

The background features a complex network of thin grey lines and dots, forming a web-like structure. Scattered throughout are various triangles of different sizes and orientations, some with solid black dots at their vertices. The overall aesthetic is modern and technical.

PROGRAMA DEV VENTURE

Desenvolvimento Android

REVISÃO
Onde estamos

01

ROOM
OMR

02

Aula 15 AGENDA



01

REVISÃO





REVISÃO

VIEWS

VIEW GROUP

LINEAR LAYOUT

CONSTRAINT LAYOUT

RESOURCES

VIEW BINDING





REVISÃO

ACTIVITIES

FRAGMENTS

CICLO DE VIDA

ANDROID JETPACK

NAVIGATION COMPONENT

KOTLIN & POO





REVISÃO

INTENTS

MANIFEST

COMPONENTS DE APPS

CLASSE R

GRADLE

RECYCLER VIEW





REVISÃO

SOLID

ARQUITETURA DE SOFTWARE

CLEAN ARCH

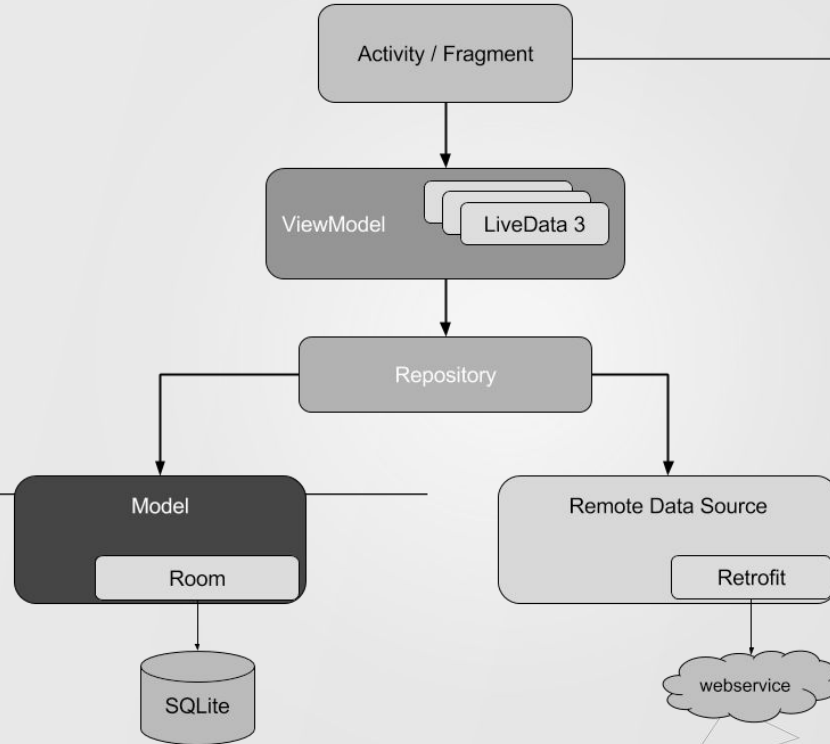
MVVM

ARCH COMPONENTS

REPOSITORY



GUIA ANDROID DE ARQUITETURA





GUIA ANDROID DE ARQUITETURA

Interface com usuário: as classes activities ou fragments ficam responsável por tratar a interação com usuário. Tarefas como exibir dados, capturar interações são de responsabilidade delas.



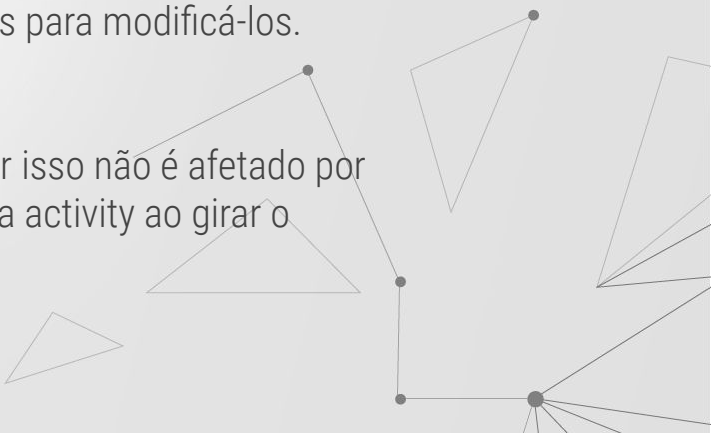


GUIA ANDROID DE ARQUITETURA

Um objeto ViewModel fornece os dados para um componente de UI específico, como um fragment ou activity, e contém lógica de manipulação de dados para se comunicar com o modelo.

Por exemplo, o ViewModel pode chamar outros componentes para carregar os dados e pode encaminhar solicitações de usuários para modificá-los.

O ViewModel não sabe sobre componentes de UI, por isso não é afetado por mudanças de configuração, como a recriação de uma activity ao girar o dispositivo.





GUIA ANDROID DE ARQUITETURA

Repositories manipulam operações de dados.

Eles disponibilizam uma interface limpa para que o restante do app possa recuperar dados com facilidade. Eles sabem onde coletar os dados e quais chamadas de API precisam ser feitas quando os dados são atualizados. Repositories são como mediadores entre fontes de dados diferentes, por exemplo, bancos de dados, serviços da Web e caches.



02

ROOM DATABASE





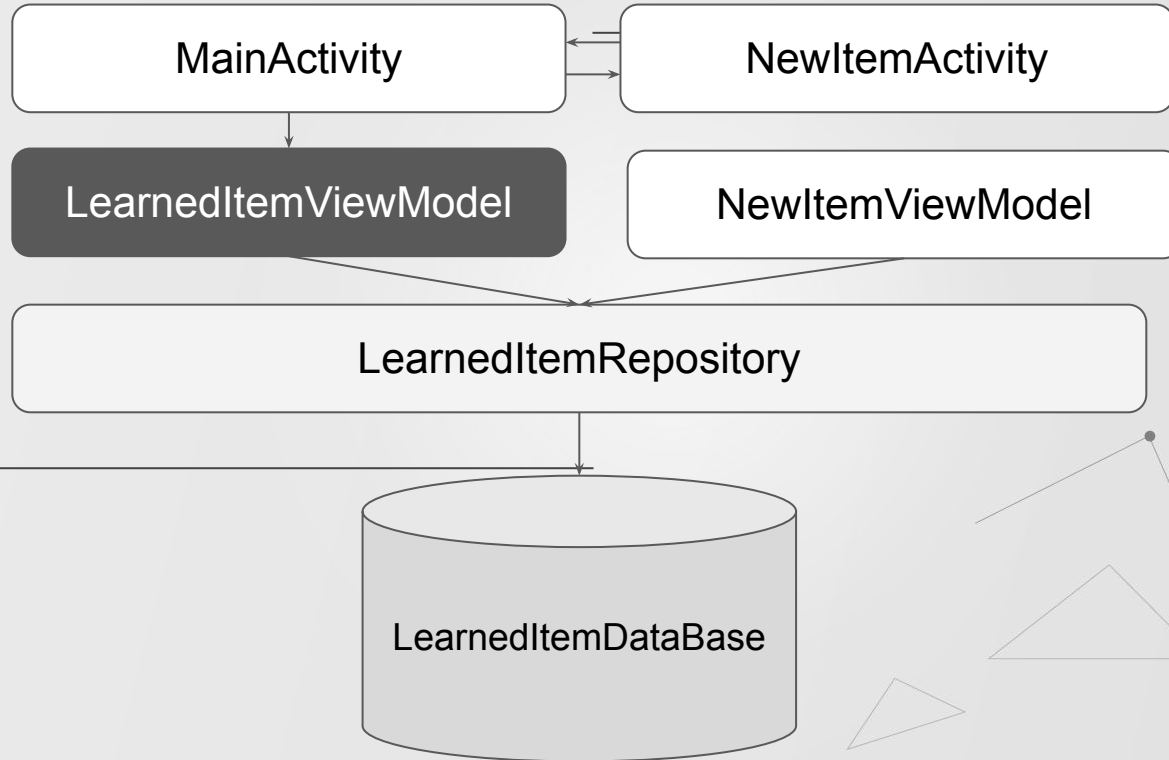
ROOM

A biblioteca de persistência Room oferece uma camada de abstração sobre o SQLite para permitir um acesso mais robusto ao banco de dados, aproveitando toda a capacidade do SQLite. (SQLite é um sistema de gestão de bancos de dados relational)

Vamos estruturar o App What Did I Learn para que as informações seja persistidas com o Room!



WHAT DID I LEARN - ARQUITETURA





dojo / DEPENDENCIAS

build.gradle (Module: app)

Aplique o plug-in Kotlin do processador de anotação kapt adicionando-o após os outros plug-ins definidos na parte superior do arquivo.

apply plugin: 'kotlin-kapt' ou
id 'kotlin-kapt'





dojo / DEPENDENCIAS

build.gradle (Module: app)

Aplique o plug-in Kotlin do processador de anotação kapt adicionando-o após os outros plug-ins definidos na parte superior do arquivo.

apply plugin: 'kotlin-kapt' ou
id 'kotlin-kapt'






dojo / DEPENDENCIAS

build.gradle (Module: whatdidilearn)

```
dependencies {  
  
    implementation "androidx.appcompat:appcompat:$rootProject.appCompatVersion"  
  
    // Dependencies for working with Architecture components  
    // You'll probably have to update the version numbers in build.gradle (Project)  
  
    // Room components  
    implementation "androidx.room:room-ktx:$rootProject.roomVersion"  
    kapt "androidx.room:room-compiler:$rootProject.roomVersion"  
    androidTestImplementation "androidx.room:room-testing:$rootProject.roomVersion"  
  
    // Lifecycle components  
    implementation "androidx.lifecycle:lifecycle-viewmodel-ktx:$rootProject.lifecycleVersion"  
    implementation "androidx.lifecycle:lifecycle-livedata-ktx:$rootProject.lifecycleVersion"  
    implementation "androidx.lifecycle:lifecycle-common-java8:$rootProject.lifecycleVersion"  
  
    // Kotlin components  
    implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk7:$kotlin_version"  
    api "org.jetbrains.kotlinx:kotlinx-coroutines-core:$rootProject.coroutines"  
    api "org.jetbrains.kotlinx:kotlinx-coroutines-android:$rootProject.coroutines"  
  
    // UI  
    implementation "androidx.constraintlayout:constraintlayout:$rootProject.constraintLayoutVersion"  
    implementation "com.google.android.material:material:$rootProject.materialVersion"  
  
}
```





dojo / DEPENDENCIAS

build.gradle (Project: whatdidilearn)

Adicione as versões das dependencias que estamos usando:

```
ext {  
    appCompatVersion = '1.2.0'  
    constraintLayoutVersion = '2.0.2'  
    coreTestingVersion = '2.1.0'  
    coroutines = '1.3.9'  
    lifecycleVersion = '2.2.0'  
    materialVersion = '1.2.1'  
    roomVersion = '2.2.5'  
}
```





ROOM

Com Room, os dados são representados em classes de dados, e os dados são acessados e modificados usando chamadas de função. No entanto, no mundo do banco de dados, você precisa de entidades e queries.

Entidade:

representa um objeto ou conceito e suas propriedades para armazenar no banco de dados.

Uma classe de entidade define uma tabela e cada instância dessa classe representa uma linha na tabela. Cada propriedade define uma coluna.





ROOM

Query: é uma solicitação de dados ou informações de uma tabela de banco de dados ou combinação de tabelas, ou uma solicitação para executar uma ação nos dados.

Usamos queries para obter, inserir e atualizar entidades.



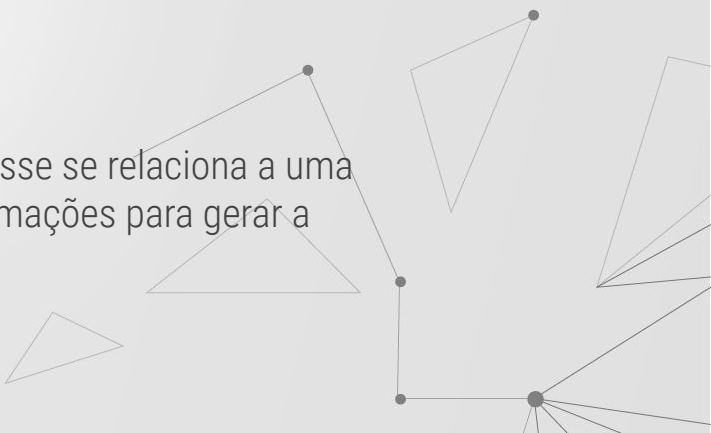


ENTITY

O Objeto **Learned Item modela** os itens aprendidos durante o curso. Para persistir essas informações vamos criar uma tabela no nosso banco de dados chamada "learned_item".

Para tornar a classe Learned Item significativa para um banco de dados da Room, precisamos anotá-la.

As anotações identificam como cada parte desta classe se relaciona a uma entrada no banco de dados. A Room usa essas informações para gerar a estrutura necessária para a gestão da tabela.





dojo / ENTITY

No arquivo `LearnedItem` adicione a anotação `@Entity` logo acima do nome da classe.

Complemente a criação da entity definindo que o nome da tabela de armazenamento das informações será `"learned_item"`.

```
@Entity(tableName = "learned_item")  
data class LearnedItem(  
    ...  
)
```





dojo / ENTITY

Para facilitar a identificação dos itens aprendidos. Adicione o atributo id: Int e o anote como a chave primária desta tabela (adicione no final da lista de atributos)

```
@PrimaryKey(autoGenerate = true)  
val id: Int,
```





dojo / ENTITY

Para facilitar a identificação dos itens aprendidos. Adicione o atributo id: Int e o anote como a chave primária desta tabela (adicione no final da lista de atributos)

```
@PrimaryKey(autoGenerate = true)  
val id: Int,
```





dojo / ENTITY

Complemente a definição da nossa tabela, adicionando `@ColumnInfo` para especificar que o nome das colunas da tabela serão diferentes dos nomes dos atributos usados para modelar a classe.

```
@Entity(tableName = "learned_item")
data class LearnedItem(
    @PrimaryKey(autoGenerate = true)
    @ColumnInfo(name = "item_id")
    val id: Int = 0
    @ColumnInfo(name = "item_title")
    val title: String,
    @ColumnInfo(name = "item_description")
    val description: String,
    @ColumnInfo(name = "item_level")
    val understandingLevel: UnderstandingLevel,
```



ARMAZENANDO DADOS COMPLEXOS

Room suporta apenas tipos primitivos de dados.
Understanding Level é um enum. Temos que explicar como essa informação
deverá ser armazenada.





dojo / CONVERTER

No pacote data, crie a classe Converters. Crie nessa classe dois métodos:

```
fun levelToInt(level: UnderstandingLevel): Int  
fun intToLevel(color: Int): UnderstandingLevel
```

Anote essas classes com @TypeConverter





dojo / CONVERTER

No pacote data, crie a classe Converters. Crie nessa classe dois métodos:

```
fun levelToInt(level: UnderstandingLevel): Int  
fun intToLevel(color: Int): UnderstandingLevel
```

Anote essas classes com @TypeConverter



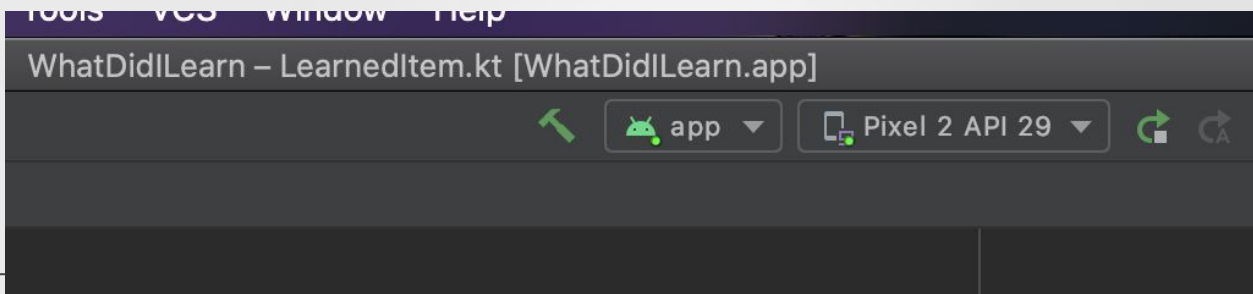
dojo / CONVERTER

```
class Converters {
    @TypeConverter
    fun levelToInt(level: UnderstandingLevel): Int{
        return level.color
    }

    @TypeConverter
    fun intToLevel(color: Int): UnderstandingLevel {
        return when(color) {
            R.color.purple_200 -> UnderstandingLevel.LOW
            R.color.purple_500 -> UnderstandingLevel.MEDIUM
            else -> UnderstandingLevel.HIGH
        }
    }
}
```

dojo / COMMIT

Bulde o projeto para verificar se não há erros e faça um commit.



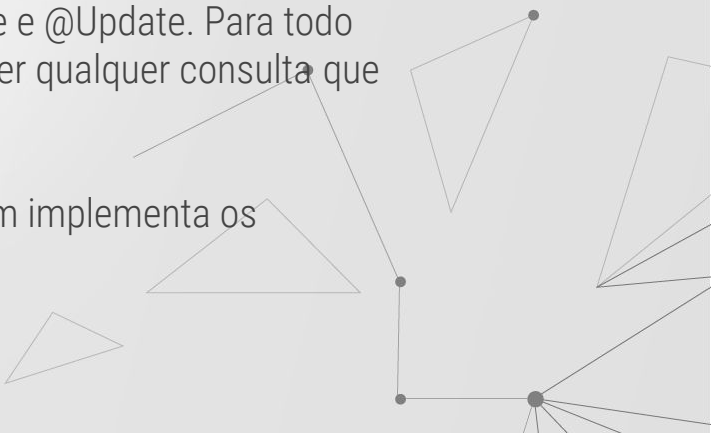


DAOs

DAO (objeto de acesso a dados), fornece métodos convenientes para inserir, excluir e atualizar o banco de dados. Nessa classe, fica especificado consultas SQL associadas à chamadas de métodos.

O compilador verifica o SQL e gera consultas. Para operações comuns, a biblioteca fornece anotações como `@Insert`, `@Delete` e `@Update`. Para todo o resto, existe a anotação `@Query`. É possível escrever qualquer consulta que seja compatível com SQLite.

DAO deve ser uma interface ou classe abstrata (quem implementa os métodos é o próprio Room)



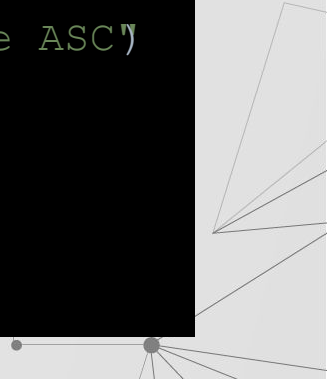


dojo / DAO

- No pacote Data, crie a interface LearnedItemDao. Nessa interface, vamos declarar os métodos insert e getAll() para adicionar novos itens e recuperar todos os itens salvos na base de dados.

```
@Dao
interface LearnedItemDao {
    @Query("SELECT * FROM learned_item ORDER BY item_title ASC")
    fun getAll(): List<LearnedItem>

    @Insert
    fun insert(item: LearnedItem)
}
```



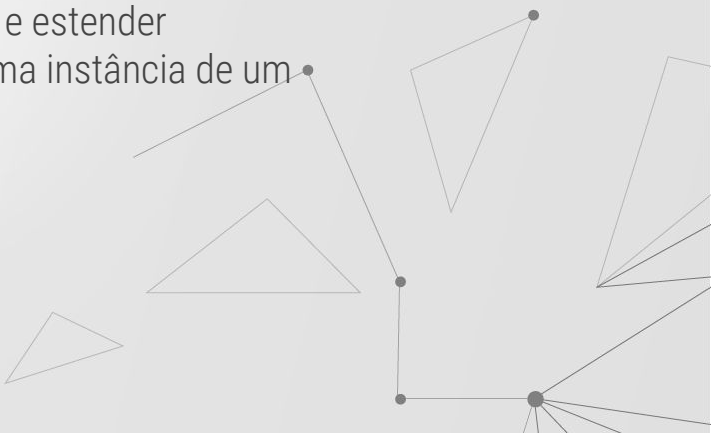


DATABASE

Room é uma camada de banco de dados sobre um banco SQLite.

- A Room usa o DAO para fazer consultas ao seu banco de dados.
- A Room fornece verificações em tempo de compilação de instruções SQLite.

A classe de banco de dados Room deve ser abstrata e estender RoomDatabase. Normalmente, você só precisa de uma instância de um banco de dados Room para todo o aplicativo.



dojo / DATABASE

```
@Database(entities = [LearnedItem::class], version = 1,  
exportSchema = false)  
@TypeConverters(Converters::class)  
abstract class LearnedItemsDatabase: RoomDatabase() {  
  
    abstract fun learnedItemDao(): LearnedItemDao  
  
    companion object {  
        // Singleton para evitar que multiplas instancias do  
        banco de dados sejam abertas ao mesmo tempo  
        @Volatile  
        private var INSTANCE: LearnedItemsDatabase? = null  
  
    }
```

dojo / DATABASE

```
fun getDatabase(context: Context): LearnedItemsDatabase {  
    // se INSTANCE não é nulo, então retorna ela mesma,  
    // se INSTANCE é nula, então cria uma instancia do banco  
    return INSTANCE ?: synchronized(this) {  
        val instance = Room.databaseBuilder(  
            context.applicationContext,  
            LearnedItemsDatabase::class.java,  
            "learned_item_database"  
        ).build()  
        INSTANCE = instance  
        instance  
    }  
}
```



dojo / GETALL()

- Mova o metodo `getAll()` para o companion object, para que seja possível acessá-lo estaticamente.
- Ajuste a chamada do método `getAll` na MainActivity:

```
val learnedItems = LearnedItemsDatabase.getAll()
```





EXERCÍCIOS

O que MVVM significa?

- Modelos, Variáveis, Valores e Módulos
 - Model, View, ViewModel
 - Módulos, Views, Variables e Mutabilidade
 - Model, Variáveis e ViewModel
-





EXERCÍCIOS

O MVVM é:

- Um paradigma de programação
 - Um padrão arquitetural
 - Um modelo para criação de interfaces amigáveis
 - Framework para testes
-





EXERCÍCIOS

No padrão de arquitetura recomendado para aplicativos android, temos as seguintes camadas:


- Activities/Fragments, ViewModels, Repositories, Models/Remote data
 - Views, ViewModels, Data
 - Activities/Fragments, Presenters, Interactors e Data
 - Views, Models e Data
-





EXERCÍCIOS


No contexto de banco de dados, as Entities representam:

- Consultas que podemos fazer
 - Estruturas para a migração de banco de dados
 - Objetos
 - ~~Objetos ou conceitos com propriedades~~ que são armazenados como tabelas na base de dados
- 



EXERCÍCIOS

O que um DAO representa num banco de dados Room?

- Construtor do banco de dados
 - Um DAO descreve as colunas das tabelas de uma entidade
 - Singleton para mantermos uma única instância de um banco de dados
 - Um DAO fornece métodos para manipulação da base de dados
- 

DESAFIO

Ajuste a tela inicial do app:

As seguintes cores foram usadas:

```
<color name="purple_200">#FFBB86FC</color>
```

```
<color name="purple_300">#984EF3</color>
```

```
<color name="purple_500">#FF6200EE</color>
```

```
<color name="purple_700">#FF3700B3</color>
```

```
<color name="purple_900">#FF1F0161</color>
```

Dica:

Adicione a dependencia da Material Components e ajuste o arquivo themes.xml (ou styles.xml) para que o fab e as barras superiores mudem suas cores.

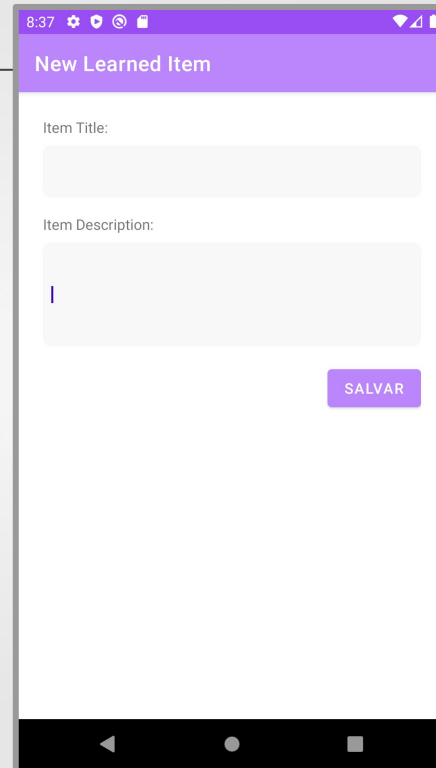


DESAFIO

Ao clicar no botão, a seguinte tela é apresentada:

Dica:

Para ajustar o título da tela, no método onCreate() use `supportActionBar?.title="New Learned Item"`
Use estilos para reduzir a quantidade de código duplicado no arquivo de layout



Palavras chave da aula de hoje:

ROOM
ENTITIES
QUERIES
SQLITE
DAO
SINGLETON
VIEWBINDING
ANDROID JETPACK
SAFETY
FINDVIEWBYID
MVVM

ARCH COMPONENTS
VIEWMODEL
VIEW
REPOSITORY

CREDITS: This presentation template was created by **Slidesgo**, including icons by **Flaticon**, and infographics & images by **Freepik**.

Please keep this slide for attribution.