

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 solid and some outlined. The overall aesthetic is modern and technical.

PROGRAMA DEV VENTURE

Desenvolvimento Android

**REVISÃO
COROUTINES**

Comunicação com
Internet

01

VIEWMODEL

Dados para UI

02

**INJEÇÃO DE
DEPENDÊNCIAS**

Koin

01

Aula 17 AGENDA



01

REVISÃO





REVISÃO

COROUTINES

JOB

SCOPE

DISPATCHER

SUSPEND

MULTI THREADING



02

MVVM - VIEW MODEL





VIEWMODEL

A classe **ViewModel** foi projetada para armazenar e gerenciar dados relacionados à IU considerando o ciclo de vida. A classe ViewModel permite que os dados sobrevivam às mudanças de configuração, como a rotação da tela.





VIEWMODEL X ANDROID VIEWMODEL

AndroidViewModel vem com o contexto do aplicativo, o que é útil se você precisar de contexto para obter um serviço do sistema ou tiver um requisito semelhante





dojo / LEARNEDITENSVIEWMODEL

A ViewModel é a classe responsável por "segurar" os dados que a activity/fragment precisa.

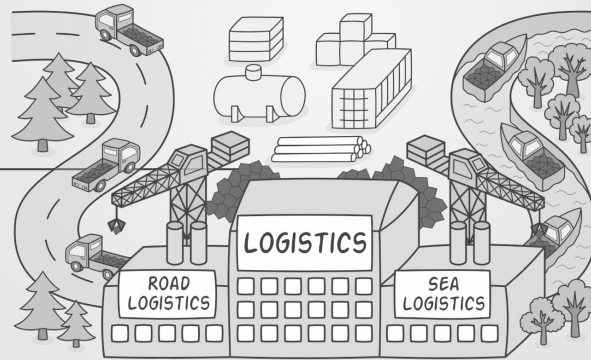
- Crie um novo pacote: viewmodel e dentro dele crie a classe LearnedItemViewModel

```
class LearnedItemViewModel(private val dao: LearnedItemDao) : ViewModel() {
```



PADRÃO FACTORY

O Factory é um padrão de design que fornece uma interface para a criação de objetos.





dojo / VIEWMODEL FACTORY

Usaremos a super classe ViewModelProvider.Factory para criar nossa "fábrica" de LearnedItemViewModel.

- Crie dentro do pacote viewmodel a classe LearnedItemViewModelFactory

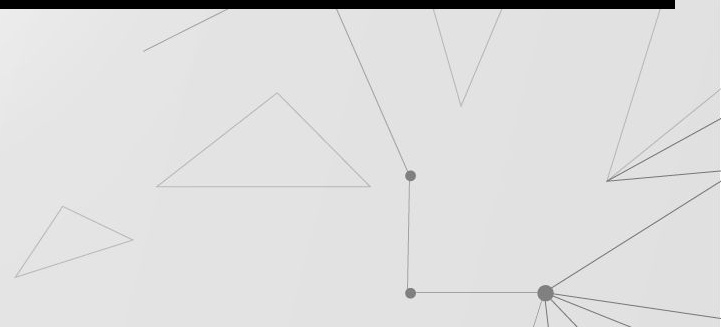
```
class LearnedItemViewModelFactory(private val dao: LearnedItemDao) :
    ViewModelProvider.Factory {
    override fun <T : ViewModel?> create(modelClass: Class<T>): T {
        if
        (modelClass.isAssignableFrom(LearnedItemViewModel::class.java)) {
            return LearnedItemViewModel(dao) as T
        }
        throw IllegalArgumentException("Unknown ViewModel class")
    }
}
```



dojo / VIEWMODEL FACTORY

- Precisamos vincular nosso viewmodel na activity.

```
val viewModelFactory = LearnedItemViewModelFactory(learnedItemsDao)
val viewModel = ViewModelProvider(this,
viewModelFactory).get(LearnedItemViewModel::class.java)
```





dojo / DADOS DA VIEWMODEL

- Refatore o código para que as informações mostradas da tela sejam puxadas a partir da viewmodel:

```
val learnedItems = viewModel.learnedItems
```



03

INJEÇÃO DE DEPENDÊNCIAS



```
class class LearnedItemRepository {  
    fun getItem(): List<LearnedItem> {  
        val database = LearnedItemDataBase.getDatabase(context)  
        val dao = database.learnedItemDao  
        return learnedItemDao.getAll()  
    }  
}
```

```
class LearnedItemRepository(  
    learnedItemDao: LearnedItemDao  
) {  
  
    fun getItem(): List<LearnedItem> {  
        return learnedItemDao.getAll()  
    }  
  
}
```

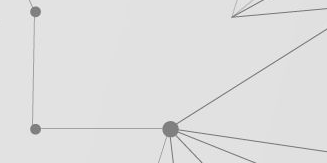


DEPENDENCY INJECTION

Injeção de dependência (Dependency Injection, em inglês) é um padrão de desenvolvimento de programas de computadores utilizado para **reduzir o nível de acoplamento entre diferentes partes de um sistema.**



A stylized illustration of a network graph. It features a series of black dots (nodes) connected by thin grey lines (edges). The nodes are arranged in a roughly triangular pattern, with one node at the top, two in the middle, and one at the bottom. The lines connect these nodes in a way that creates a complex web of triangles. At the bottom center of the image, there is a cartoon illustration of a person's head with brown hair and a neutral expression. The background is a light grey color.



FRAMEWORKS

Para simplificar ainda mais o trabalho, podemos contar com auxílio de ferramentas que apoiam a injeção de dependências.



Dagger



Koin



Hilt



FRAMEWORKS

Dagger	Koin	Hilt
DI pattern	Service Locator Pattern	DI pattern
Compile time	Execution time	Compile time
Bigger impact on build time	None impact on build time	Bigger impact on build time
Smaller impact on run time	Bigger impact on run time	Smaller impact on run time
Open source	Open source	Open source



KOIN





DEPENDÊNCIAS

<https://gist.github.com/marcellalcs/2634073b7cfa004e3fd2805028d50f42>






dojo / MODULE

Crie um objeto `LearnedItemModule` e chame a função `module` para declararmos os componentes da nossa aplicação

```
object LearnedItemModule {  
  val module = module { this: Module  
}
```

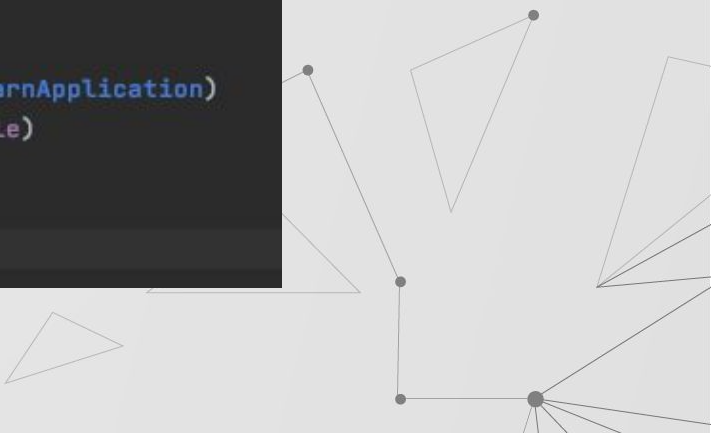




INICIANDO O KOIN

Na classe `LearnedItemApplication`, inicialize o koin

```
class WhatDidILearnApplication: Application() {  
    override fun onCreate() {  
        super.onCreate()  
        startKoin{ this: KoinApplication  
            androidLogger()  
            androidContext(this@WhatDidILearnApplication)  
            modules(LearnedItemModule.module)  
        }  
    }  
}
```



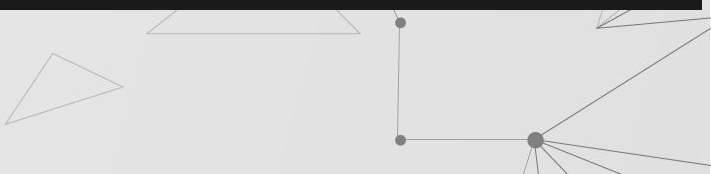


MODULE

Um módulo Koin reúne definições que você injetará / combinará em seu aplicativo. Para criar um novo módulo, basta usar a seguinte função:

module { // module content }

```
factory { } //fornece uma factory para a classe descrito  
single { } //fornece um singleton para a classe descrito (also aliased as bean)  
get() - //encontra a dependência de um componente
```



dojo / MODULE

Defina como os objetos serão construídos usando as funções factory, single e get

```
val module = module { this: Module  
    factory { CoroutineScope(Dispatchers.IO) }  
  
    single { this: Scope  
        LearnedItemDatabase.getDatabase(get(), get())  
    }  
  
    single { this: Scope  
        get<LearnedItemDatabase>().LearnedItemDao()  
    }  
  
    factory { LearnedItemRepository(get()) }  
  
    viewModel { MainViewModel(get()) }  
}
```




VIEWMODEL

Usando `by viewModel()` todo o trabalho que tivemos com o `ViewModelFactory` poderá ser removido. O Koin (add a dependência `koin-viewmodel`) cuidará desta parte.

```
class MainActivity : AppCompatActivity() {  
    private lateinit var binding: ActivityMainBinding  
    private val viewModel: MainViewModel by viewModel()  
}
```

