MVVM in Kotlin: ViewModel + LiveData

Quem somos

C . E . S . A . R

ehammo

Eduardo Maia

Patrick Steiger

psteiger

Engenheiros de Software - CESAR

Android Embarcado

Android Associate Developers

C.E.S.A.R

Motivation

Google I/O 2017

Pontos de dores dos devs

1. **Gerenciamento de Ciclo de Vida**

2. **Ausência de Arquitetura Recomendada**

ARQUITETURAS

RIB

VIPER

MVC

MVP

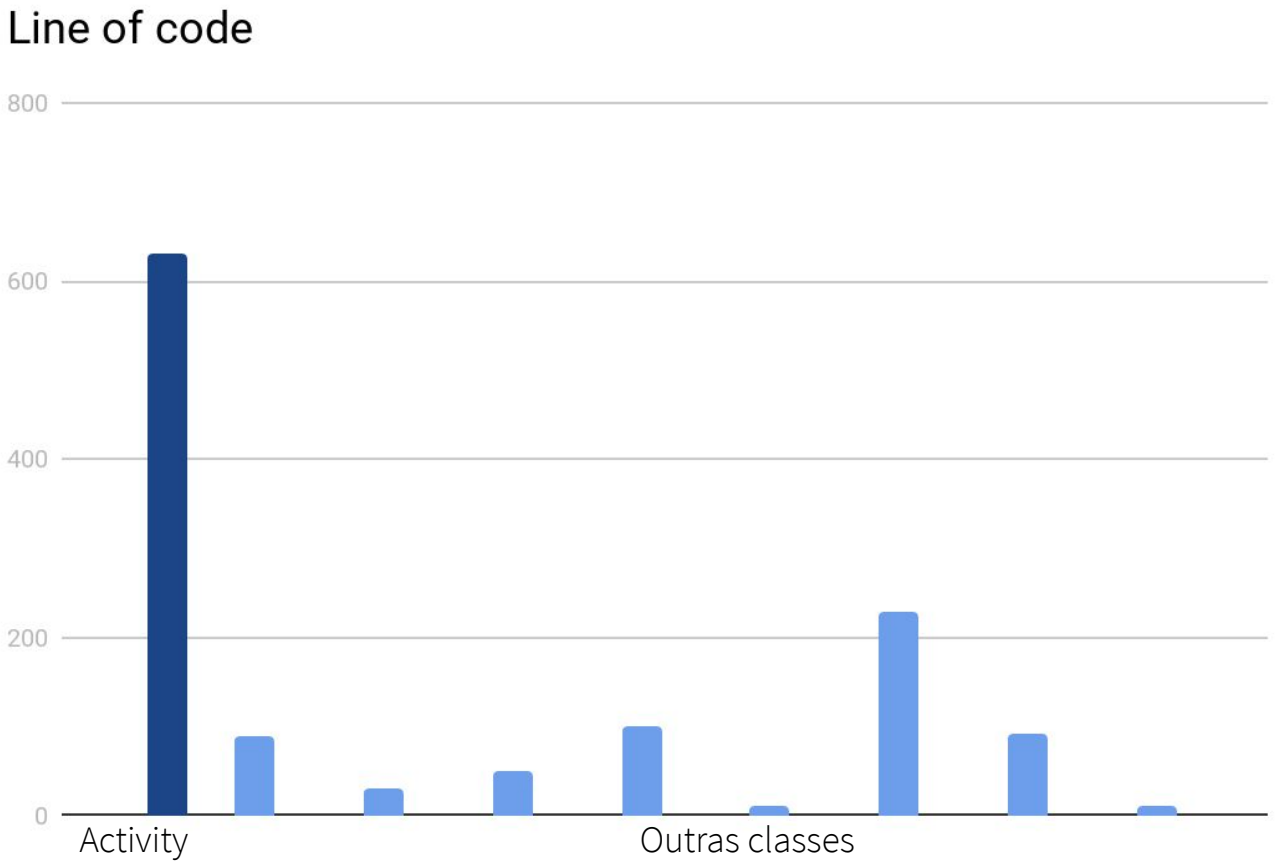
CLEAN

PRNSAASPFRUICC

MVI

REDUX

ARQUITETURAS - God Activity Arch



C . E . S . A . R

Motivation

“God Activity” Architecture

Anti-pattern God Object

Difícil de testar

Difícil de manter

Difícil de criar a atividade

Motivation

Google I/O 2017

Google introduz componentes
LiveData & ViewModel
em uma arquitetura
MVVM

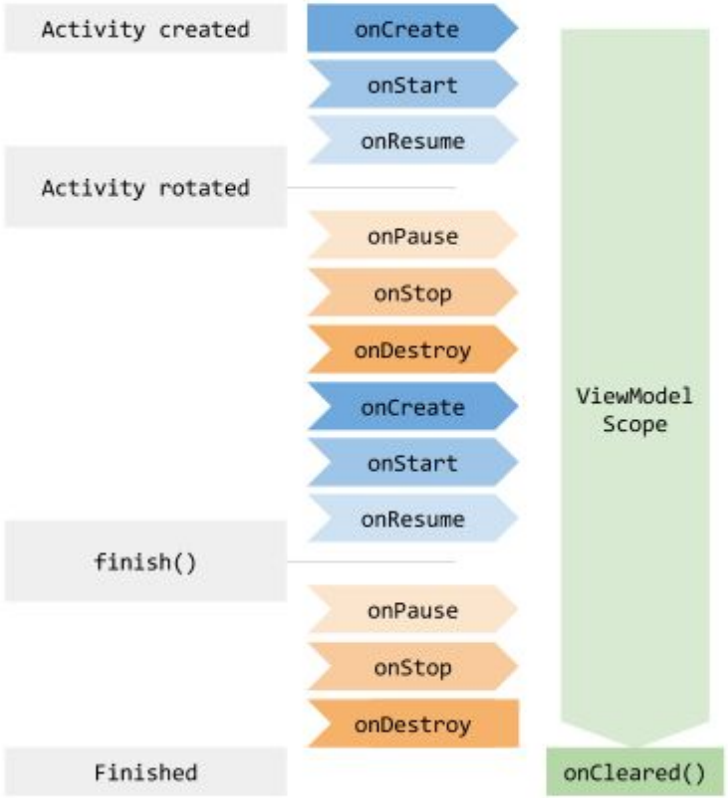
Jetpack
Architecture
Components

C.E.S.A.R

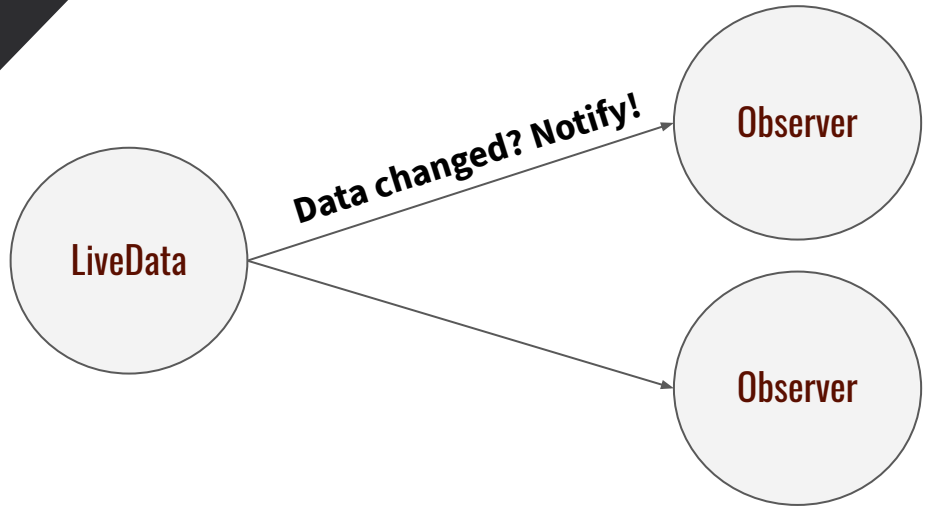
ViewModel LifeCycle-Aware

Sobrevive à **recriação**

Se **destrói** ao **fim**



LiveData LifeCycle-Aware

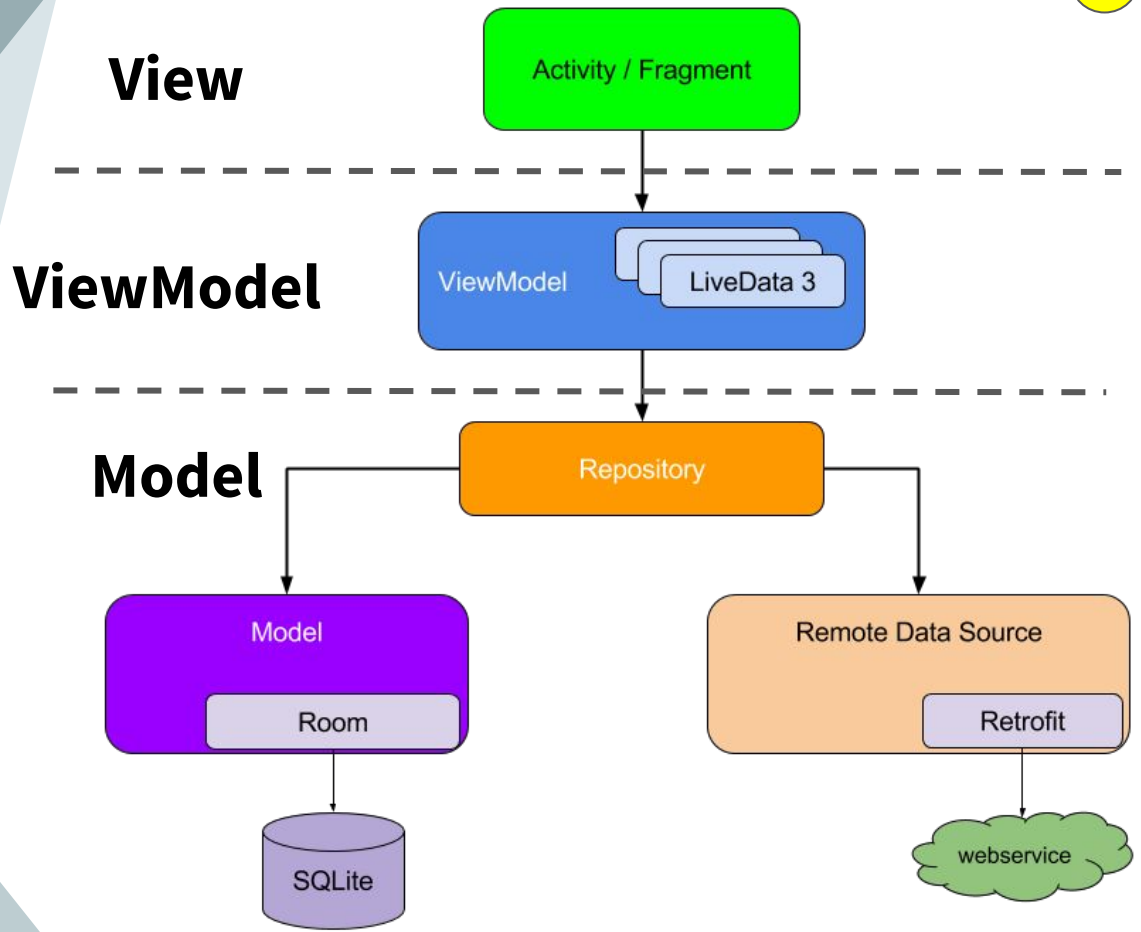


Notifica apenas

observadores ativos

STARTED or RESUMED

MVVM


View
Activity / Fragment
ViewModel
ViewModel
LiveData 3
Model
Repository
Model
Room
SQLite
Remote Data Source
Retrofit
webservice

C . e . S . A . R

View:

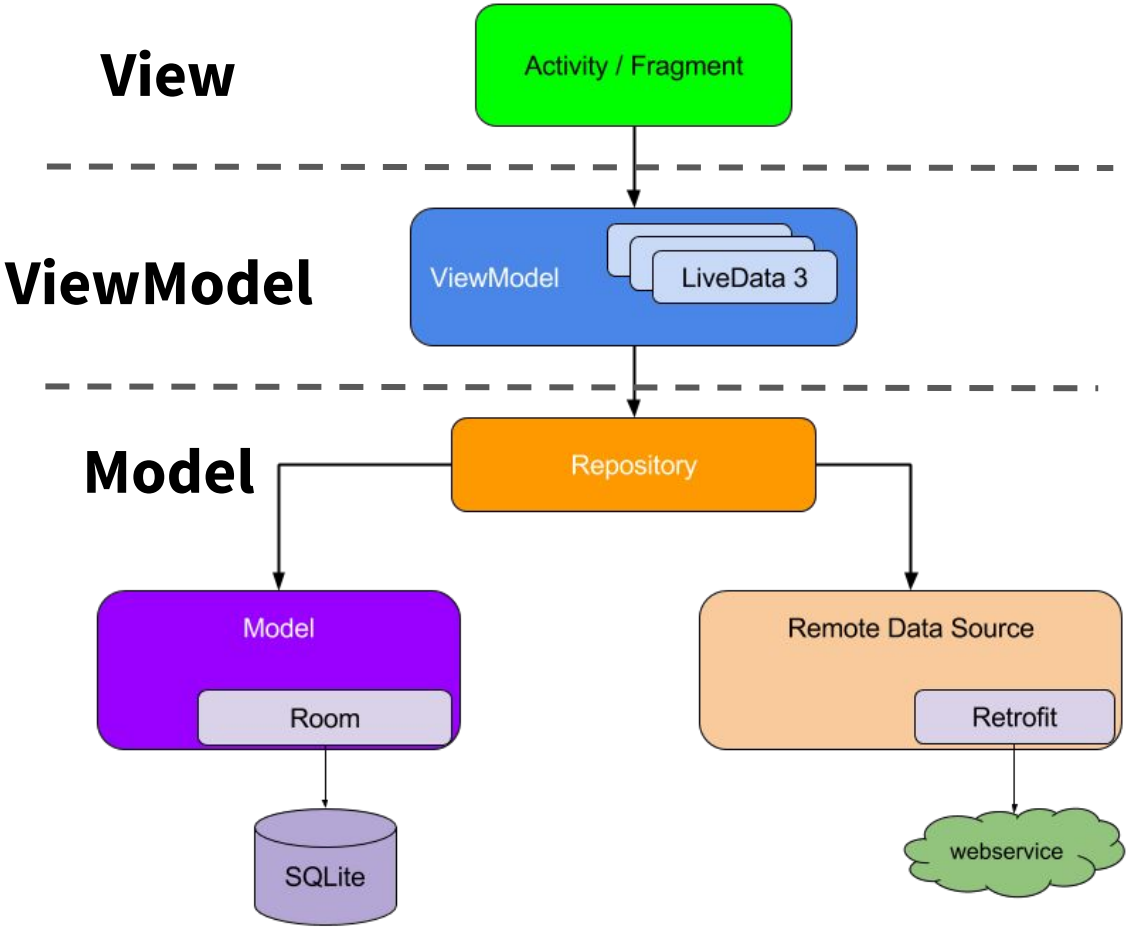
- Lógica de UI

ViewModel:

- Retém Dados
- Notifica View

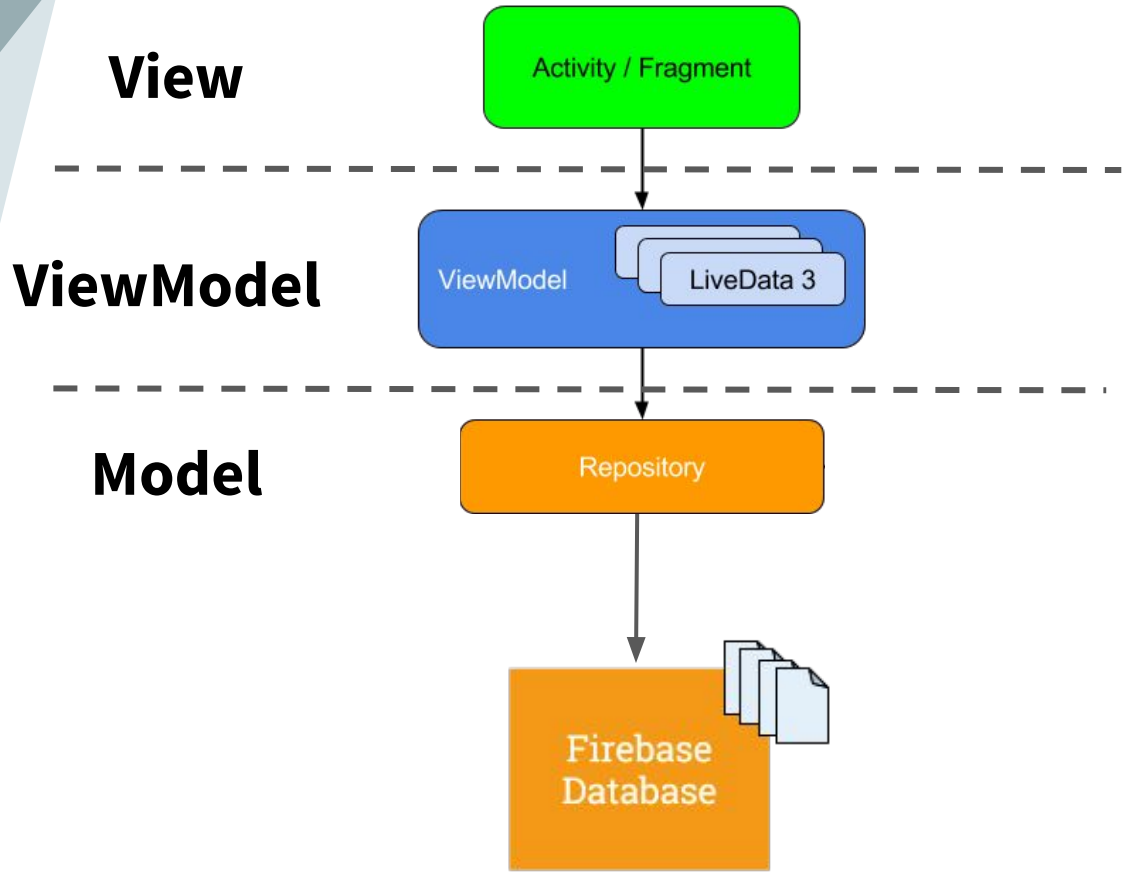
Model:

- Lógica de Negócio
- Modelagem dos Dados
- Armazenamento dos Dados
- Entrega dados (ViewModel)



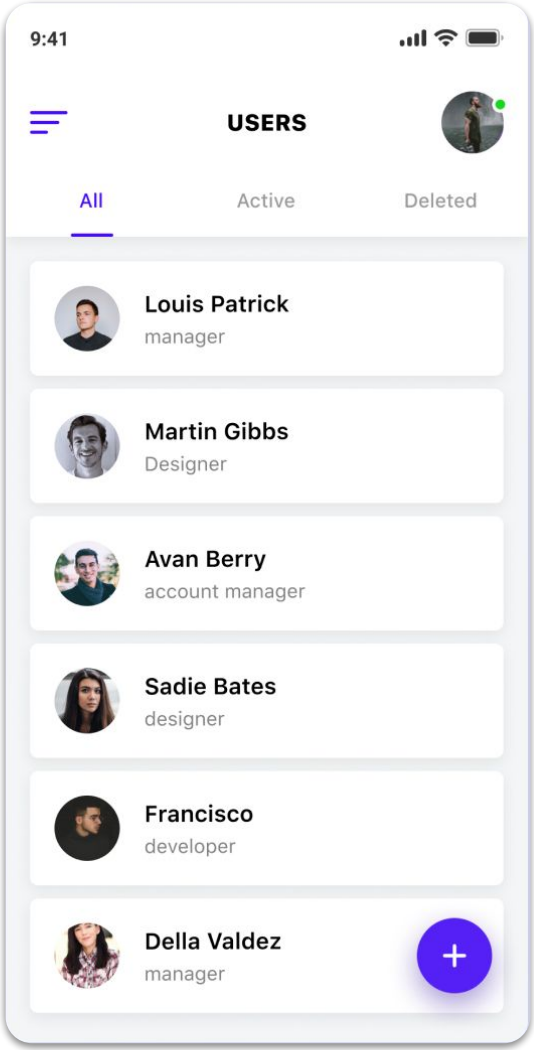
C . E . S . A . R

Construiremos



C . e . S . A . R

Caso de Uso



C . e . S . A . R


“Talk is cheap.
Show me
the code”
—Linus Torvalds

View

Activity

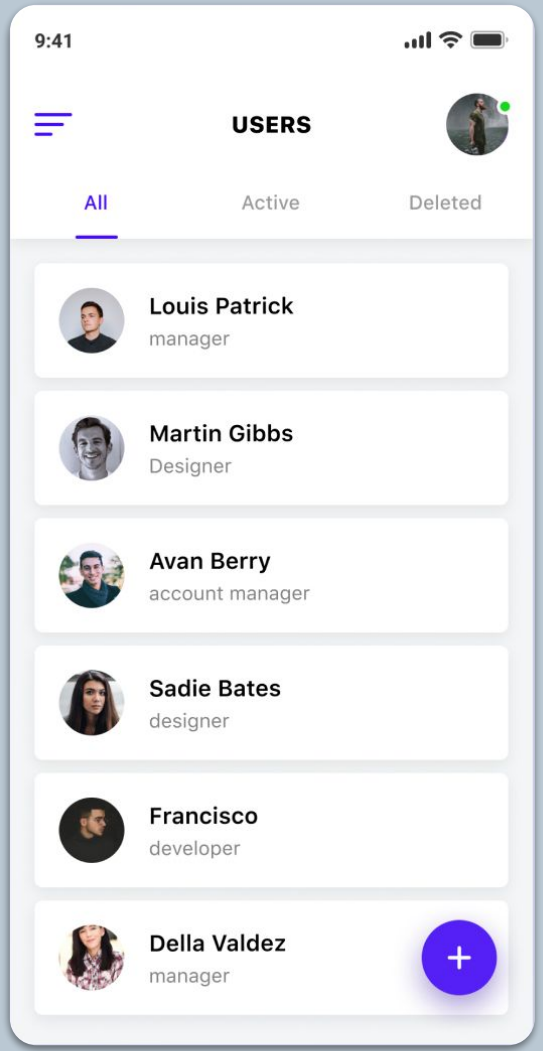
Nosso **objetivo final** é usar:

MyActivity.kt

```
usersLiveData.observe(this, Observer {
    it.onSuccess { users ->
        usersLoaded(users)
    }.onFailure { lastValidUsers, error ->
        usersLoadingFail(lastValidUsers, error)
    }
})
```

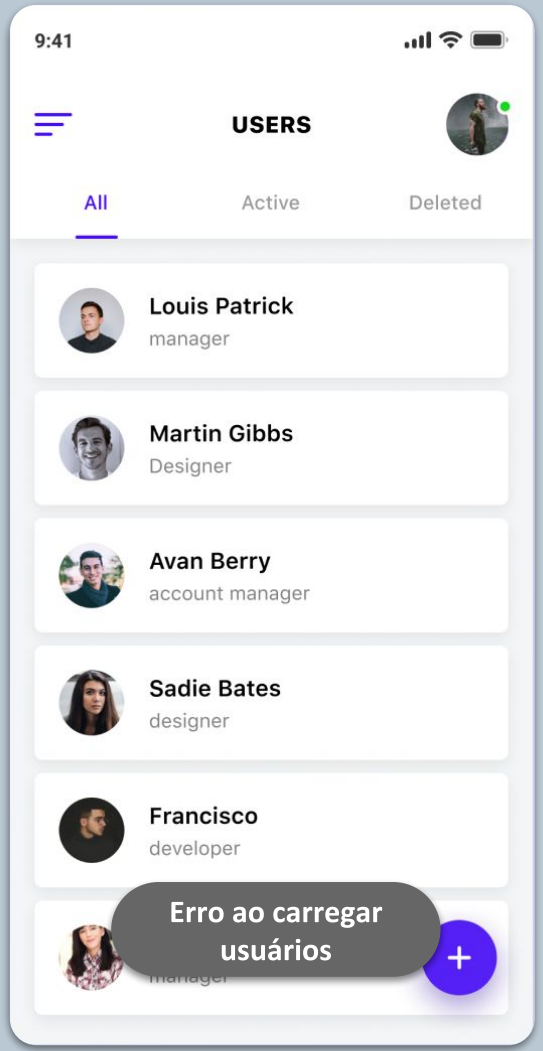
C.E.S.A.R

onSuccess { ... }



C.e.S.A.R

onFailure { ... }



C.E.S.A.R

Model

Primeiro passo

Construir nosso
modelo de dados

C.E.S.A.R

Model

Data Class

Kotlin's data class

User.kt

```kotlin
data class User(
    var name: String? = null
    var email: String? = null
)
```

C.e.S.A.R

Model

Data Class

Kotlin's data class

`toString()`

```
User(name=Patrick, email=...)
```

`copy()`

```
user.copy(email=...)
```

`equals()/hashCode()`

`getters & setters`

...

C.e.S.A.R

Model Resource

Todos nossos dados tem um **estado** relativo ao seu carregamento na fonte:

`Success`

`Failure`

C.E.S.A.R

Model

Resource

Kotlin's sealed class

Resource.kt

```kotlin
sealed class Resource<T>(open val data: T) {

  data class Success<T>(override val data: T) : Resource<T>()

  data class Failure<T>(override val data: T? = null,
                        val error: Throwable? = null) : Resource<T>()
}
```

Model

Resource

Kotlin's sealed class

Resource.kt

```
sealed class Resource<T>(open val data: T) {

  data class Success<T>(override val data: T) : Resource<T>()

  data class Failure<T>(override val data: T? = null,
                        val error: Throwable? = null) : Resource<T>()
}
```

Model

Resource

C.e.S.A.R

Generics

Resource.kt

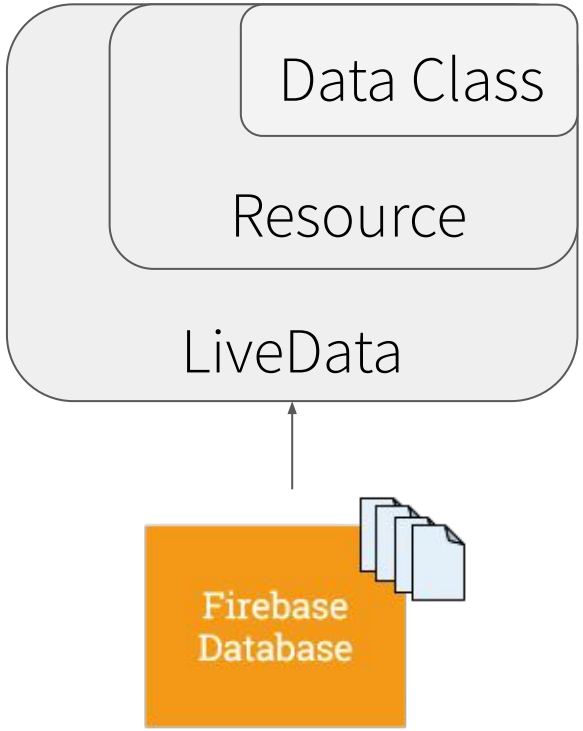
```
sealed class Resource<T>(open val data: T) {

  data class Success<T>(override val data: T) : Resource<T>()

  data class Failure<T>(override val data: T? = null,
                        val error: Throwable? = null) : Resource<T>()
}
```

Model

LiveData



C.E.S.A.R

LiveData de Recurso Genérico

FirebaseResourceLiveData.kt

```kotlin
class FirebaseResourceLiveData<T>(path: String, type: GenericTypeIndicator<T>)
        : LiveData<Resource<T>>() {

    private val ref = FirebaseDatabase.getInstance().getReference(path)

    private val listener = object : ValueEventListener {
        override fun onDataChange(snap: DataSnapshot) {
            value = Resource.Success(snap.getValue(type))
        }

        override fun onCancelled(error: DatabaseError) {
            value = Resource.Failure(value?.data, error.toException())
        }
    }

    override fun onActive() {
        ref.addValueEventListener(listener)
    }

    override fun onInactive() {
        ref.removeEventListener(listener)
    }
}
```

LiveData de Recurso Genérico

FirebaseResourceLiveData.kt

```
// >= 1 observador ativo
override fun onActive() {
    ref.addValueEventListener(listener)
}

// Nenhum observador ativo
override fun onInactive() {
    ref.removeEventListener(listener)
}
```

LiveData de Recurso Genérico

FirebaseResourceLiveData.kt

```
private val listener = object : ValueEventListener {
  override fun onDataChange(snap: DataSnapshot) {
      value = Resource.Success(snap.getValue(type))
  }

  override fun onCancelled(error: DatabaseError) {
      value = Resource.Failure(value?.data, error.toException())
  }
}
```

Resource Repository ...

ResourceRepository.kt

```
class ResourceRepository<T>(path: String,
                            type: GenericTypeIndicator<T>) {

    val allItems = FirebaseResourceLiveData(path, type)

}
```

... usá-lo no ViewModel

C.E.S.A.R

UsersViewModel.kt

```
class UsersViewModel : ViewModel() {

    val usersLiveData =
        ResourceRepository(
            "users", // path
            object : GenericTypeIndicator<HashMap<String, User>>() {} // type
        ).allItems

}
```

... usá-lo no ViewModel

C.E.S.A.R

UsersViewModel.kt

```
class UsersViewModel : ViewModel() {

    val usersLiveData =
        ResourceRepository(
            "users", // path
            object : GenericTypeIndicator<HashMap<String, User>>() {} // type
        ).allItems

}
```

... usá-lo no ViewModel


```
// Extension Function on Generic T type
// Use: "a string".getType()

fun <T> T.getType() {
    return object : GenericTypeIndicator<T>() {}
}
```

... usá-lo no ViewModel

C.E.S.A.R

```
// Extension Function on Generic T type
// Use: "a string".getType()

fun <T> T.getType() =
            object : GenericTypeIndicator<T>() {}
```

... usá-lo no ViewModel

C.E.S.A.R

```
// Extension Function on Generic T type
// Use: "a string".getType()

val <T> T.type
    get() = object : GenericTypeIndicator<T>() {}
```

... usá-lo no ViewModel

UsersViewModel.kt

```
class UsersViewModel : ViewModel() {

    val usersLiveData =
        ResourceRepository(
            "users", // path
            object : GenericTypeIndicator<HashMap<String, User>>() {} // type
        ).allItems

}
```

... usá-lo no ViewModel

UsersViewModel.kt

```
class UsersViewModel : ViewModel() {

    val usersLiveData =
        ResourceRepository(
            "users", // path
            (HashMap<String, User>>()).type
        ).allItems

}
```

Generic Versátil!

C.E.S.A.R

UsersViewModel.kt

```
class CarsViewModel : ViewModel() {

    val carsLiveData =
        ResourceRepository(
            "cars", // path
            (HashMap<String, Car>>()).type
        ).allItems

}
```

View

Activity

Nosso objetivo final

MyActivity.kt

```
usersLiveData.observe(this, Observer {
    it.onSuccess { users ->
        usersLoaded(users)
    }.onFailure { lastValidUsers, error ->
        usersLoadingFail(lastValidUsers, error)
    }
})
```

C.E.S.A.R

View

Activity

Nosso objetivo final

MyActivity.kt

```
usersLiveData.observe(this, Observer {
    it.onSuccess({ users ->
        usersLoaded(users)
    }).onFailure({ lastValidUsers, error ->
        usersLoadingFail(lastValidUsers, error)
    })
})
```

C.E.S.A.R

onSuccess

Resource.kt

```
fun onSuccess(run: (data: T) -> Unit)) {

    if (this is Success) run(data)

    return this

}
```

onFailure

Resource.kt

```
fun onFailure(
    run: (data: T?,
          error: Throwable?) -> Unit
) {

    if (this is Failure) run(data)

    return this

}
```

Obrigado!

Dúvidas?

Implementação completa em:

https://blog.usejournal.com/android-architecture-components-livedata-in-idiomatic-kotlin-cc626819db96

https://github.com/psteiger/Android_MVVM/

C.E.S.A.R

Obrigado!

Dúvidas?

Ou simplesmente em:

C.E.S.A.R