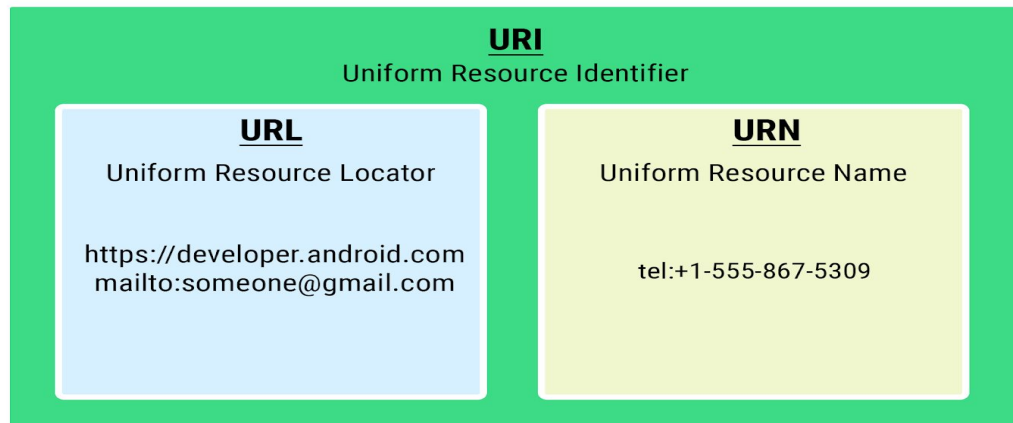

2. Em seguida, abra o `WordAdapter` e, no método `onBindViewHolder()`, chame `setOnClickListener()` no botão. Comece criando um `URI` para a consulta da pesquisa. Ao chamar o método `parse()` para criar um `URI` usando uma `String`, você precisa usar a formatação de string para que a palavra seja anexada ao `SEARCH_PREFIX`.

```
holder.button.setOnClickListener {  
    val queryUrl: Uri =  
Uri.parse("${DetailActivity.SEARCH_PREFIX}${item}")  
}
```

URI, é a sigla de *Uniform Resource Identifier* um URL ou *Uniform Resource Locator* é uma string que aponta para uma página da Web. Um URI é um termo mais geral para esse formato.

Todos os URLs são URIs, mas nem todos os URIs são URLs. Outros URIs, por exemplo, um endereço de um número de telefone, começam com **tel:**, mas são considerados um URN ou *Uniform Resource Name*, em vez de um URL.

O tipo de dados usado para representar ambos é chamado de **URI**.



2. Depois de definir o `queryUrl`, inicialize um novo objeto `intent`.

```
val intent = Intent(Intent.ACTION_VIEW, queryUrl)
```

Em vez de transmitir um contexto e uma atividade, transmita uma `Intent.ACTION_VIEW` junto ao `URI`.

A `ACTION_VIEW` é uma intent genérica que aceita um URI, no seu caso, um endereço da Web. O sistema, então, saberá que precisa processar essa intent abrindo o URI no navegador da Web do usuário. Alguns outros tipos de intent incluem:

- `CATEGORY_APP_MAPS`: para inicializar apps de mapas.
- `CATEGORY_APP_EMAIL`: para inicializar apps de e-mail.
- `CATEGORY_APP_GALLERY`: para inicializar apps de galeria (fotos).
- `ACTION_SET_ALARM`: para configurar um alarme em segundo plano.
- `ACTION_DIAL`: para inicializar uma chamada.

3. Por fim, mesmo que não esteja inicializando uma atividade específica no app, você está instruindo o sistema a iniciar outro app chamando `startActivity()` e transmitindo a `intent`.

```
context.startActivity(intent)
```

Agora, ao iniciar o app, navegue até a lista de palavras e toque em uma delas. O dispositivo acessará o URL ou apresentará uma lista de opções, dependendo dos apps instalados.

O comportamento exato varia para cada usuário, proporcionando uma experiência perfeita para todos, sem complicar o código.

Referência

<https://developer.android.com/codelabs/basic-android-kotlin-training-activities-intents?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-3-pathway-1%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-activities-intents#6>