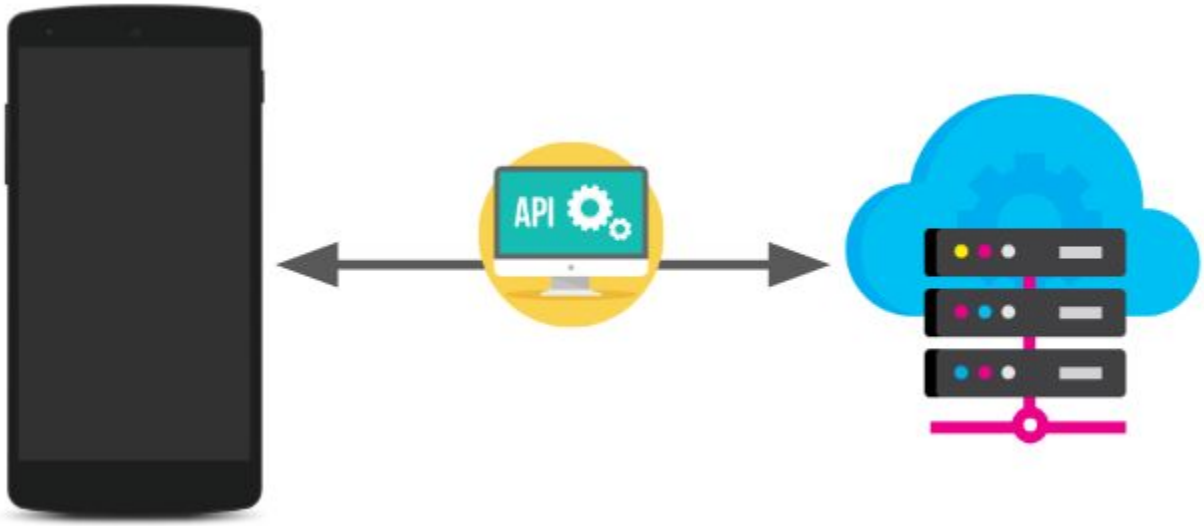

Ame
FastForward



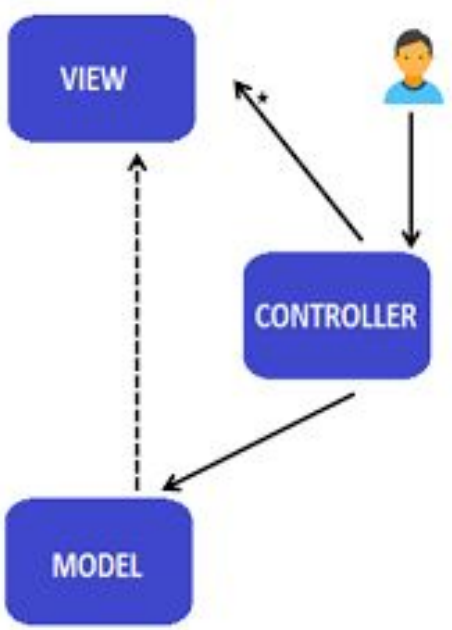
O que veremos hoje?

Agenda de hoje

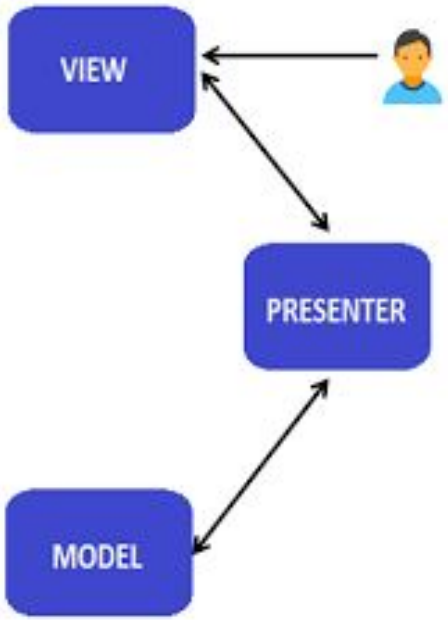
- **Architecture Patterns**
  - MVC, MVP, MVVM
- **Android Jetpack**
  - Lifecycle, ViewModel e Live Data
- **Network**
- **Trabalhando com Listas**
- **Permissões**

Hora de começar

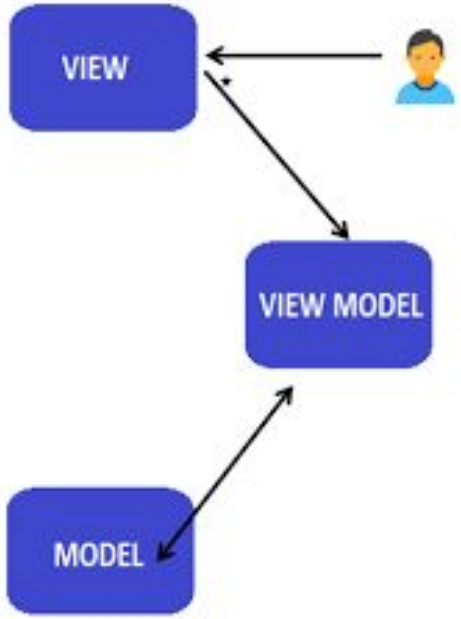
Arquitetura de aplicação



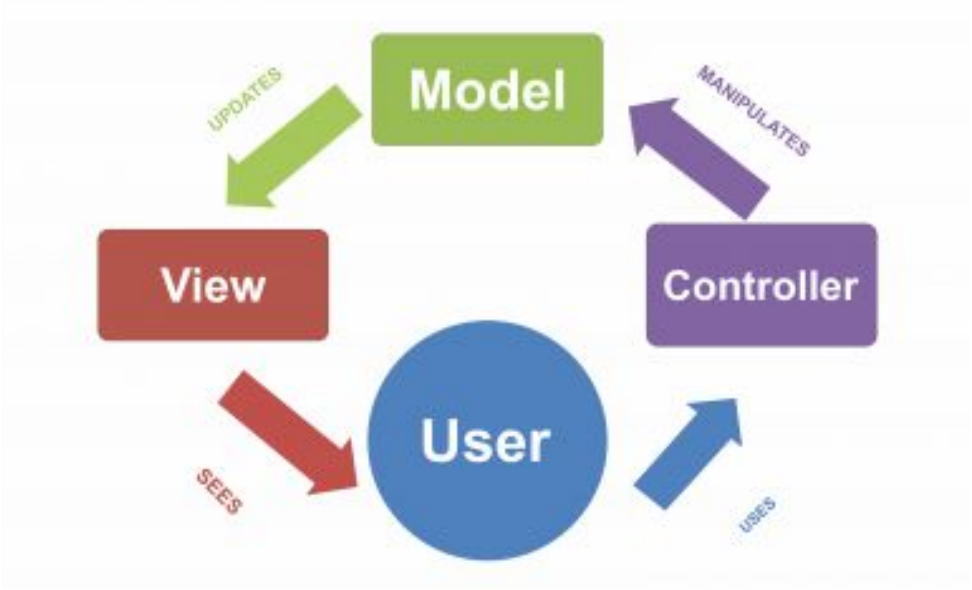
MVC



MVP



MVVM



MVC

MVC

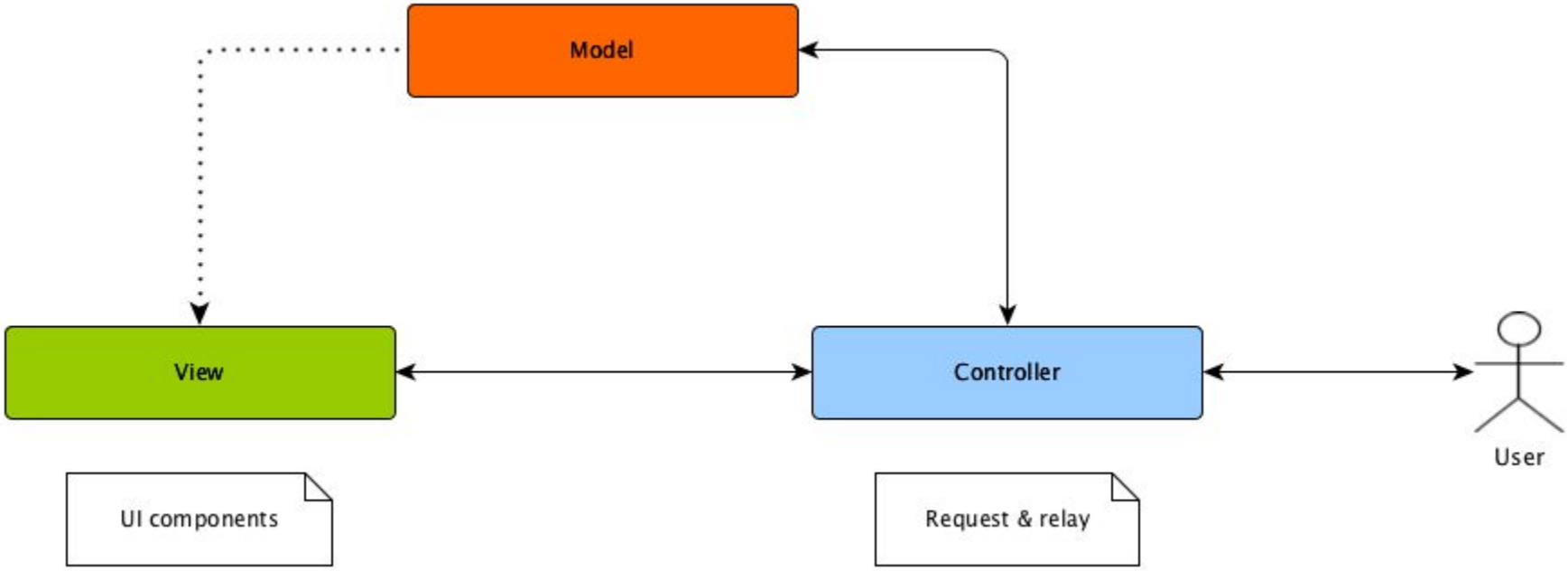
O design pattern MVC divide um aplicativo em três aspectos principais:

Model
View
Controller

Força uma separação de responsabilidade, isso significa que o modelo de domínio e a lógica do controlador são desacoplados da interface do usuário (visualização).

Como resultado, a manutenção e o teste do aplicativo tornam-se mais simples e fáceis

MVC



Model

O modelo representa um conjunto de classes que descrevem a lógica de negócio, isto é, o modelo do negócio, bem como as operações de acesso a dados, ou seja, o modelo de dados.

Ele também define regras de negócios para os dados: como podem ser alterados ou manipulados

View

A View representa os componentes da UI.

É apenas responsável por exibir os dados recebidos do controlador como resultado.

Isso também transforma o (s) modelo (s) em UI.

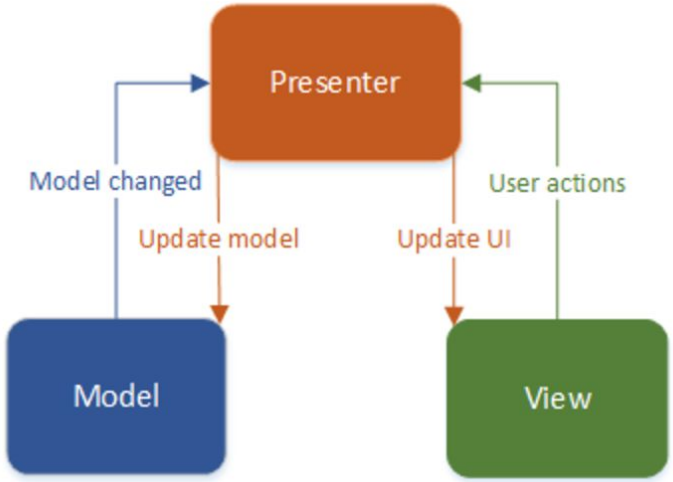
Controller

O **controller** é responsável por processar pedidos recebidos.

Recebe a entrada dos usuários através da **View**, depois processa os dados do usuário com a ajuda do **Model** e passa os resultados para a **View**.

Normalmente, ele atua como coordenador entre a **View** e o **Model**.

MVP


Presenter
Model changed
Update model
Update UI
User actions
Model
View

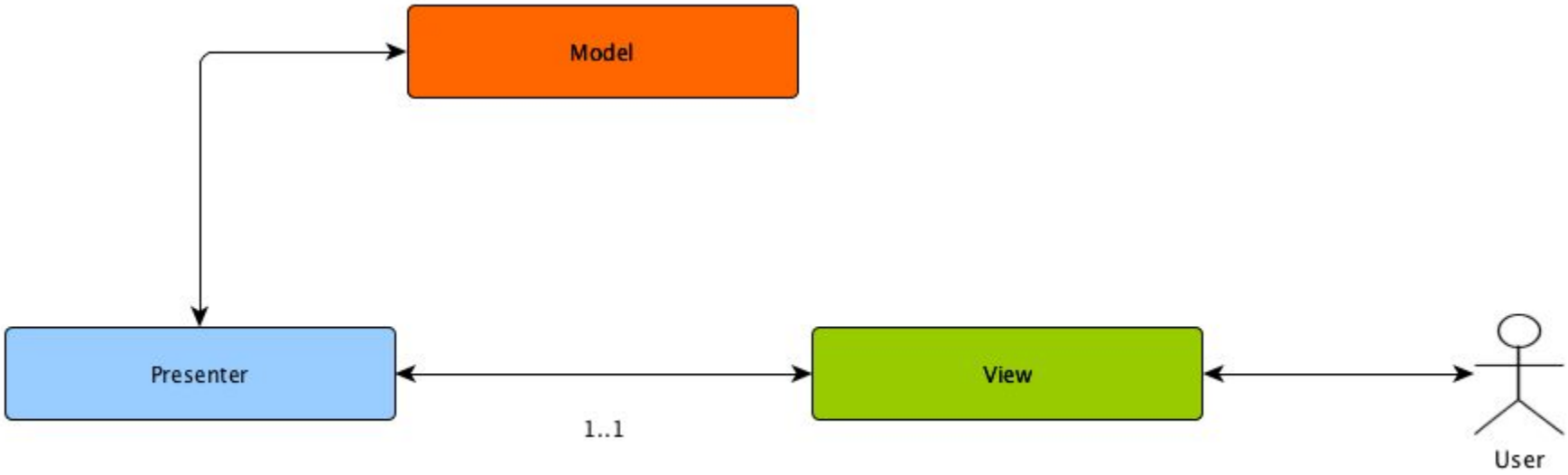
MVP

MVP

Esse padrão é semelhante ao padrão MVC no qual o **controller** foi substituído pelo **presenter**.

Este padrão de design divide um aplicativo em três aspectos principais: Model, View e Presenter

MVP



Model

Representa um conjunto de classes que descreve a lógica e os dados do negócio.

Ele também define regras de negócios para dados significa como os dados podem ser alterados e manipulados.

View

Representa os componentes da UI.

É apenas responsável por exibir os dados recebidos do apresentador como resultado.

Isso também transforma o (s) modelo (s) em UI.

Presenter

É responsável por manipular todos os eventos de UI em nome da visualização, isso é, recebe entrada dos usuários através da **View**, depois processa os dados do usuário com a ajuda do **Model** e passa os resultados de volta para a **View**.

Ao contrário da **view** e do **controller**, a exibição e o **presenter** estão completamente desacoplados uns dos outros e se comunicam uns com os outros por uma interface.

Além disso, o **presenter** não gerencia o tráfego de solicitação recebida como controlador.

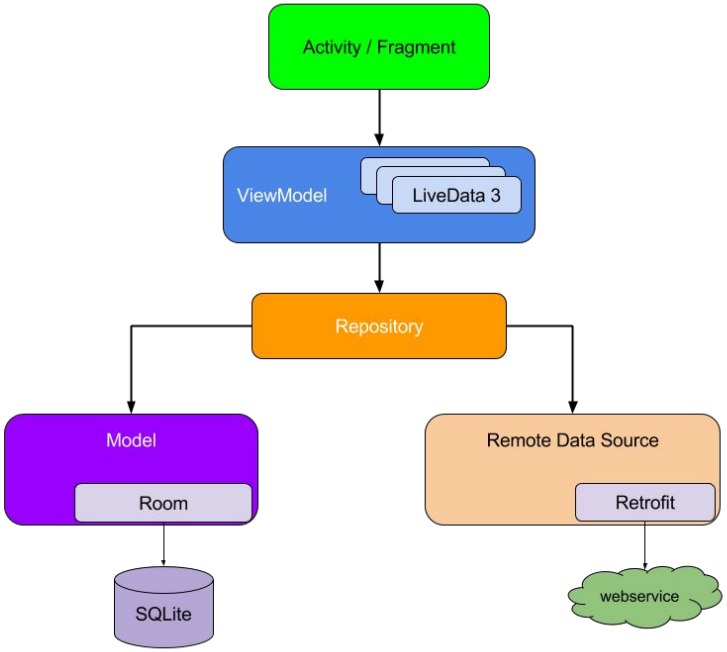
Pontos chaves do MVP

O usuário interage com a **View**.

Existe uma relação um-para-um entre **View** e **Presenter,** ou seja, um **View** é mapeado para apenas um **Presenter**.

A **View** tem uma referência ao **presenter**, mas a exibição não faz referência ao **model**.

Fornece comunicação bidirecional entre **View** e **Presenter**.


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

MVVM

MVVM

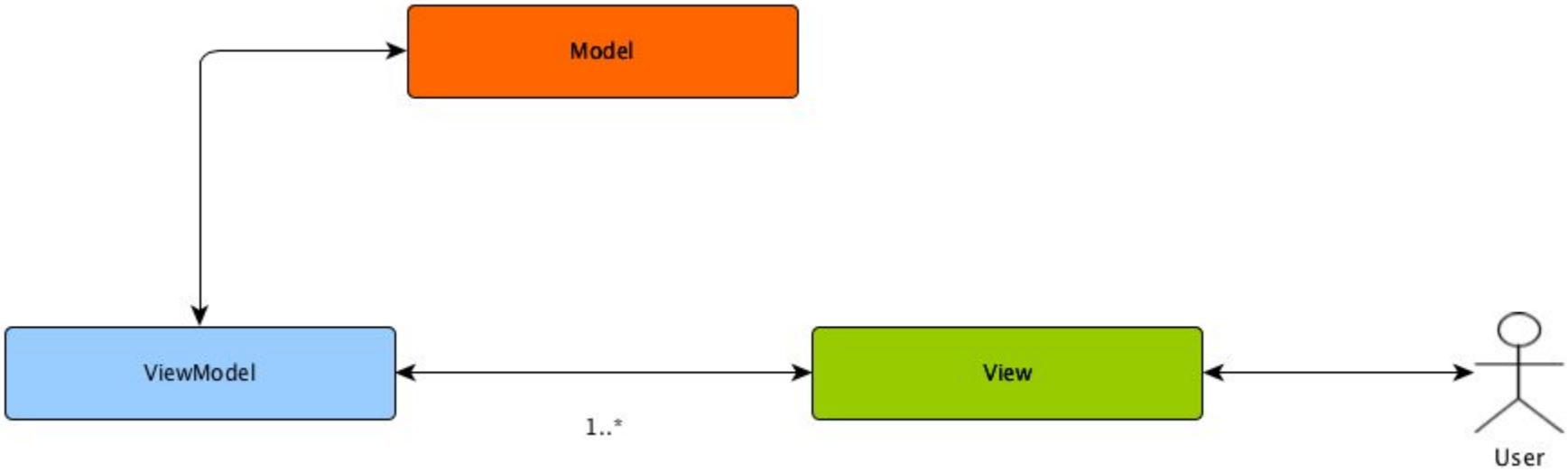
MVVM significa Model-View-ViewModel.

Este padrão suporta ligação bidirecional de dados entre a View e o ViewModel.

Isso permite a propagação automática de mudanças, dentro do modelo de estado de exibição para a **View**.

Normalmente, o modelo de visão usa o padrão **Observer** para notificar mudanças do **ViewModel** para **Model**.

MVVM



Model

Representa um conjunto de classes que descreve a lógica e os dados do negócio.

Ele também define regras de negócios para dados significa como os dados podem ser alterados e manipulados.

View

A Visualização representa os componentes da UI.

É apenas responsável por exibir os dados recebidos do controlador como resultado.

ViewModel

É responsável por expor métodos, comandos e outras propriedades que ajudam a manter o estado da **view**, manipular o **model** como resultado de ações na **view** e desencadear eventos na própria **view**.

Pontos chaves ViewModel

O usuário interage com a **View**.

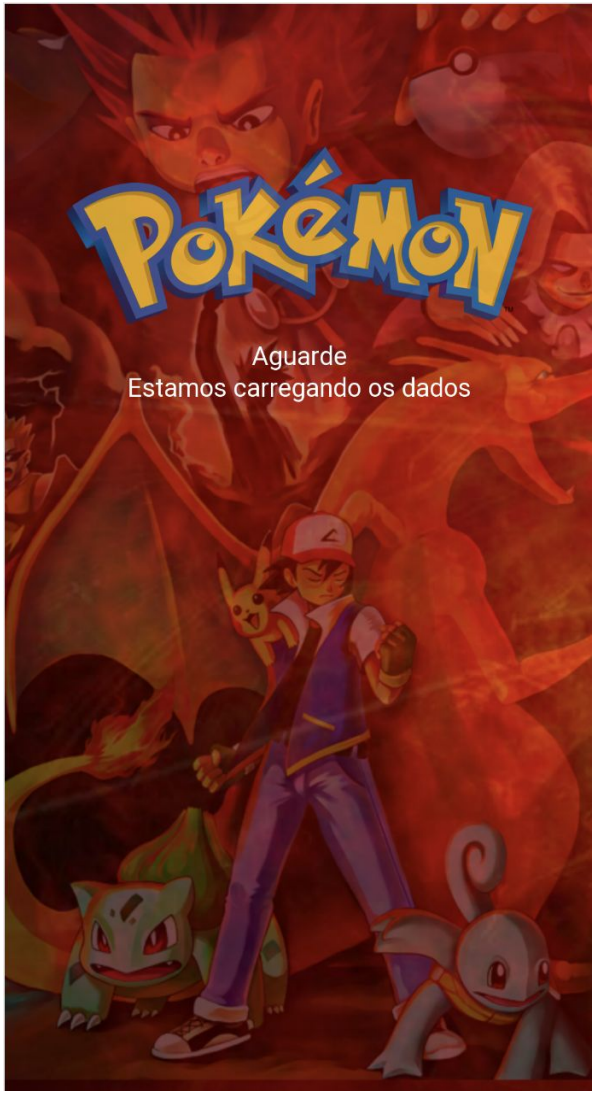
Existe uma relação de muitos para um entre **View** e **ViewModel,** isso significa que muitas **Views** podem ser mapeados para um **ViewModel**.

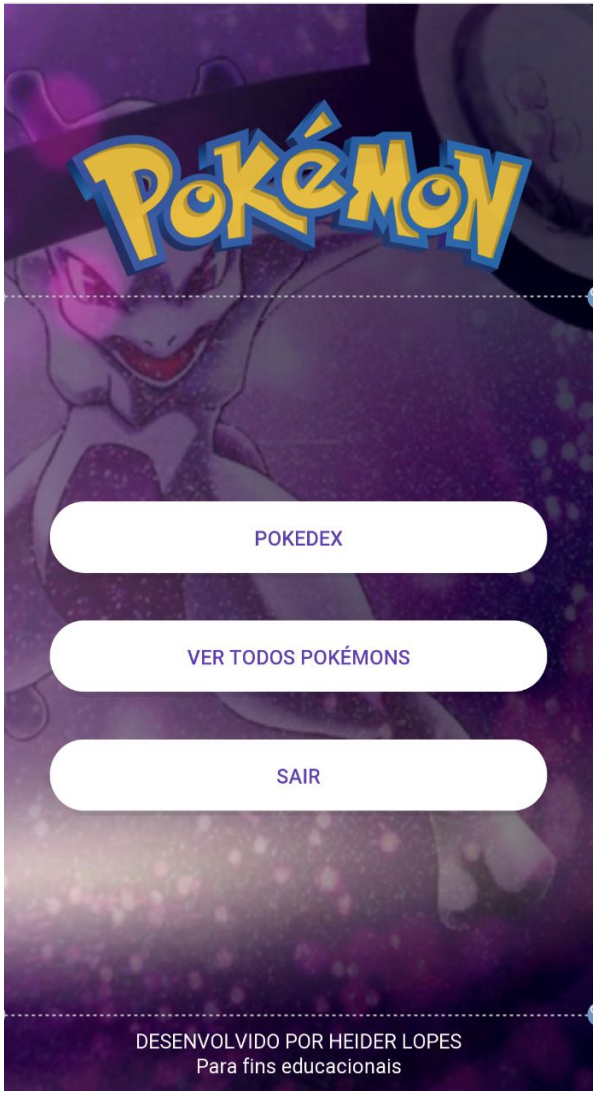
A **view** tem uma referência ao **ViewModel**, mas o **ViewModel** não possui informações sobre a **View**.

Suporta ligação bidirecional de dados entre **View** e **ViewModel**.

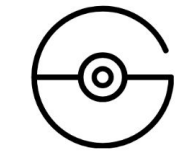
Conhecendo nosso projeto

APP: POKERMAO

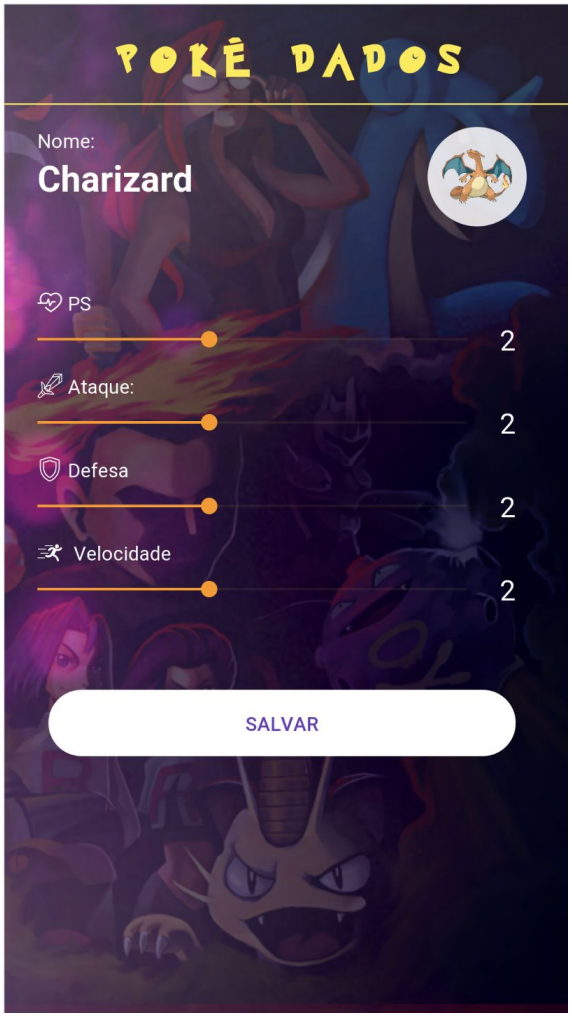




APP: POKERMAO



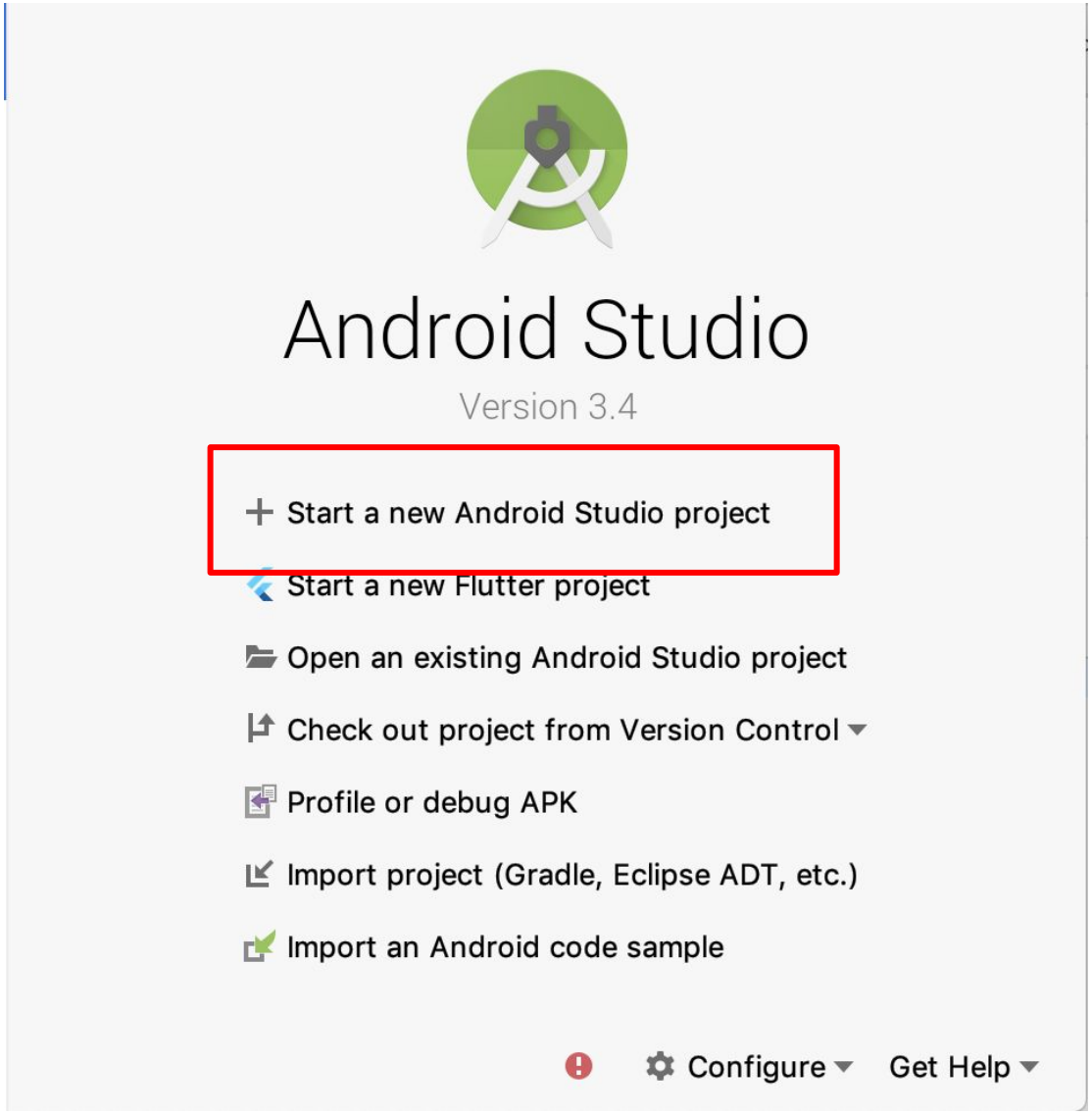
Carregando os dados



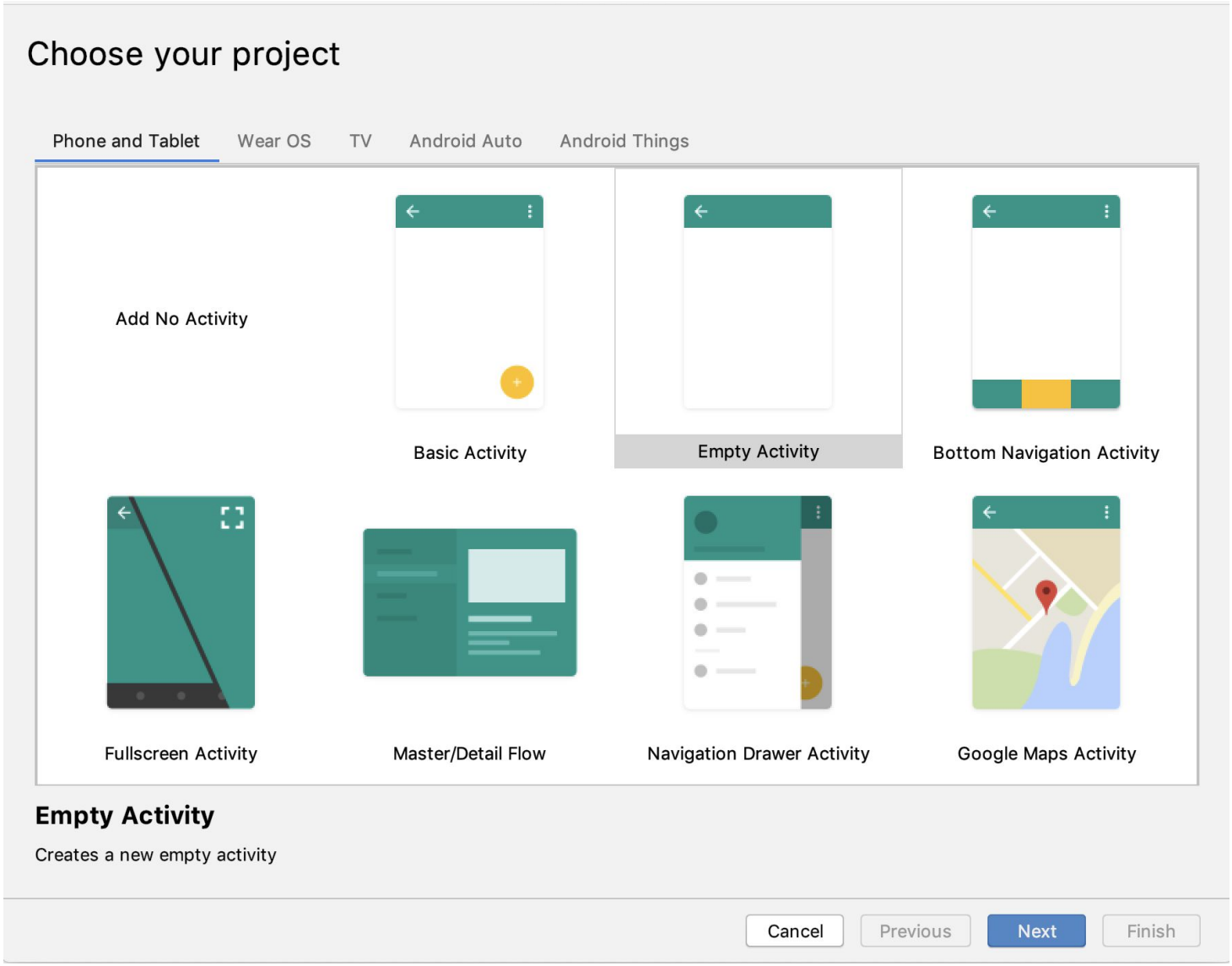
APP: POKERMAO

Criando nosso projeto

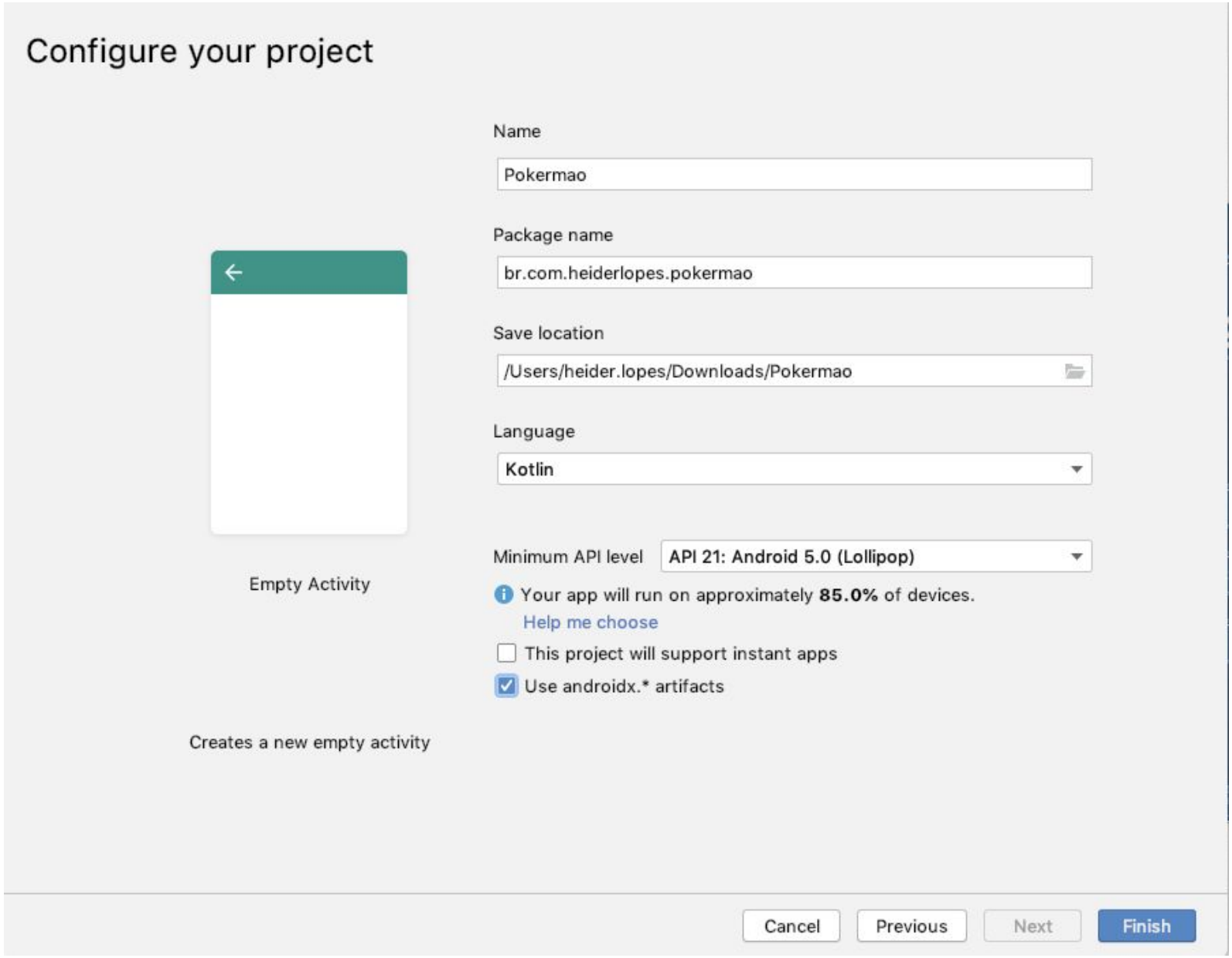
APP: POKERMAO



APP: Pokermao - Criando o projeto



APP: Pokermao - Criando o projeto



APP: Pokermao - Adicionando as primeiras bibliotecas

Abra o arquivo **build.gradle (app)** e dentro de dependencies adicione as bibliotecas abaixo:

```
//Biblioteca para adicionar ler QRCode
implementation "me.dm7.barcodescanner:zxing:1.9.13"

//Biblioteca para adicionar lista performatica ao aplicativo
implementation 'androidx.recyclerview:recyclerview:1.0.0'

//Biblioteca para adicionar cards ao aplicativo
implementation 'androidx.cardview:cardview:1.0.0'

//Biblioteca para adicionar animacoes no projeto
implementation 'com.airbnb.android:lottie:3.0.7'
```

RecyclerView

RecyclerView é uma “evolução” da **ListView** e da **GridView**, componentes presentes desde da primeira versão do Android para se fazer listas e grades de conteúdo. É muito comum aplicativos terem listas para apresentarem seu conteúdo de forma eficiente. E quando mal implementada pelo desenvolvedor pode trazer descontentamento para o usuário. Pensando nisso o Android nos deu um componente poderoso chamado RecyclerView.

RecyclerView - Componentes

RecyclerView: Componente visual que ficará na Activity/Fragment e irá posicionar a lista na tela do usuário.

LayoutManager: É o gerenciador de conteúdo descrito acima. Nele é definido qual é a disposição do itens da lista (se será uma lista vertical ou horizontal, por exemplo).

Adapter: responsável por associar a lista de conteúdo/objeto a view correspondente. Onde cada objeto da lista será um item na lista. É no Adapter onde se define se um item será exibido ou não.

ViewHolder: É a referência para a view que é a parte visual de cada item da lista, que será replicada para todos elementos (na estrutura acima, ficaria dentro do Adapter).

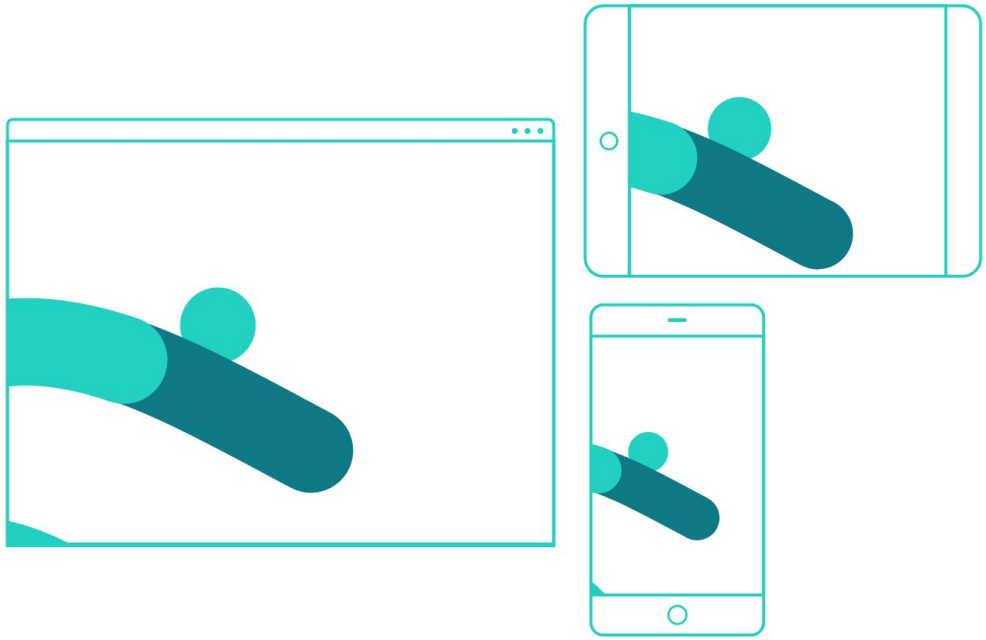
Conhecendo a Lottie

Lottie

Easily add high-quality animation to any native app.

Lottie is an iOS, Android, and React Native library that renders After Effects animations in real time, allowing apps to use animations as easily as they use static images.

Get Started Learn more ›



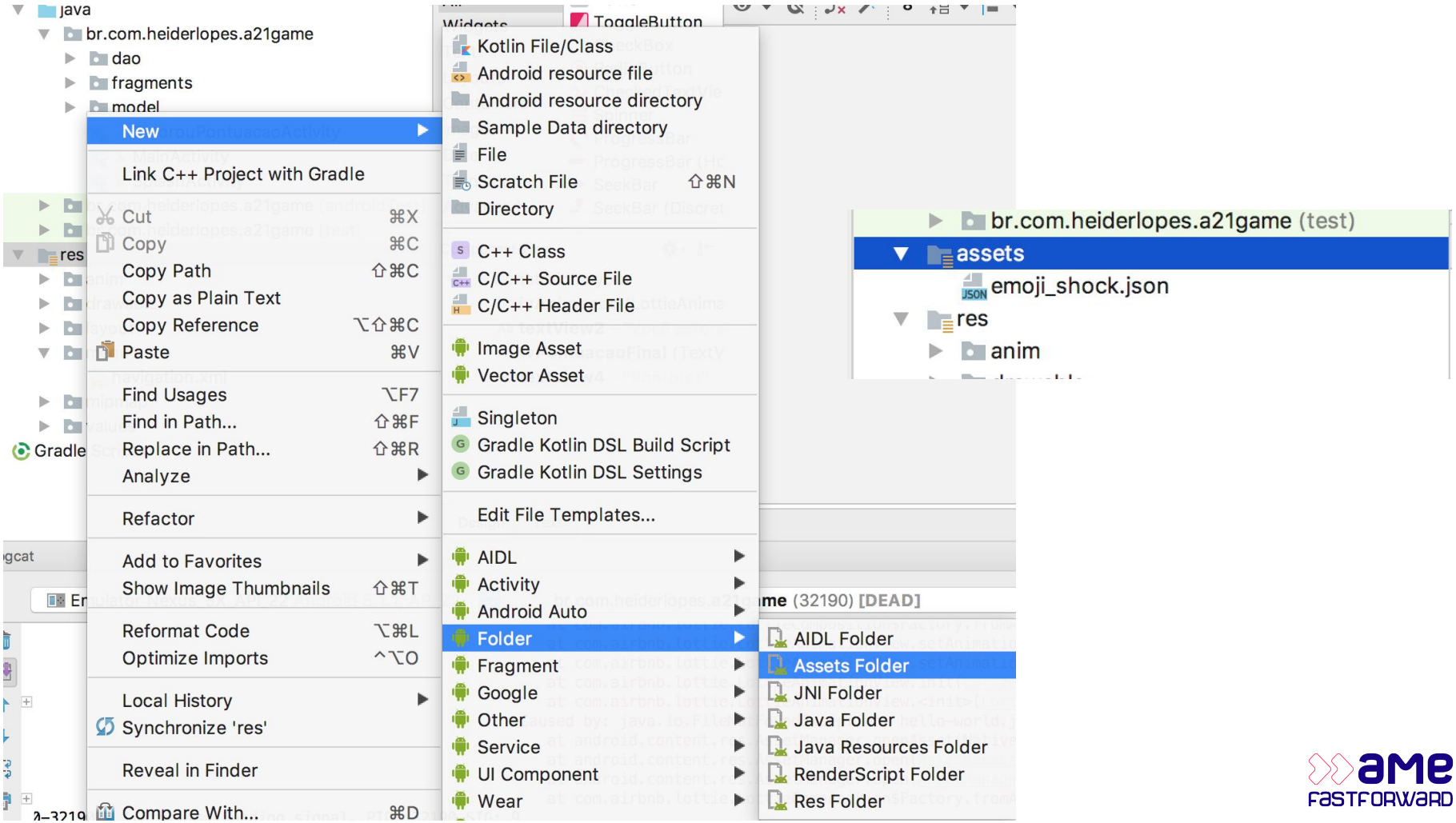
Saiba mais: **https://airbnb.design/lottie/**

https://www.lottiefiles.com

ame FASTFORWARD

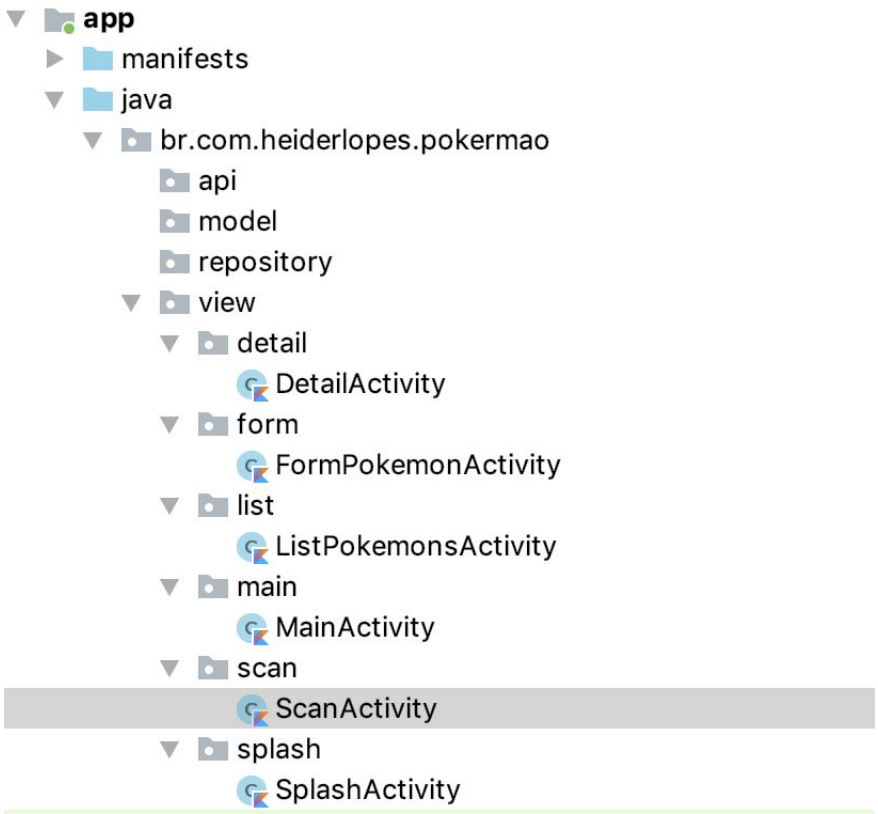
Conhecendo a Lottie - Adicionando no projeto

Criar a pasta **assets** e colar a animação baixada do site:
https://lottiefiles.com/4366-game-east-west



APP: Pokermao - UI

Crie as seguintes Activities (**Empty Activity**) com os seguintes nomes e dentro dos seus respectivos packages:



Os layouts e resources já estão disponíveis em:

adicionar a url com os recursos

APP: Pokermao - Tema FullScreen

Abra o arquivo **AndroidManifest.xml** e aplique o tema **FullScreen** no aplicativo":

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/FullScreen">
```

APP: Pokermao - Tema FullScreen

Ainda no arquivo **AndroidManifest.xml** altere para que a **SplashActivity** abra ao iniciar o aplicativo:

```
<activity android:name=".view.splash.SplashActivity">
  <intent-filter>
     <action android:name="android.intent.action.MAIN"/>

     <category android:name="android.intent.category.LAUNCHER"/>
  </intent-filter>
</activity>
```

Começando a integração com a API

APP: Pokermao - Retrofit - Adicionando a dependência

Abra o arquivo **build.gradle (app)** e adicione as seguintes dependências:

```
//Biblioteca para consumir webservice
implementation 'com.squareup.retrofit2:retrofit:2.6.0'

//Biblioteca para realizar o parse json para objeto/objeto para json
implementation 'com.squareup.retrofit2:converter-gson:2.6.0'

//Biblioteca para fazer debug das requests
implementation 'com.facebook.stetho:stetho:1.5.1'
implementation 'com.facebook.stetho:stetho-okhttp3:1.5.1'

//Biblioteca para auxiliar o carregamento de imagens no aplicativo
implementation 'com.squareup.picasso:picasso:2.71828'

//Biblioteca AAC
implementation "androidx.lifecycle:lifecycle-extensions:2.0.0"

//Injecao de dependencia
implementation "org.koin:koin-android-viewmodel:2.0.1"
implementation "org.koin:koin-android:2.0.1"
```

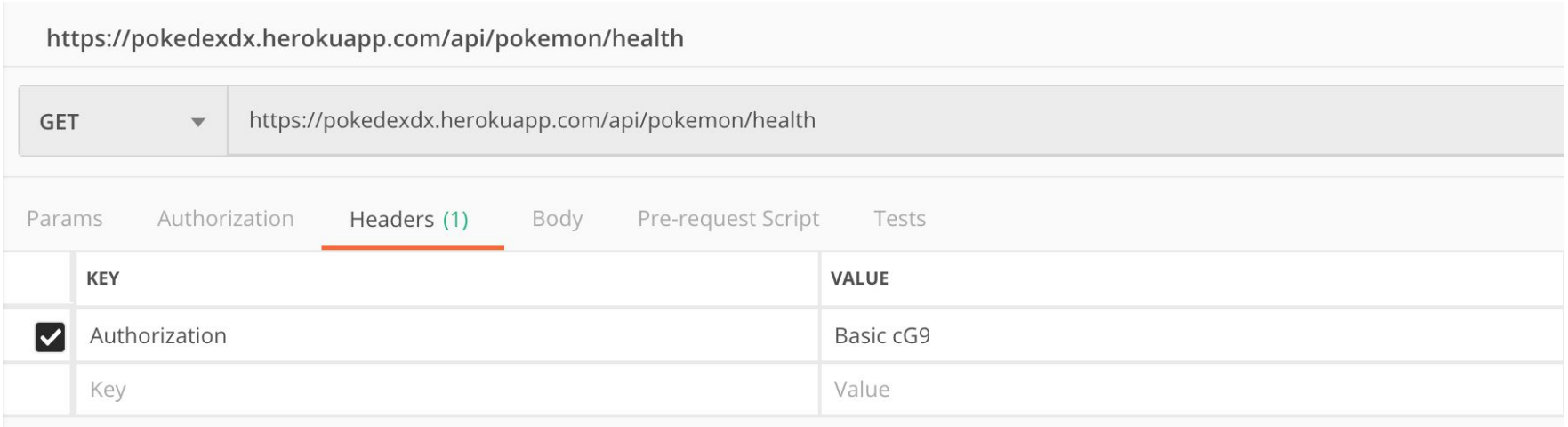
APP: Pokermao - Retrofit - Testando no Postman

O primeiro **Endpoint** a ser chamado será para verificar se o serviço está no ar.

A url para a chamada é a seguinte:

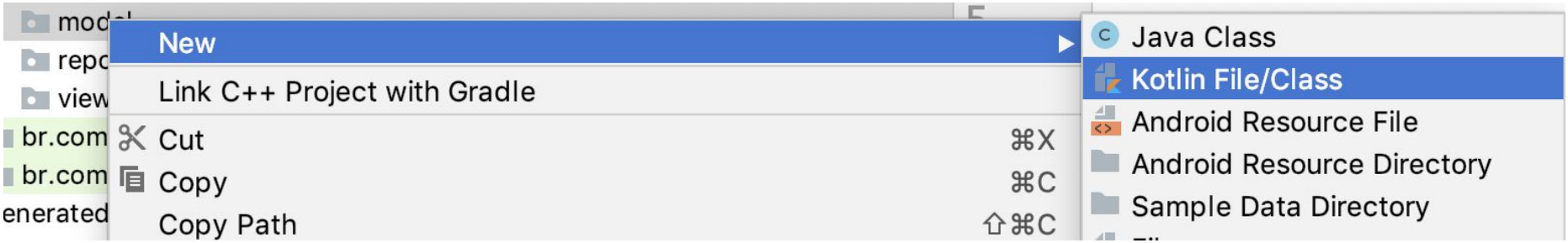
https://pokedexdx.herokuapp.com/api/pokemon/health

E precisamos informar no Header da request a chave do aplicativo:

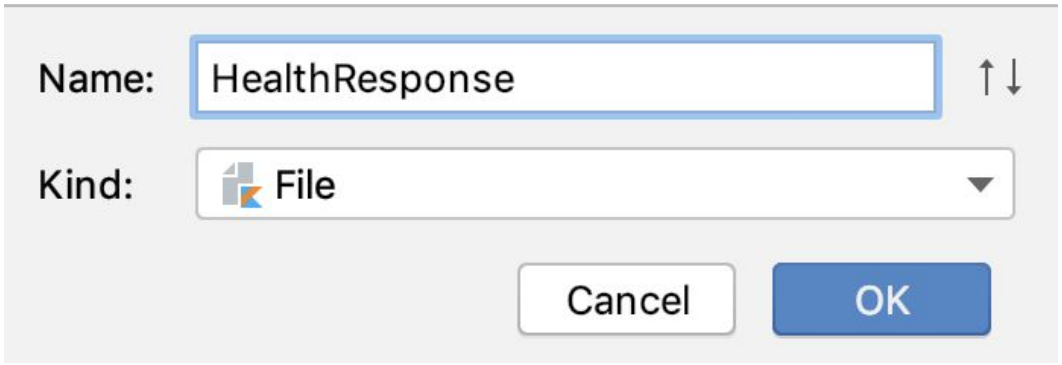


APP: Pokermao - Model

Clique com o botão direito sobre o package **model** em seguida: **New → Kotlin File/Class**



Chamaremos essa classe como **HealthResponse**



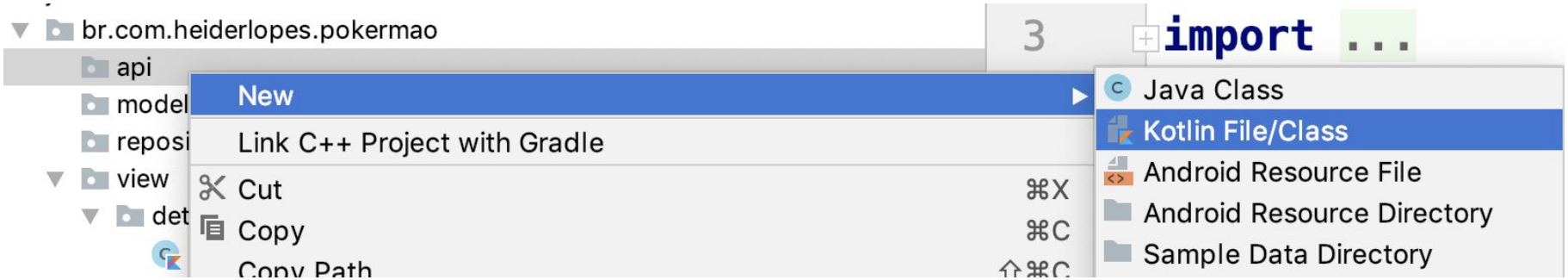
APP: Pokermao - Model

Segue o código da classe **HealthResponse**

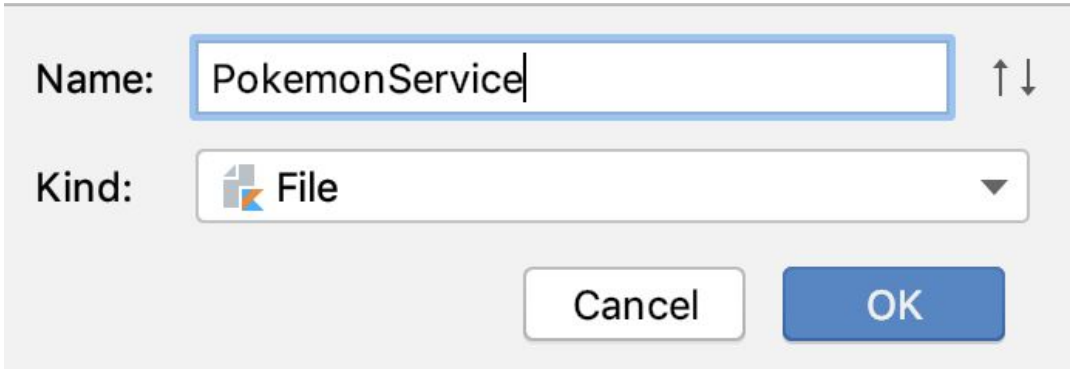
```
data class HealthResponse(
   val status: String
)
```

APP: Pokermao - Service

Clique com o botão direito sobre o package **api** em seguida: **New → Kotlin File/Class**



Chamaremos essa classe como **PokemonService**



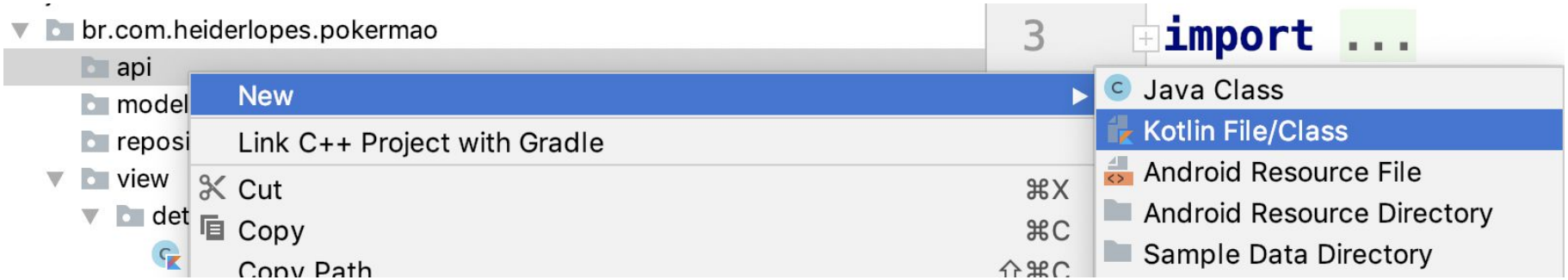
APP: Pokermao - Service

Segue o código referente a classe criada no slide anterior:

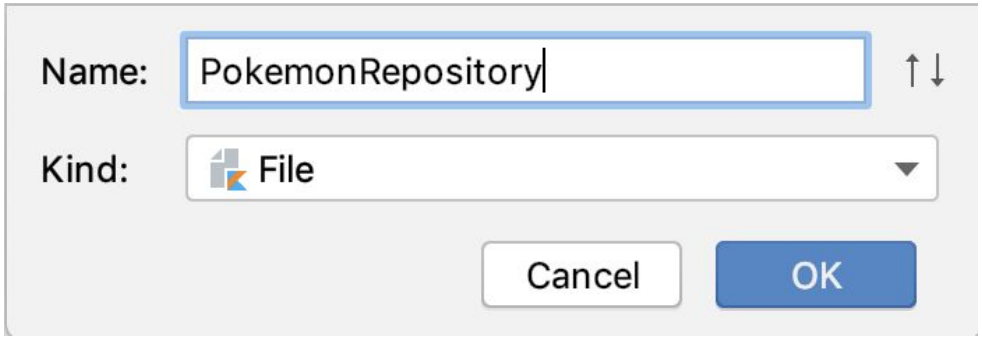
```
interface PokemonService {
   @GET("/api/pokemon/health")
   fun checkHealth(): Call<HealthResponse>
}
```

APP: Pokermao - Repository

Clique com o botão direito sobre o package **repository** em seguida: **New → Kotlin File/Class**



Chamaremos essa classe como **PokemonRepository**



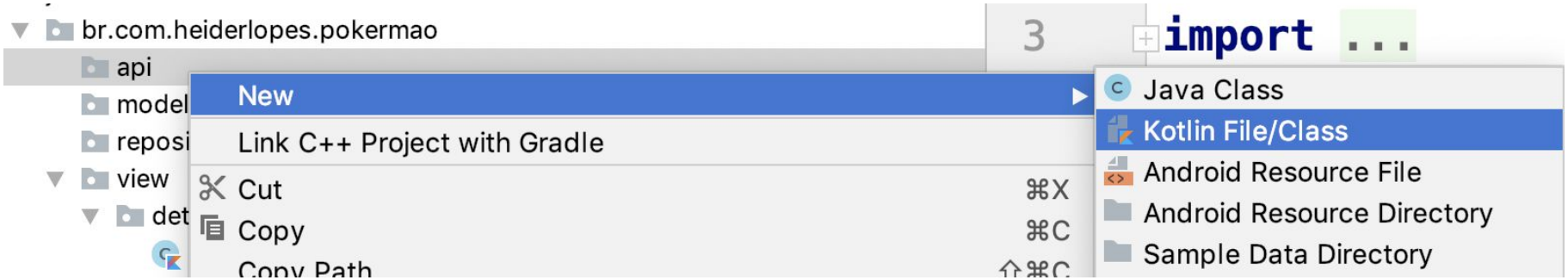
APP: Pokermao - Repository

Segue o código referente a classe criada no slide anterior:

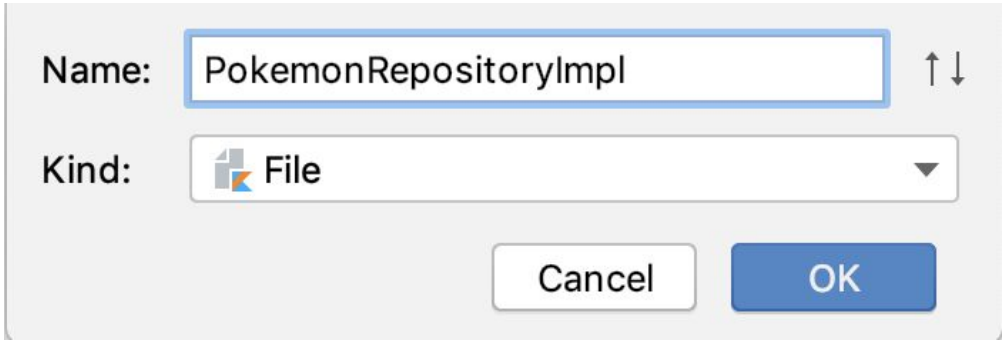
```
interface PokemonRepository {
    fun checkHealth(
        onComplete:() -> Unit,
        onError: (Throwable?) -> Unit
    )
}
```

APP: Pokermao - Repository

Clique com o botão direito sobre o package **repository** em seguida: **New → Kotlin File/Class**



Chamaremos essa classe como **PokemonRepositoryImpl**



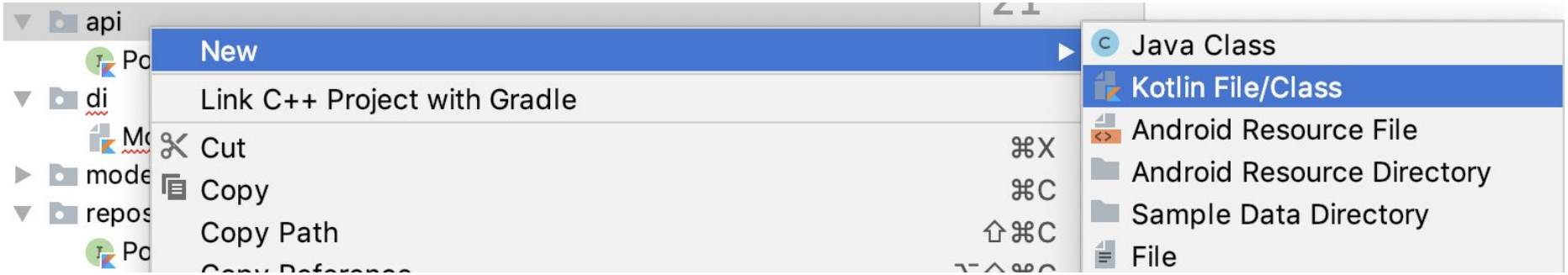
APP: Pokermao - Repository

Segue o código referente a classe criada no slide anterior:

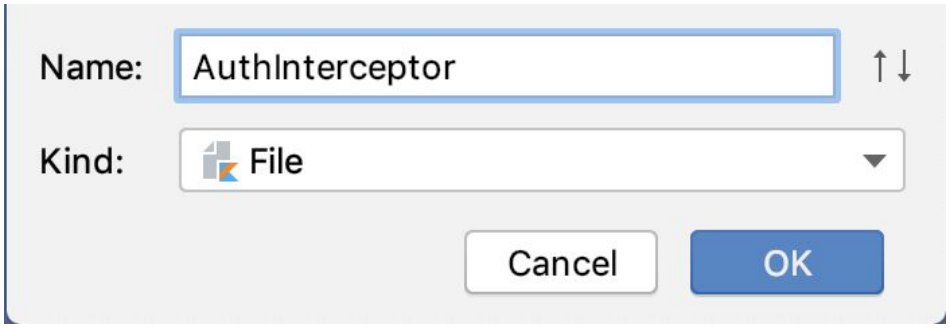
```
class PokemonRepositoryImpl(var pokemonService: PokemonService) :
PokemonRepository {

  override fun checkHealth(onComplete: () -> Unit, onError: (Throwable?) -> Unit) {
    pokemonService.checkHealth()
      .enqueue(object : Callback<HealthResponse> {
        override fun onFailure(call: Call<HealthResponse>, t: Throwable) {
          onError(t)
        }

        override fun onResponse(call: Call<HealthResponse>, response:
Response<HealthResponse>) {
          onComplete()
        }
      })
  }
}
```

APP: Pokermao - Interceptor

Clique com o botão direito sobre o package **api** em seguida: **New → Kotlin File/Class**



Chamaremos essa classe como **AuthInterceptor**



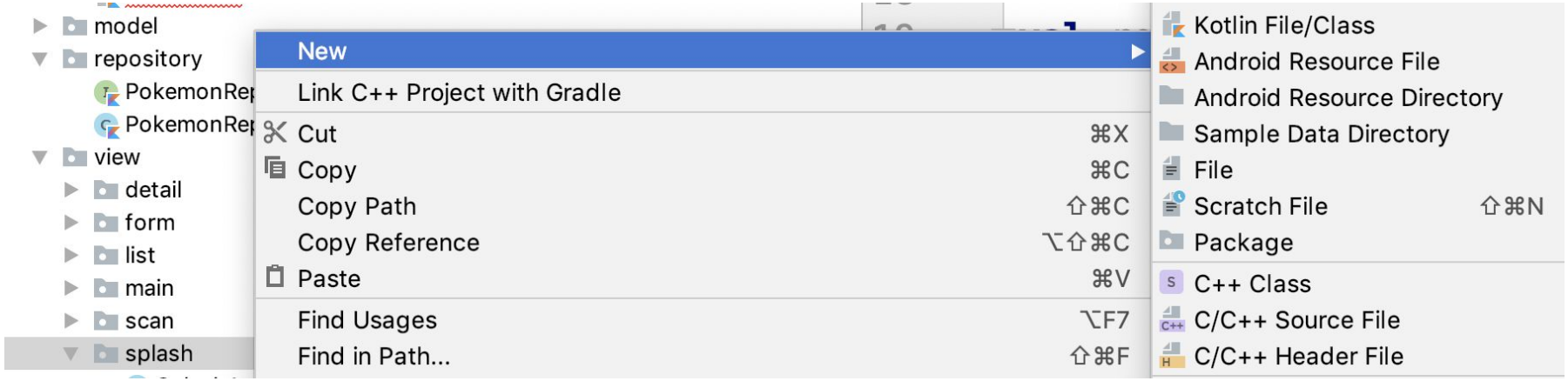
APP: Pokermao - Repository

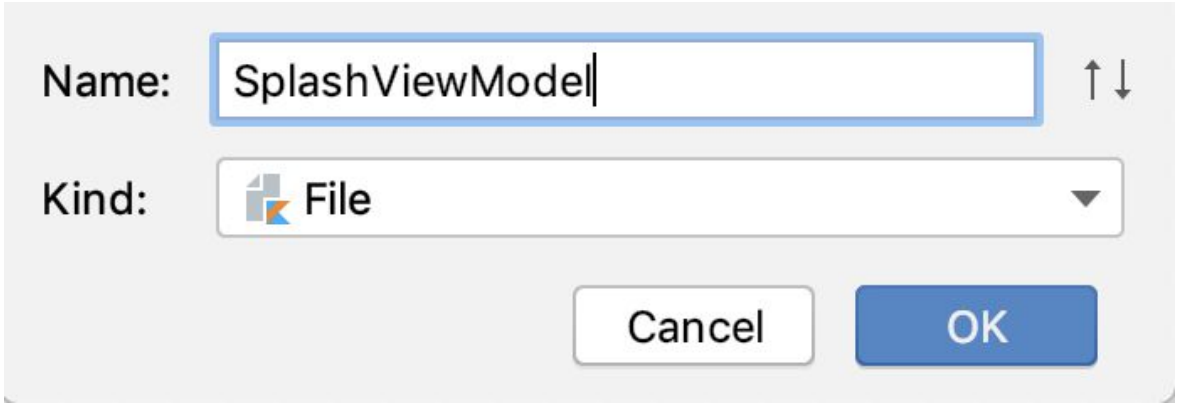
Segue o código referente a classe criada no slide anterior:

```
class AuthInterceptor : Interceptor {

   override fun intercept(chain: Interceptor.Chain?): Response {
      val requestBuilder = chain!!.request().newBuilder()
      requestBuilder.addHeader("Authorization", "Basic cG9rZWFwaTpwb2tlbW9u")
      val request = requestBuilder.build()
      val response = chain.proceed(request)
      if (response.code() == 401) {
         Log.e("MEUAPP", "Error API KEY")
      }
      return response
   }
}
```

APP: Pokermao - Repository

Crie uma nova classe dentro do package **splash** chamada de **SplashViewModel**





APP: Pokermao - Repository

O código da nossa **SplashViewModel**

```
class SplashViewModel(val pokemonRepository: PokemonRepository) : ViewModel() {

    val messageError: MutableLiveData<String> = MutableLiveData()

    fun checkHealth() {
        pokemonRepository.checkHealth(
            onComplete = {
                messageError.value = ""
            },
            onError = {
                messageError.value = it?.message
            }
        )
    }
}
```

Injeção de Dependência

APP: Pokermao - Dependência

Dependências são objetos que uma classe precisa para realizar os comportamentos esperados, portanto, se uma classe acessa o banco de dados e usa um DAO pra isso, o DAO é uma dependência da classe.

APP: Pokermao - Injeção de Dependência

É a técnica que delega a responsabilidade de inicializar dependências para o software.

Por exemplo, ao invés instanciarmos as dependências em algum momento do código, o próprio framework de injeção de dependência realiza esses passos pra gente.

O grande benefício é delegar a responsabilidade de inicialização das dependências, permitindo que membros do projeto apenas peçam o que precisam e a instância é fornecida automaticamente de acordo com o escopo necessário, como por exemplo, um Singleton ou Factory (instância sempre nova).

APP: Pokermao - Koin Módulos

A base de configuração do **Koin** é por meio de seus **módulos**, que são as entidades que mantém as **instruções de como as dependências devem ser inicializadas**.

Isso significa que a partir deles, ensinamos o Koin como ele deve injetar as dependências pra gente.

Para configurar é bem simples, basta apenas utilizar a função **module()**, que é uma **Higher-Order Function**, e definir a instância desejada a partir da expressão lambda.

Koin - Antes de iniciarmos o código é necessário saber

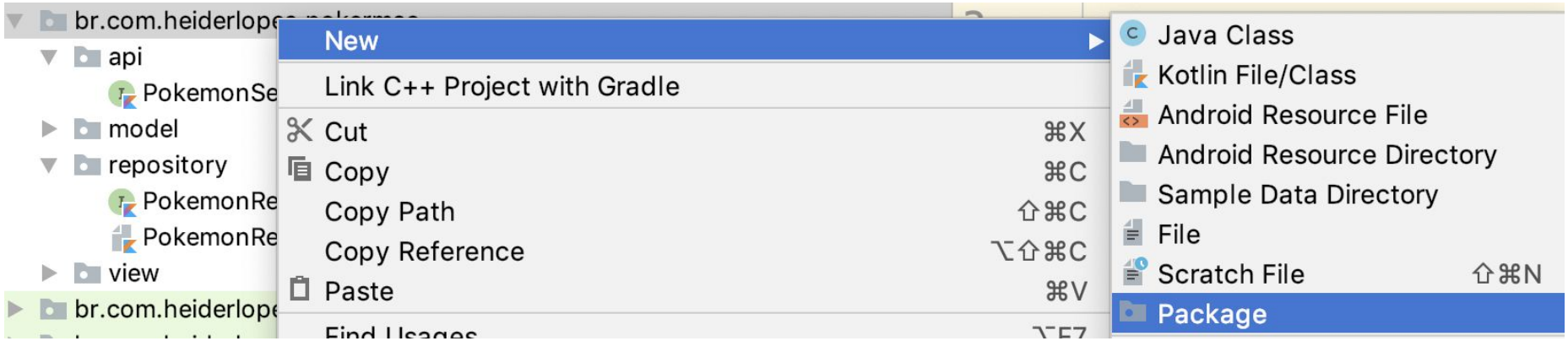
get(): é usado dentro de construtores para resolver instâncias necessárias.

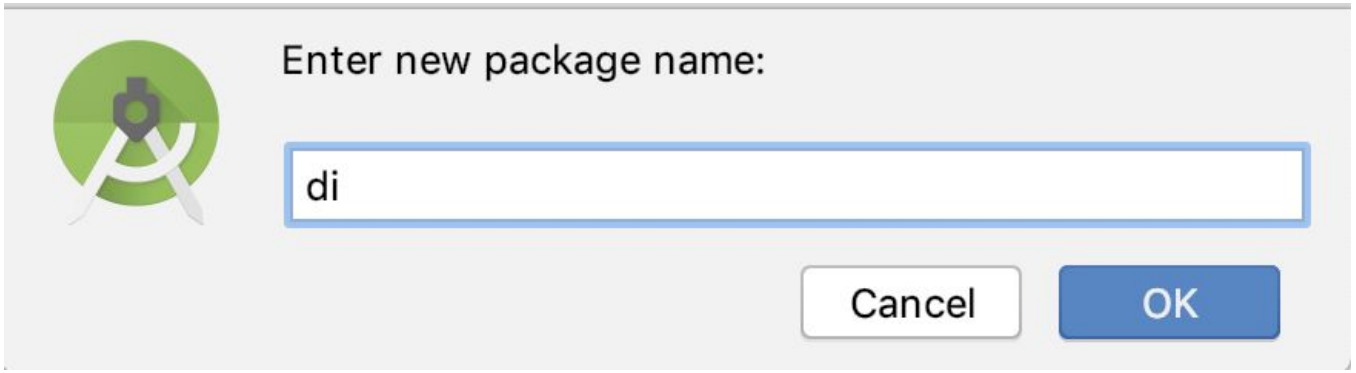
factory: é usado para indicar que uma nova instância deve ser criada sempre que for injetada.

single: indica que uma instância única será criada no início e compartilhada em todas as futuras injeções.

APP: Pokermao - Koin Módulos

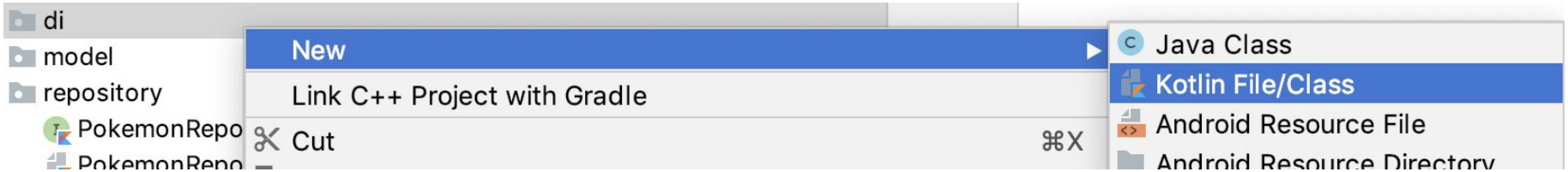
Crie um novo package chamado **di:**

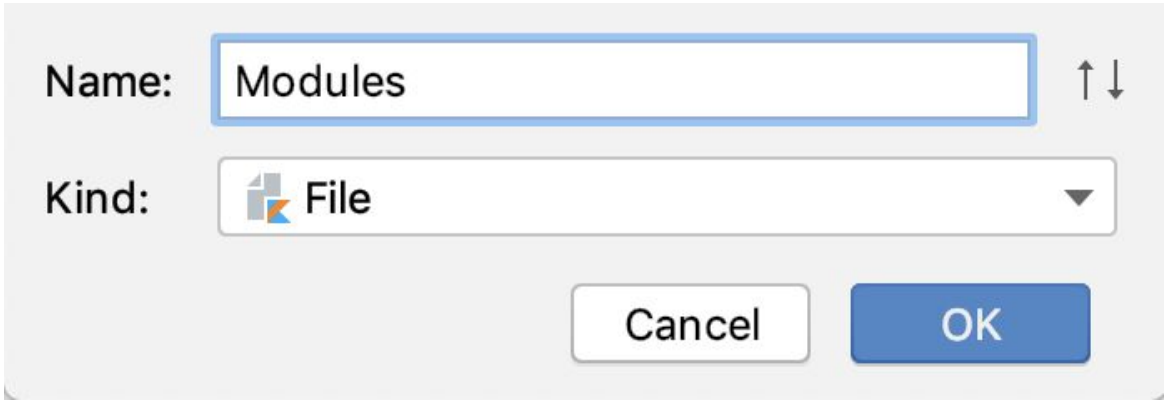




APP: Pokermao - Koin Módulos

Dentro do package **di** clique com o botão direito **New → Kotlin File/Class**





APP: Pokermao - Koin Módulos

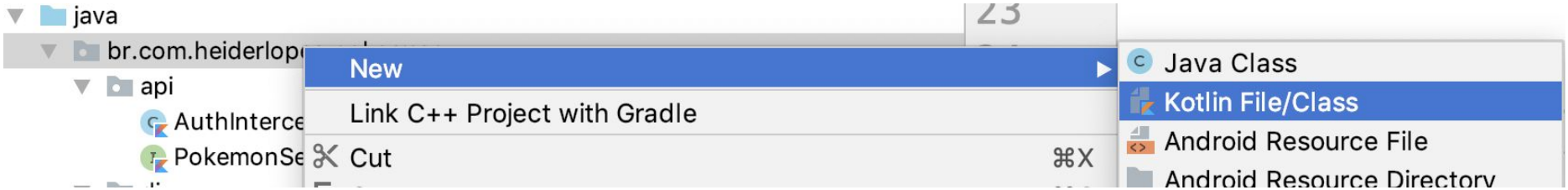
```
private fun createNetworkClient(okHttpClient: OkHttpClient): Retrofit {
    return Retrofit.Builder()
        .client(okHttpClient)
        .baseUrl("https://pokedexdx.herokuapp.com")
        .addConverterFactory(GsonConverterFactory.create())
        .build()
}

private fun createOkhttpClientAuth(authInterceptor: Interceptor): OkHttpClient {
    val builder = OkHttpClient.Builder()
        .addInterceptor(authInterceptor)
         .addNetworkInterceptor(StethoInterceptor())
        .connectTimeout(30, TimeUnit.SECONDS)
        .readTimeout(30, TimeUnit.SECONDS)
        .writeTimeout(30, TimeUnit.SECONDS)
    return builder.build()
}
```

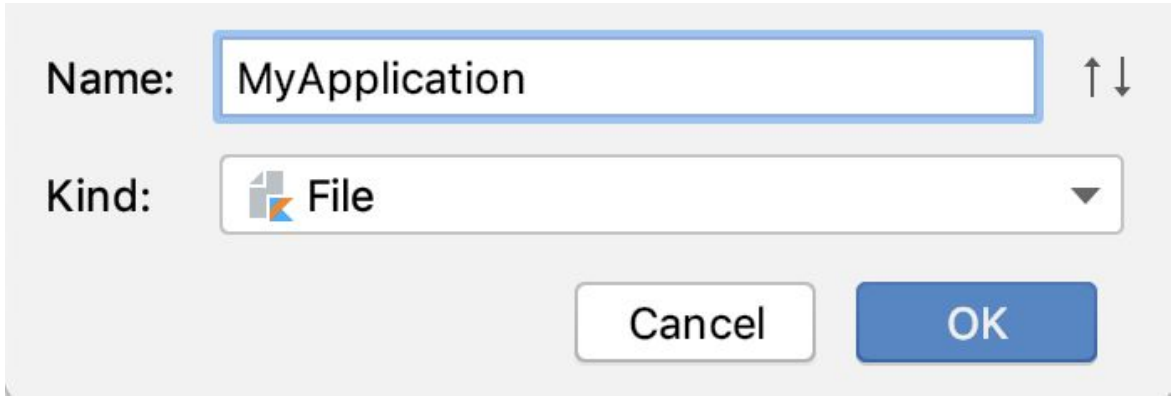
APP: Pokermao - Koin Módulos

```
val viewModelModule = module {
   viewModel { SplashViewModel(get()) }
}

val repositoryModule = module {
   single<PokemonRepository> { PokemonRepositoryImpl(get()) }
}

val networkModule = module {
   single<Interceptor> { AuthInterceptor() }
   single { createNetworkClient(get()).create(PokemonService::class.java) }
   single { createOkhttpClientAuth(get()) }

}
```

APP: Pokermao - Inicializando o Koin

Dentro do package **raiz** clique com o botão direito **New → Kotlin File/Class.**





APP: Pokermao - Inicializando o Koin

O código da **MyApplication.kt**

```
class MyApplication : Application(){
   override fun onCreate() {
      super.onCreate()
      // Start stetho
       Stetho.initializeWithDefaults(this)
      // Start Koin
      startKoin {
         androidLogger()
         androidContext(this@MyApplication)
         modules(
            listOf(
               viewModelModule,
               networkModule,
               repositoryModule
            )
         )
      }
   }
}
```

APP: Pokermao - Inicializando o Koin

Abra o arquivo **AndroidManifest.xml** e adicione a linha em negrito abaixo:

```
<application
    android:name=".MyApplication"
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/FullScreen">
```

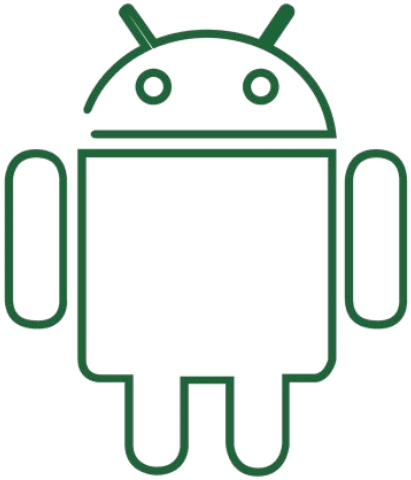
APP: Pokermao - Inicializando o Koin

Ainda no o arquivo **AndroidManifest.xml** e adicione a linha em negrito abaixo para avisar que iremos utilizar internet no nosso aplicativo.

```
<uses-permission android:name="android.permission.INTERNET"/>
```

APP: Pokermao - Programando nossa SplashActivity

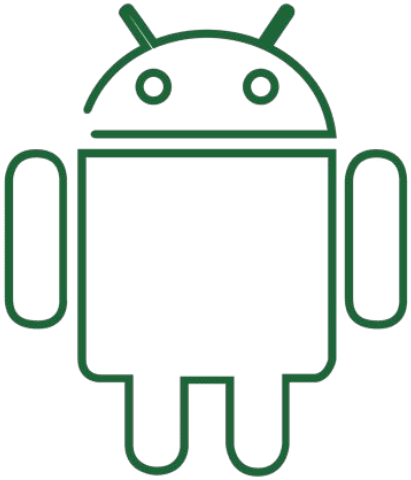
```
class SplashActivity : AppCompatActivity() {

    val splashViewModel: SplashViewModel by viewModel()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_splash)

        splashViewModel.checkHealth()

        splashViewModel.messageError.observe(this, Observer {
            if (it == "") {
                startActivity(Intent(this, MainActivity::class.java))
                finish()
            } else {
                Toast.makeText(this, it, Toast.LENGTH_LONG).show()
            }
        })
    }
}
```



Vamos programar a nossa MainActivity

APP: Pokermao - Programando nossa MainActivity

```
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        btPokedex.setOnClickListener {
            startActivity(Intent(this, ScanActivity::class.java))
        }

        btPokemonList.setOnClickListener {
            startActivity(Intent(this, ListPokemonsActivity::class.java))
        }

        btClose.setOnClickListener {
            finish()
        }
    }
}
```



Lista com os primeiros 150 Pokemons

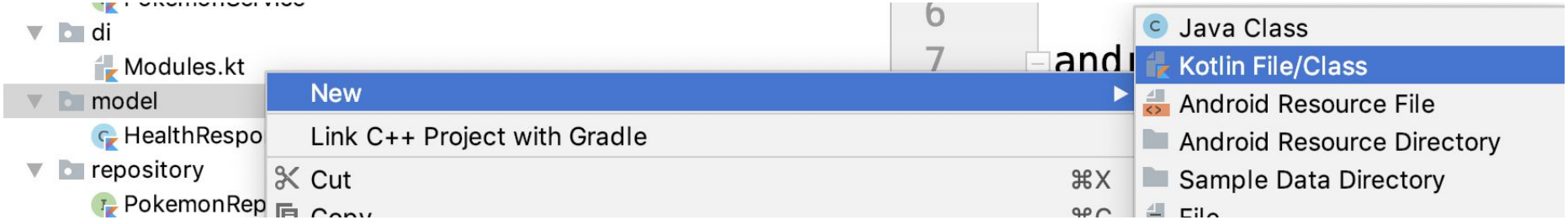
APP: Pokermao - Listando os 150 Pokemons

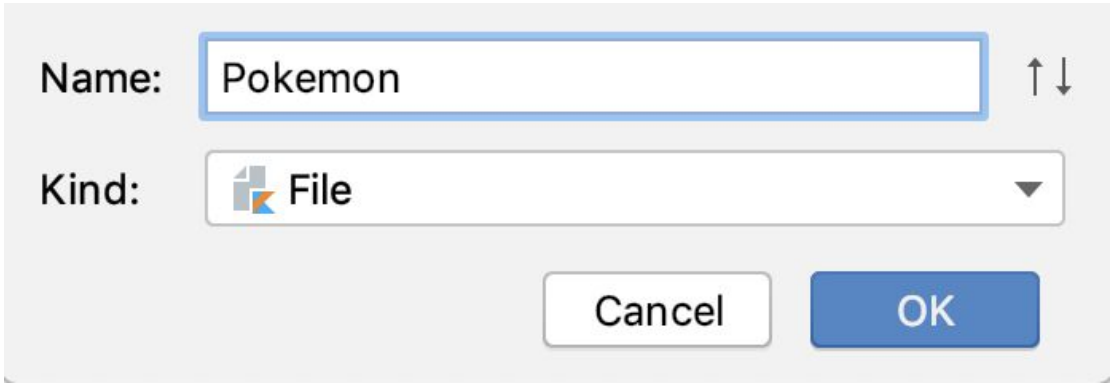
Abra o arquivo **build.gradle** e adicione a seguinte linha para que possamos utilizar a annotation **Parcelize**

```
androidExtensions {
   experimental = true
}
```

APP: Pokermao - Inicializando o Koin

Dentro do package **model** clique com o botão direito **New → Kotlin File/Class.**





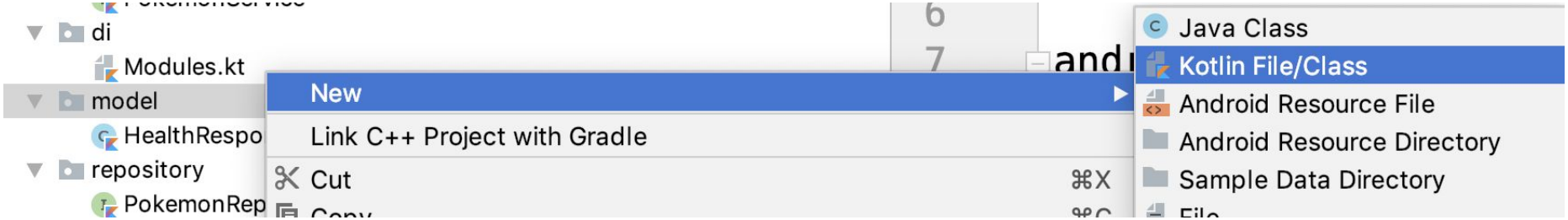
APP: Pokermao - Listando os 150 Pokemons

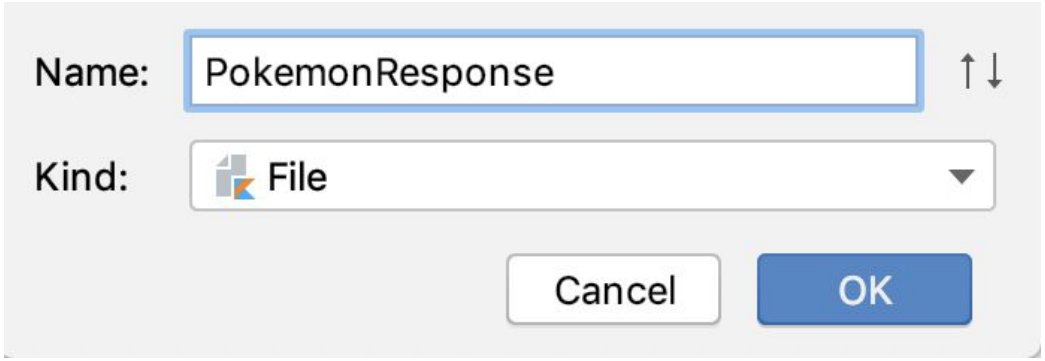
Nossa classe do **Pokemon**

```
data class Pokemon(
   @SerializedName("number") val numero: String,
   @SerializedName("name") val nome: String,
   @SerializedName("imageURL") val urlImagem: String
)
```

APP: Pokermao - Inicializando o Koin

Dentro do package **model** clique com o botão direito **New → Kotlin File/Class.**





APP: Pokermao - Listando os 150 Pokemons

Nossa classe do **PokemonResponse**

```
data class PokemonResponse(
  val content: List<Pokemon>
)
```

APP: Pokermao - Listando os 150 Pokemons

Abra o arquivo **PokemonService** e adicione o seguinte código:

```
@GET("/api/pokemon")
fun getPokemons(
    @Query("size") size: Int,
    @Query("sort") sort: String
): Call<PokemonResponse>
```

APP: Pokermao - Listando os 150 Pokemons

Abra o arquivo **PokemonRepository** e adicione o seguinte código:

```
fun getPokemons(
    size: Int,
    sort: String,
    onComplete: (List<Pokemon>?) -> Unit,
    onError: (Throwable?) -> Unit
)
```

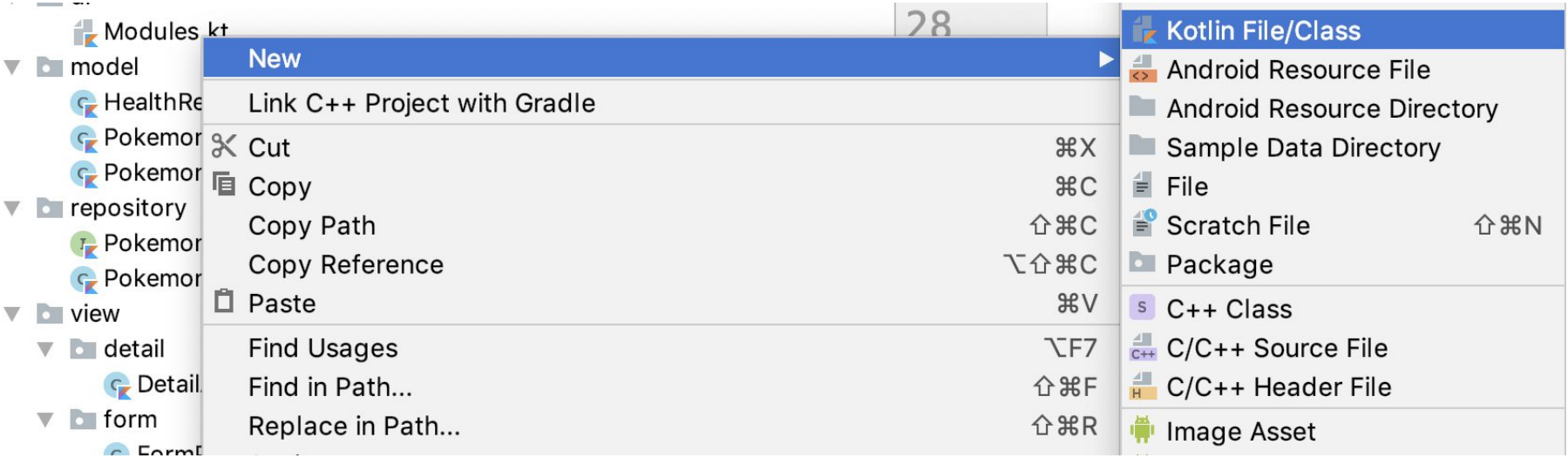
APP: Pokermao - Listando os 150 Pokemons

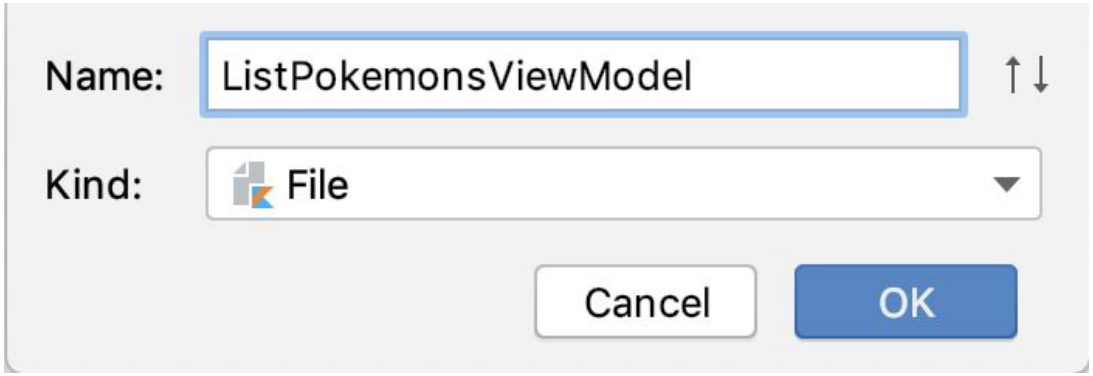
Abra o arquivo **PokemonRepositoryImpl** e adicione o seguinte código:

```
override fun getPokemons(
    size: Int,   sort: String,
    onComplete: (List<Pokemon>?) -> Unit,
    onError: (Throwable?) -> Unit
) {
    pokemonService.getPokemons(size, sort)
        .enqueue(object : Callback<PokemonResponse> {
            override fun onFailure(call: Call<PokemonResponse>, t: Throwable) {
                onError(t)
            }
            override fun onResponse(call: Call<PokemonResponse>, response:
Response<PokemonResponse>) {
                if (response.isSuccessful) {
                    onComplete(response.body()?.content)
                } else {
                    onError(Throwable("Não foi possível carregar os Pokémons"))
                }
            }
        })
}
```

APP: Pokermao - Listando os 150 Pokemons

Crie uma nova classe chamada **ListPokemonsViewModel** dentro do package **view.list**





APP: Pokermao - Listando os 150 Pokemons

```
class ListPokemonsActivity : AppCompatActivity() {
    val listaPokemonsViewModel: ListPokemonsViewModel by viewModel()
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_list_pokemons)
        listaPokemonsViewModel.getPokemons()
        listaPokemonsViewModel.isLoading.observe(this, Observer {
            if(it == true) {
                containerLoading.visibility = View.VISIBLE
            } else {
                containerLoading.visibility = View.GONE
            }
        })
        listaPokemonsViewModel.messageError.observe(this, Observer {
            if(it != "") {
                Toast.makeText(this, it, Toast.LENGTH_LONG).show()
            }
        })
        listaPokemonsViewModel.pokemons.observe(this, Observer {
            Log.i("Pokemons", it[0].nome)
        })
    }
}
```

APP: Pokermao - Criando o Adapter

Dentro do package **view.list** crie uma classe Kotlin com o nome **ListPokemonsAdapter**

```
class ListPokemonsAdapter(
   val pokemons: List<Pokemon>,
   val picasso: Picasso,
   val clickListener: (Pokemon) -> Unit
) : RecyclerView.Adapter<ListPokemonsAdapter.PokemonViewHolder>() {

   override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
PokemonViewHolder {
      val view = LayoutInflater.from(parent.context).inflate(R.layout.pokemon_list_item,
parent, false)
      return PokemonViewHolder(view)
   }
   override fun getItemCount(): Int {
      return pokemons.size
   }
```

APP: Pokermao - Criando o Adapter

```
override fun onBindViewHolder(holder: PokemonViewHolder, position: Int) {
        val pokemon = pokemons[position]
        holder.bindView(pokemon, picasso, clickListener)
    }

    class PokemonViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {

        fun bindView(pokemon: Pokemon,
                     picasso: Picasso,
                     clickListener: (Pokemon) -> Unit) = with(itemView) {
            tvPokemonName.text = pokemon.name
            tvPokemonNumber.text = pokemon.number

picasso.load("https://pokedexdx.herokuapp.com${pokemon.imageURL}").into(ivPoke
mon)

            setOnClickListener { clickListener(pokemon) }
        }
    }
}
```

APP: Pokermao - Criando o Picasso

Abra o arquivo **Modules.kt** e adicione as seguintes linhas:

```
val networkModule = module {
  single { createPicassoAuth(get(), get()) }
}
```

```
private fun createPicassoAuth(context: Context, okHttpClient: OkHttpClient): Picasso {
  return Picasso
      .Builder(context)
      .downloader(OkHttp3Downloader(okHttpClient))
      .build()
}
```

APP: Pokermao - Exibindo os dados na lista

Vamos alterar nossa **ListPokemonsActivity.kt** para exibir os dados

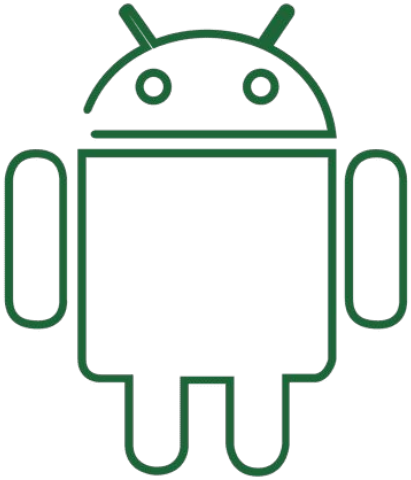
```
class ListPokemonsActivity : AppCompatActivity() {

    val listPokemonsViewModel: ListPokemonsViewModel by viewModel()
    val picasso: Picasso by inject()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_list_pokemons)

        listPokemonsViewModel.getPokemons()

        listPokemonsViewModel.isLoading.observe(this, Observer {
            if(it == true) {
                containerLoading.visibility = View.VISIBLE
            } else {
                containerLoading.visibility = View.GONE
            }
        })
```

APP: Pokermao - Exibindo os dados na lista

```
listPokemonsViewModel.messageError.observe(this, Observer {
        if(it != "") {
            Toast.makeText(this, it, Toast.LENGTH_LONG).show()
        }
    })

    listPokemonsViewModel.pokemons.observe(this, Observer {
        rvPokemons.adapter = ListPokemonsAdapter(it, picasso) {
            val intent = Intent(this, FormPokemonActivity::class.java)
            intent.putExtra("POKEMON", it)
            startActivity(intent)
            finish()
        }
        rvPokemons.layoutManager = GridLayoutManager(this, 3)
    })

  }
}
```

APP: Pokermao - Exibindo os dados na lista

```
listPokemonsViewModel.messageError.observe(this, Observer {
        if(it != "") {
            Toast.makeText(this, it, Toast.LENGTH_LONG).show()
        }
    })

    listPokemonsViewModel.pokemons.observe(this, Observer {
        rvPokemons.adapter = ListPokemonsAdapter(it, picasso) {
            val intent = Intent(this, FormPokemonActivity::class.java)
            intent.putExtra("POKEMON", it)
            startActivity(intent)
            finish()
        }
        rvPokemons.layoutManager = GridLayoutManager(this, 3)
    })

  }
}
```



Alterando os dados do Pokémon selecionado

APP: Pokermao - Alterando os dados do Pokémon

Abra o arquivo **PokemonService** e adicione o seguinte código:

```
@PUT("/api/pokemon")
fun updatePokemon(
   @Body pokemon: Pokemon
) : Call<Pokemon>
```

APP: Pokermao - Alterando os dados do Pokémon

Abra o arquivo **PokemonRepository** e adicione o seguinte código:

```
fun updatePokemon(
   pokemon: Pokemon,
   onComplete:(Pokemon?) -> Unit,
   onError:(Throwable) -> Unit
)
```

APP: Pokermao - Alterando os dados do Pokémon

Abra o arquivo **PokemonRepositoryImpl** e adicione o seguinte código:

```
override fun updatePokemon(pokemon: Pokemon, onComplete: (Pokemon?) -> Unit,
onError: (Throwable) -> Unit) {
   pokemonService
       .updatePokemon(pokemon)
       .enqueue(object : Callback<Pokemon>{
           override fun onFailure(call: Call<Pokemon>, t: Throwable) {
               onError(t)
           }
           override fun onResponse(call: Call<Pokemon>, response: Response<Pokemon>) {
               if(response.isSuccessful) {
                   onComplete(response.body())
               } else {
                   onError(Throwable("Não foi possível realizar a requisição"))
               }
           }
       })
}
```

APP: Pokermao - Alterando os dados do Pokémon

Dentro do package **view.form** crie uma classe Kotlin chamada **FormPokemonViewModel.kt** e adicione o seguinte código:

APP: Pokermao - Alterando os dados do Pokémon

```
class FormPokemonViewModel(
    val pokemonRepository: PokemonRepository
) : ViewModel() {
    val isLoading = MutableLiveData<Boolean>()
    val messageResponse = MutableLiveData<String>()

    fun updatePokemon(pokemon: Pokemon) {
        isLoading.value = true
        pokemonRepository.updatePokemon(
            pokemon = pokemon,
            onComplete = {
                isLoading.value = false
                messageResponse.value = "Dados atualizados com sucesso"
            },
            onError = {
                isLoading.value = false
                messageResponse.value = it.message
            }
        )
    }
}
```

APP: Pokermao - Alterando os dados do Pokémon

Abra o arquivo **Modules.kt** e adicione o seguinte código em negrito:

```
val viewModelModule = module {
   viewModel { SplashViewModel(get()) }
   viewModel { ListPokemonsViewModel(get()) }
   viewModel { FormPokemonViewModel(get()) }
}
```

APP: Pokermao - Alterando os dados do Pokémon

Abra o arquivo FormPokemonActivity.kt e adicione o seguinte código:

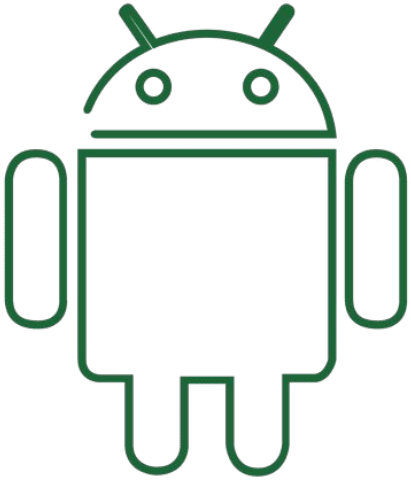
```
class FormPokemonActivity : AppCompatActivity() {
  val formPokemonViewModel: FormPokemonViewModel by viewModel()
  val picasso: Picasso by inject()
  lateinit var pokemon : Pokemon
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_form_pokemon)
    setValues()
    formPokemonViewModel.messageResponse.observe(this, Observer {
      Toast.makeText(this, it, Toast.LENGTH_LONG).show()
    })

    btSaveForm.setOnClickListener {
      pokemon.attack = sbAttack.progress
      pokemon.defense = sbDefense.progress
      pokemon.velocity = sbVelocity.progress
      pokemon.ps = sbPS.progress

      formPokemonViewModel.updatePokemon(pokemon)
    }
  }
```

APP: Pokermao - Alterando os dados do Pokémon

```
private fun setValues() {
    pokemon = intent.getParcelableExtra<Pokemon>("POKEMON")
    tvPokemonNameForm.text = pokemon.name


picasso.load("https://pokedexdx.herokuapp.com${pokemon.imageURL}").into(ivPokemonForm)

    sbAttack.progress = pokemon.attack
    sbDefense.progress = pokemon.defense
    sbPS.progress = pokemon.ps
    sbVelocity.progress = pokemon.velocity

    tvAttackValue.text = pokemon.attack.toString()
    tvDefenseValue.text = pokemon.defense.toString()
    tvPSValue.text = pokemon.ps.toString()
    tvVelocityValue.text = pokemon.velocity.toString()

    setListener(sbAttack, tvAttackValue)
    setListener(sbDefense, tvDefenseValue)
    setListener(sbVelocity, tvVelocityValue)
    setListener(sbPS, tvPSValue)
}
```

APP: Pokermao - Alterando os dados do Pokémon

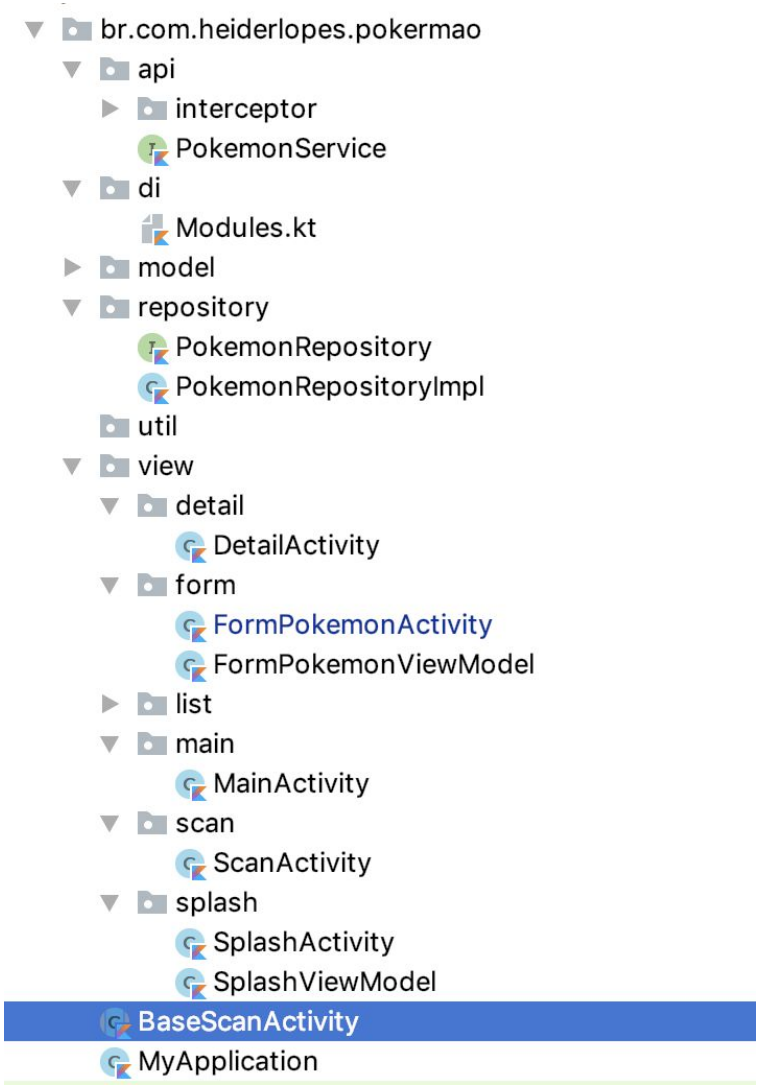
```
private fun setListener(seekBar: SeekBar, textView: TextView) {
    seekBar.setOnSeekBarChangeListener(object :
SeekBar.OnSeekBarChangeListener {
        override fun onProgressChanged(seekBar: SeekBar?, progress: Int, fromUser:
Boolean) {
            textView.text = progress.toString()
        }

        override fun onStartTrackingTouch(seekBar: SeekBar?) {}

        override fun onStopTrackingTouch(seekBar: SeekBar?) {}
    })
  }
}
```



Lendo o QRCode

APP: Pokermao - Lendo o QRCode

Crie uma classe Kotlin chamada **BaseScanActivity** na raíz do projeto e adicione o seguinte código:



APP: Pokermao - Lendo o QRCode

```
abstract class BaseScanActivity : AppCompatActivity() {

    private val cameraResult = 101
    abstract val baseScannerView: ZXingScannerView?

    abstract fun onPermissionDenied()
    abstract fun onPermissionGranted()

    @RequiresApi(Build.VERSION_CODES.M)
    fun requestPermission() {
        if (ContextCompat.checkSelfPermission(this, Manifest.permission.CAMERA) ==
PackageManager.PERMISSION_GRANTED) {
            onPermissionGranted()
        } else {
            if (shouldShowRequestPermissionRationale(Manifest.permission.CAMERA)) {
                onPermissionDenied()
            }
            requestPermissions(arrayOf(Manifest.permission.CAMERA), cameraResult)
        }
    }
```

APP: Pokermao - Lendo o QRCode

```
public override fun onPause() {
        super.onPause()
        baseScannerView?.stopCamera()
    }

    override fun onRequestPermissionsResult(requestCode: Int, permissions:
Array<String>, grantResults: IntArray) {
        if (requestCode == cameraResult) {
            if (grantResults.isNotEmpty() && grantResults[0] ==
PackageManager.PERMISSION_GRANTED) {
                onPermissionGranted()
            } else {
                onPermissionDenied()
            }
        } else {
            super.onRequestPermissionsResult(requestCode, permissions, grantResults)
        }
    }
}
```

APP: Pokermao - Lendo o QRCode

Abra o arquivo **ScanActivity.kt** e adicione o seguinte código:

```
class ScanActivity : BaseScanActivity(), ZXingScannerView.ResultHandler {
  override val baseScannerView: ZXingScannerView?
    get() = mScannerView

  @RequiresApi(Build.VERSION_CODES.M)
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_scan)
    btPermission.setOnClickListener {
      val intent = Intent(
        Settings.ACTION_APPLICATION_DETAILS_SETTINGS,
        Uri.parse("package:$packageName")
      )
      intent.addCategory(Intent.CATEGORY_DEFAULT)
      startActivity(intent)
    }

    super.requestPermission()
  }
```

APP: Pokermao - Lendo o QRCode

```
public override fun onResume() {
   super.onResume()
   if (ContextCompat.checkSelfPermission(this, Manifest.permission.CAMERA) ==
PackageManager.PERMISSION_GRANTED) {
       containerPermission.visibility = View.GONE
       mScannerView.setResultHandler(this)
       mScannerView.startCamera()
   } else {
       containerPermission.visibility = View.VISIBLE
   }
}

override fun onPermissionDenied() {
   containerPermission.visibility = View.VISIBLE
}

override fun onPermissionGranted() {
   containerPermission.visibility = View.GONE
}
```

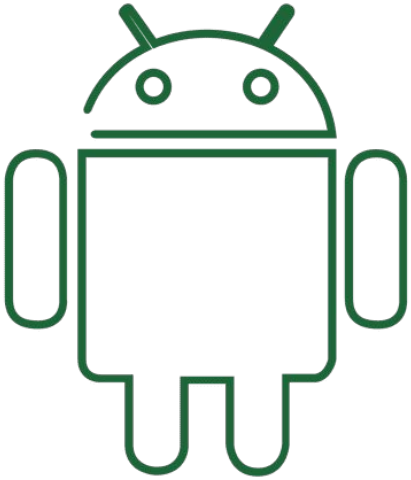
APP: Pokermao - Lendo o QRCode

```
override fun handleResult(rawResult: Result?) {
        val pokemonNumber = rawResult?.text
        val intent = Intent(this, DetailActivity::class.java)
        intent.putExtra("POKEMON_NUMBER", pokemonNumber)
        startActivity(intent)
        finish()
        //baseScannerView?.resumeCameraPreview(this)
    }
}
```

APP: Pokermao - Lendo o QRCode

Abra o arquivo **AndroidManifest.xml** e adicione as linhas abaixo para utilização da câmera:

```
<uses-permission android:name="android.permission.CAMERA" />

<uses-feature
    android:name="android.hardware.camera"
    android:required="true" />
<uses-feature
    android:name="android.hardware.camera.autofocus"
    android:required="false" />
```



Pesquisando Pokémon pelo seu número

APP: Pokermao - Pesquisando Pokémon pelo número

Abra o arquivo **PokemonService** e adicione o seguinte código:

```
@GET("/api/pokemon/{number}")
fun getPokemon(
   @Path("number") number: String
) : Call<Pokemon>
```

APP: Pokermao - Pesquisando Pokémon pelo número

Abra o arquivo **PokemonRepository** e adicione o seguinte código:

```
fun getPokemon(
    number: String,
    onComplete:(Pokemon?) -> Unit,
    onError:(Throwable) -> Unit
)
```

APP: Pokermao - Pesquisando Pokémon pelo número

Abra o arquivo **PokemonRepositoryImpl** e adicione o seguinte código:

```
override fun getPokemon(number: String, onComplete: (Pokemon?) -> Unit, onError:
(Throwable) -> Unit) {
   pokemonService
       .getPokemon(number)
       .enqueue(object : Callback<Pokemon>{
          override fun onFailure(call: Call<Pokemon>, t: Throwable) {
              onError(t)
          }
         override fun onResponse(call: Call<Pokemon>, response: Response<Pokemon>) {
              if(response.isSuccessful) {
                  onComplete(response.body())
              } else {
                  onError(Throwable("Não foi possível realizar a requisição"))
              }
          }
       })
}
```

APP: Pokermao - Pesquisando Pokémon pelo número

Dentro do package **view.detail** crie uma classe Kotlin chamada **DetailViewModel.kt** e adicione o seguinte código:

APP: Pokermao - Pesquisando Pokémon pelo número

```
class DetailViewModel(
    val pokemonRepository: PokemonRepository
) : ViewModel() {

    val isLoading = MutableLiveData<Boolean>()
    val pokemon = MutableLiveData<Pokemon>()
    fun getPokemon(number: String) {
        isLoading.value = true
        pokemonRepository.getPokemon(
            number,
            onComplete = {
                isLoading.value = false
                pokemon.value = it
            },
            onError = {
                isLoading.value = false
            }
        )
    }
}
```

APP: Pokermao - Pesquisando Pokémon pelo número

Abra o arquivo **Modules.kt** e adicione a seguinte linha em negrito:

```
val viewModelModule = module {
   viewModel { DetailViewModel(get()) }
}
```

APP: Pokermao - Pesquisando Pokémon pelo número

Abra o arquivo **DetailActivity.kt** e adicione a seguinte linha em negrito:

```
class DetailActivity : AppCompatActivity() {

    val detailViewModel: DetailViewModel by viewModel()
    val picasso: Picasso by inject()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_detail)

        detailViewModel.getPokemon(intent.getStringExtra("POKEMON_NUMBER"))

        detailViewModel.pokemon.observe(this, Observer {
            picasso.load("https://pokedexdx.herokuapp.com${it.imageURL}").into(ivPokemon)
            tvPokemonName.text = "${it.number} ${it.name}"

        })
```

APP: Pokermao - Pesquisando Pokémon pelo número

```
        detailViewModel.messageError.observe(this, Observer {
            if(it != "")
                Toast.makeText(this, it, Toast.LENGTH_LONG).show()
        })

        detailViewModel.isLoading.observe(this, Observer{
            if(it == true) {
                containerLoading.visibility = View.VISIBLE
            } else {
                containerLoading.visibility = View.GONE
            }
        })
    }
}
```