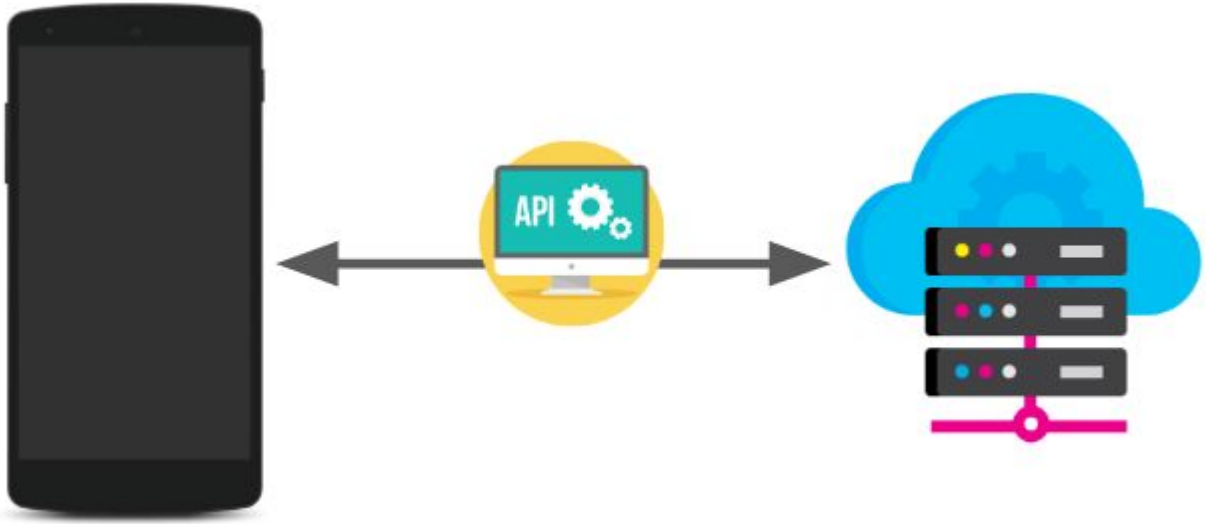

FIAP
MBA

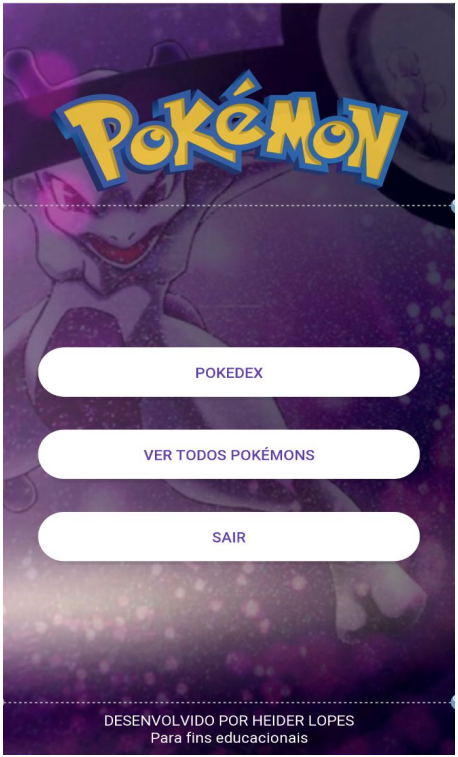
MBA em Full Stack Development - Design, Engineering & Deployment

MOBILE DEVELOPMENT

O que veremos: Consumindo WebServices

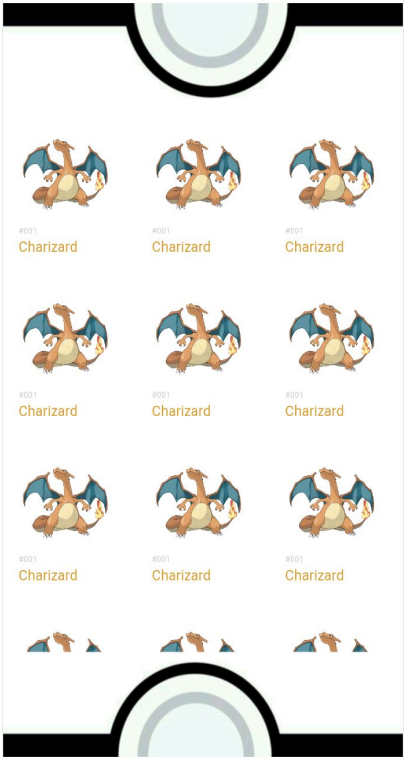


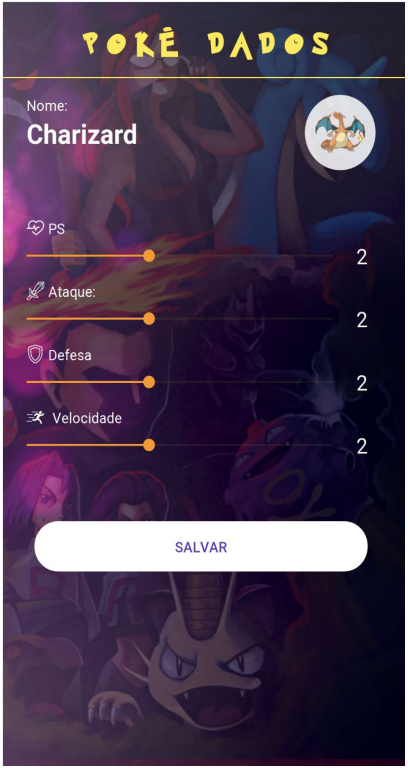
Conhecendo o projeto



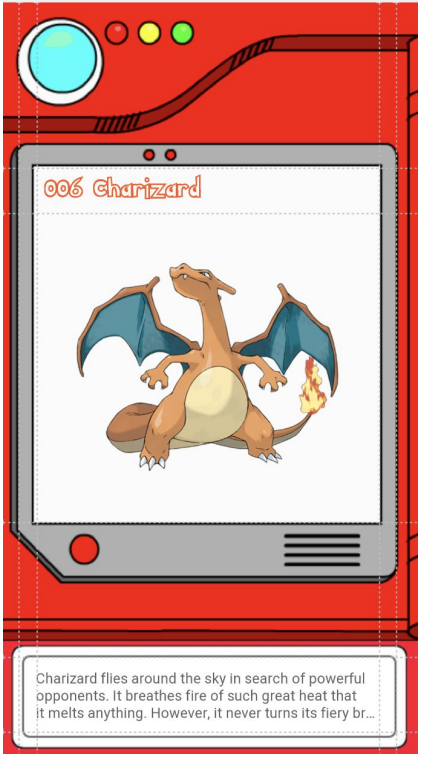
Conhecendo o projeto

Carregando os dados





Conhecendo o projeto



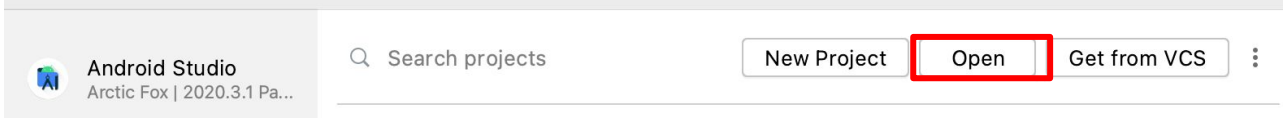
ABRINDO O PROJETO

BAIXANDO OS ARQUIVOS

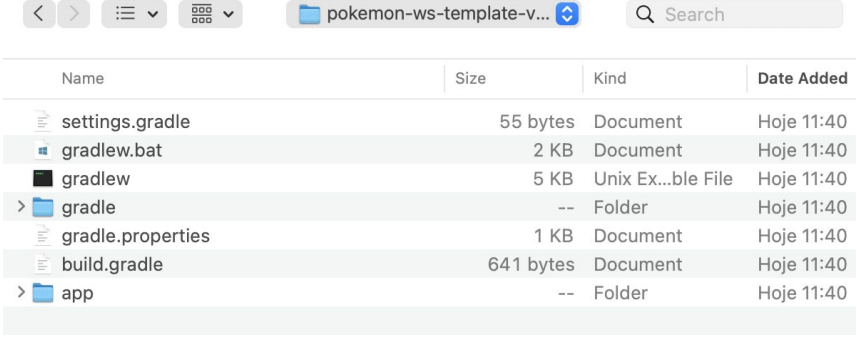
Baixe o projeto no repositório:

https://github.com/heiderlopes/pokemon-ws-template-v2

Clique sobre o Botão **Open**

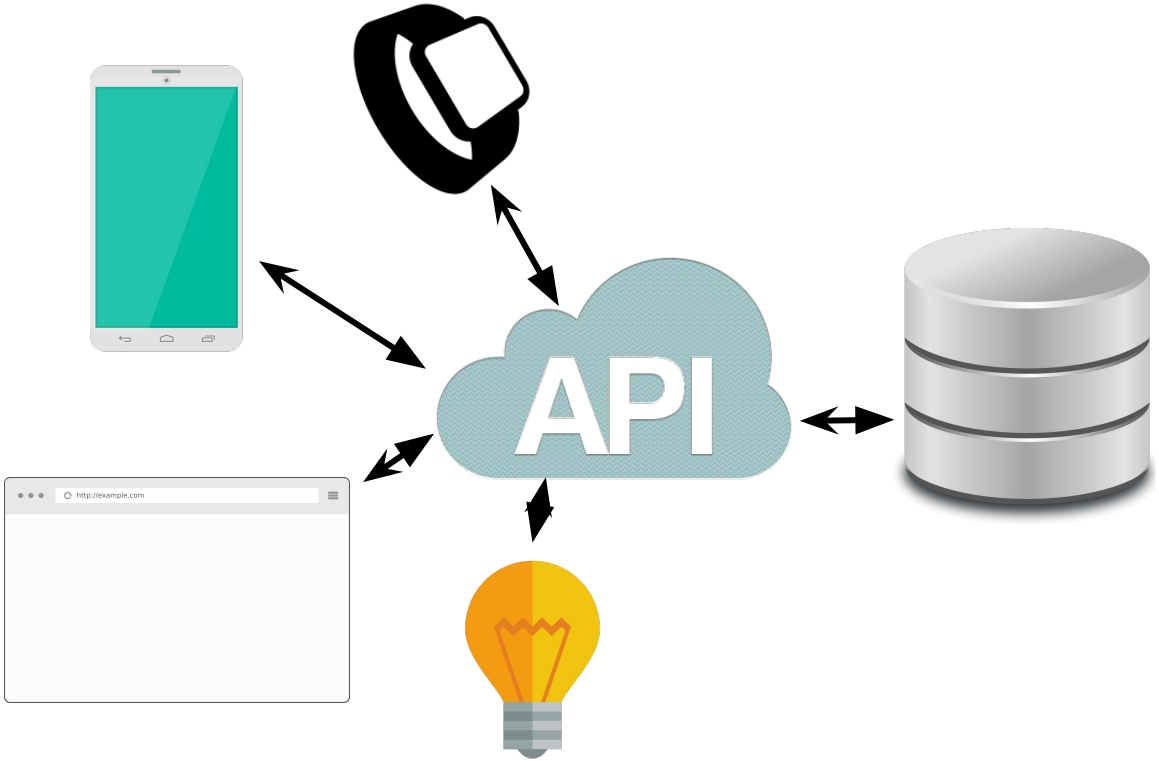


Selecione o projeto baixado:



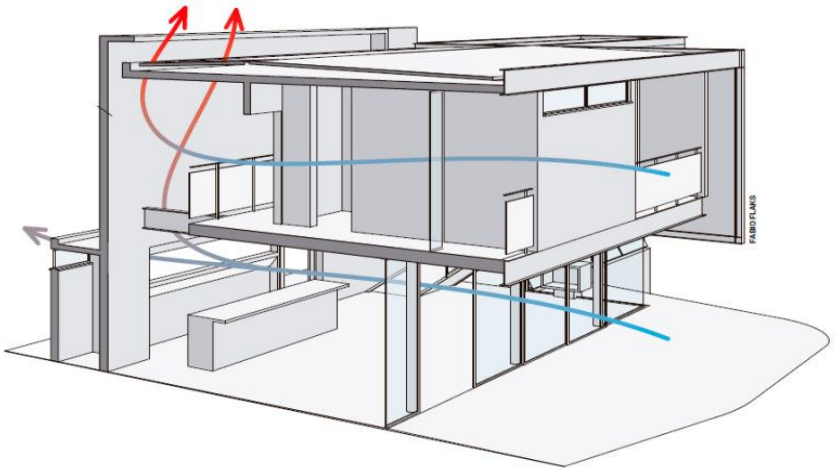
COMEÇANDO A INTEGRAÇÃO COM A API

API



API

Backend: seria toda a estrutura interna: pilares, armação, vigas, tubulação de gás e hidráulica, fiação elétrica, etc.



APIs: seriam as tomadas, torneiras, mangueira do gás e tudo que permite instalar ou remover algo com facilidade.

Frontend: seria toda parte visível: acabamento, pisos, pia, gesso e outros.

WEBSERVICES

É um conjunto de métodos acedidos e invocados por outros programas utilizando tecnologias Web.

Um Web service é utilizado para transferir dados através de protocolos de comunicação para diferentes plataformas, independentemente das linguagens de programação utilizadas nessas plataformas.

Funcionam com qualquer sistema operacional, plataforma de hardware ou linguagem de programação de suporte Web.

Permitem reutilizar sistemas já existentes numa organização e acrescentar-lhes novas funcionalidades sem que seja necessário criar um sistema a partir do zero, sendo possível melhorar os sistemas já existentes, integrando mais informação e novas funcionalidades de forma simples e rápida.

PRINCIPAIS BENEFÍCIOS

- Integração de informação e sistemas
- Reutilização de código
- Redução do tempo de desenvolvimento
- Maior segurança
- Redução de custos

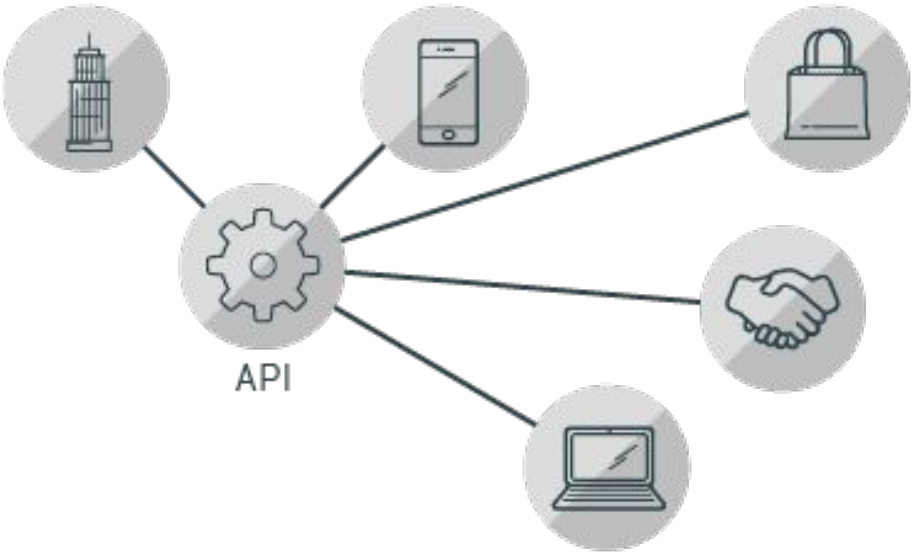
PRINCIPAIS BENEFÍCIOS

A aplicação solicita uma dessas operações, o Web service efetua o processamento e envia os dados para a aplicação que requistou a operação. A aplicação recebe os dados e faz a sua interpretação, convertendo-os para a sua linguagem própria.

Para que isso funcione é necessário uma linguagem intermédia que garanta a comunicação entre a linguagem do Web service e o sistema que faz o pedido ao Web service.

Para tal, existem protocolos de comunicação como o **SOAP (Simple Object Access Protocol) e o REST (Representational State Transfer).**

COMO FUNCIONA



REQUISIÇÕES

São os pedidos feitos pelo cliente à API.

Para que uma requisição seja válida, ela deve ter um formato específico.

O protocolo usado nas requisições é o HTTP (HyperText Transfer Protocol).

Quando você acessa um site que tem “http://” no começo, você está dizendo ao seu navegador para usar esse protocolo ao carregar o site.

REQUISIÇÕES

Quando você acessa um site que tem “http://” no começo, você está dizendo ao seu navegador para usar esse protocolo ao carregar o site.

A requisição deve ser composta por 4 partes:

- URL (Uniform Resource Locator)
- Método
- Header
- Body

URL

É o endereço que deve ser acessado.

Você sempre tem endereço que navega pelos diferentes recursos que estão sendo expostos. Por exemplo:

https://pokedexdx.herokuapp.com

Esse endereço é conhecido como o endpoint em que os recursos estarão disponíveis.

Para visualizar os pokemons, podemos acessar o seguinte recurso:

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

MÉTODO

Esse é o verbo que você usará para interagir com o recurso.

Normalmente, temos 4 opções para interagir:

GET: que pede ao servidor o recurso;

POST: que pede ao servidor que crie um recurso novo;

DELETE: que pede ao servidor que apague um recurso;

PUT: que pede ao servidor a atualização ou edição de um recurso.

HEADER

O Header contém uma lista de detalhes sobre como o cliente quer que a mensagem seja interpretada. Os diferentes servidores ou APIs podem aceitar diferentes headers.

Através de uma informação específica no header, o servidor recebe a informação e pode retornar a versão mobile (tanto do site como dados)

BODY

Aqui vão todos os parâmetros que tornam cada requisição diferente entre si, são os detalhes.

O produto que você quer comprar pode ser playstation4, televisão, camiseta ou qualquer outro.

Ao invés de criar um recurso chamado /playstation4 e outro /televisao e assim por diante, existe apenas o /**produto** e cada tipo diferente é determinado pelas informações detalhadas que vão no Body.

Esse é um exemplo que facilita na criação de recursos.

RESPONSE

Na realidade, a resposta tem uma anatomia bem definida, parecida com a requisição. Ela é composta por três partes:

Código HTTP;
Header;
Body.

O **Header** e o **Body** tem os mesmos princípios dos usados para as requisições. Um dá certas diretrizes para analisar a mensagem e o outro dá detalhes.

É importante notar que cada API pode usar suas próprias regras para definir as informações que serão dadas ao cliente na resposta.

RESPONSE

Os códigos seguem uma padronização simples para serem definidos:

1xx: Informações gerais;
2xx: Sucesso na requisição e na resposta;
3xx: Redirecionamento para outra URL;
4xx: Erro (por parte do cliente);
5xx: Erro (por parte do servidor).

Portanto, o **erro 404** se trata de uma requisição inválida, já que o cliente está pedindo ao servidor algo que não existe.

Outro código famoso é o **200 OK**, que é retornado sempre que a requisição foi entendida e retornada com sucesso.

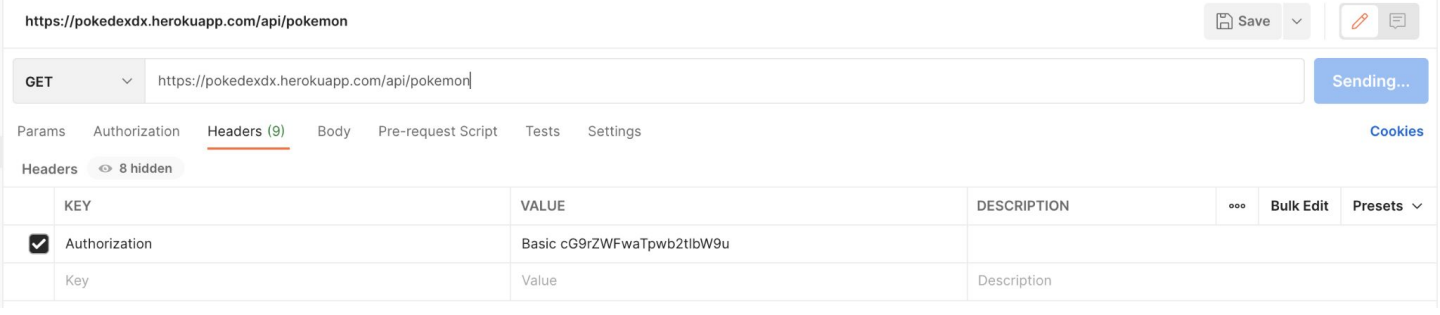
TESTANDO NO POSTMAN

Por exemplo: 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

E precisamos informar no Header da request a chave do aplicativo:
cG9rZWFwaTpwb2tlbW9u



ADICIONANDO AS DEPENDÊNCIAS

ADICIONANDO AS DEPENDÊNCIAS

Abra o arquivo **build.gradle (app)** e atualize as configurações do projeto:

```
android {

    compileSdk 32

    defaultConfig {
        applicationId "br.com.heiderlopes.pokemonwstemplatev2"
        minSdk 21
        targetSdk 32
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner
"androidx.test.runner.AndroidJUnitRunner"
    }
```

ADICIONANDO AS DEPENDÊNCIAS

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

```
implementation "org.jetbrains.kotlin:kotlin-stdlib:1.5.10"

implementation 'androidx.core:core-ktx:1.7.0'

implementation 'androidx.appcompat:appcompat:1.4.1'

implementation 'com.google.android.material:material:1.5.0'

implementation 'androidx.constraintlayout:constraintlayout:2.1.3'

testImplementation 'junit:junit:4.+'

androidTestImplementation 'androidx.test.ext:junit:1.1.3'

androidTestImplementation
'androidx.test.espresso:espresso-core:3.4.0'
```

ADICIONANDO AS DEPENDÊNCIAS

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

```
//Biblioteca para adicionar animacoes no projeto
implementation 'com.airbnb.android:lottie:4.0.0'

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

//Biblioteca para adicionar lista performatica ao aplicativo
implementation 'androidx.recyclerview:recyclerview:1.2.1'

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

//Biblioteca para consumir webservice
implementation 'com.squareup.retrofit2:retrofit:2.9.0'

//Biblioteca para realizar o parse json para objeto/objeto para json
implementation 'com.squareup.retrofit2:converter-gson:2.9.0'
implementation 'com.google.code.gson:gson:2.8.7'
```

ADICIONANDO AS DEPENDÊNCIAS

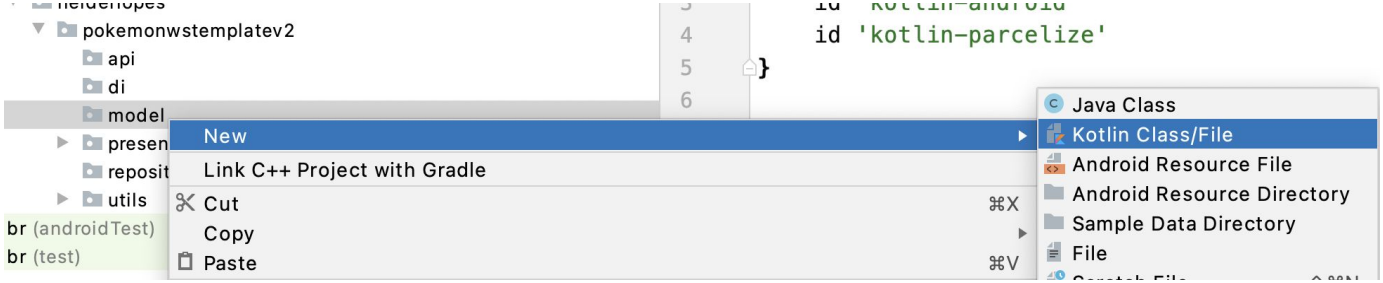
Adicione também as seguintes dependências:

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

//Biblioteca AAC
implementation "androidx.lifecycle:lifecycle-extensions:2.2.0"
implementation "androidx.lifecycle:lifecycle-viewmodel-ktx:2.4.0"

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

// Coroutines
implementation "org.jetbrains.kotlinx:kotlinx-coroutines-core:1.5.2"
implementation
"org.jetbrains.kotlinx:kotlinx-coroutines-android:1.5.2"
```

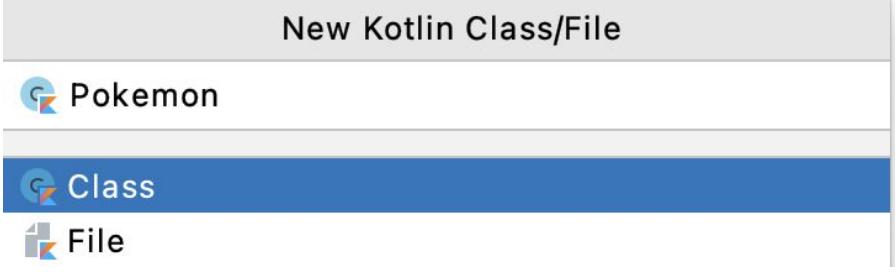
EXIBINDO A LISTA DOS POKÉMONS

CRIANDO MODEL

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



Dê o nome da classe de **Pokemon**



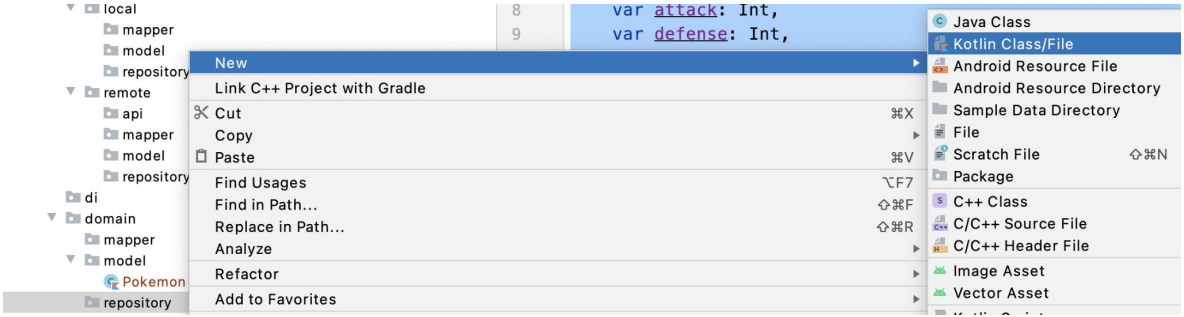
CRIANDO MODEL

Adicione o seguinte código na classe **Pokemon**

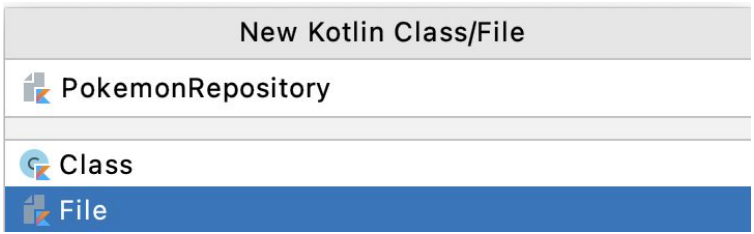
```
data class Pokemon(
    val number: String,
    val name: String,
    val imageURL: String,
    var ps: Int,
    var attack: Int,
    var defense: Int,
    var velocity: Int
)
```

CRIANDO A INTERFACE DO REPOSITÓRIO

Dentro do pacote **domain/repository** clique com o botão direito **New → Kotlin Class/File.**



Dê o nome da classe de **PokemonRepository**



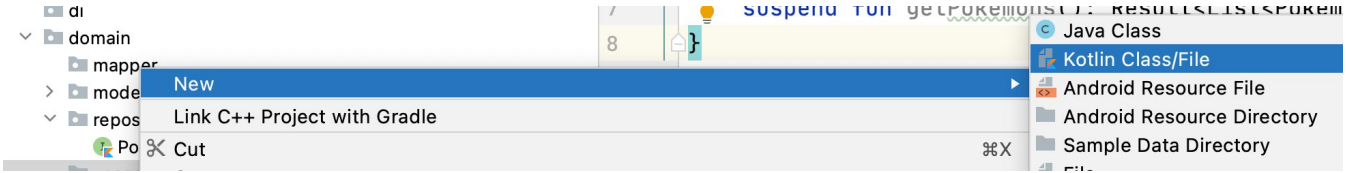
CRIANDO MODEL

Adicione o seguinte código na classe **Pokemon**

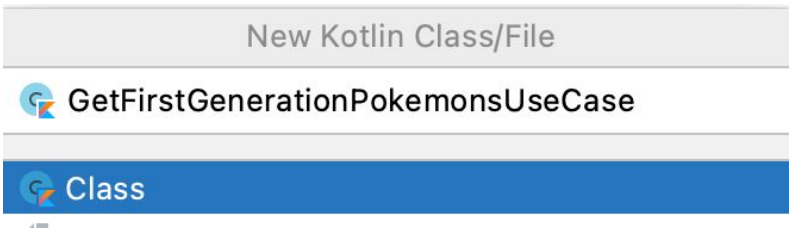
```
interface PokemonRepository {

    suspend fun getPokemons(
        size: Int,
        sort: String
    ): Result<List<Pokemon>>
}
```

CRIANDO O CASO DE USO

Dentro do pacote **domain/usecase** clique com o botão direito **New → Kotlin Class/File.**



Dê o nome da classe de **GetFirstGenerationPokemonsUseCase**



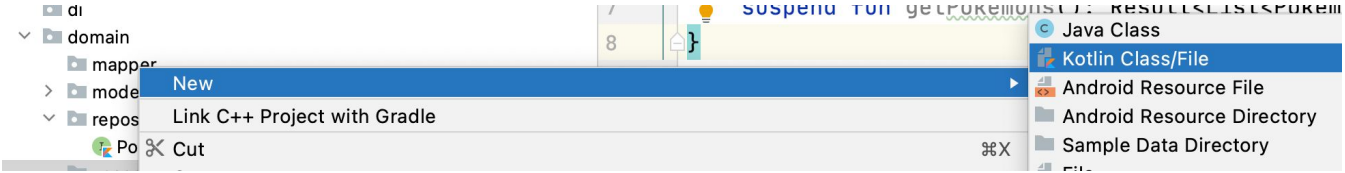
CRIANDO MODEL

Adicione o seguinte código na classe **Pokemon**

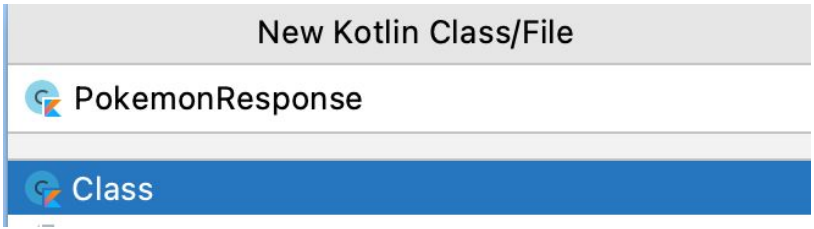
```
class GetFirstGenerationPokemonsUseCase(
    private val pokemonRepository: PokemonRepository
) {

    suspend operator fun invoke() = pokemonRepository.getPokemons(
        150,
        "number,asc"
    )
}
```

MODEL DA API

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



Dê o nome da classe de **PokemonResponse**



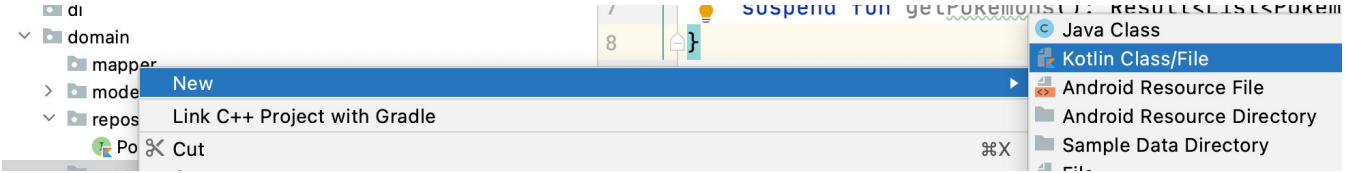
CRIANDO MODEL

Adicione o seguinte código na classe **PokemonResponse**

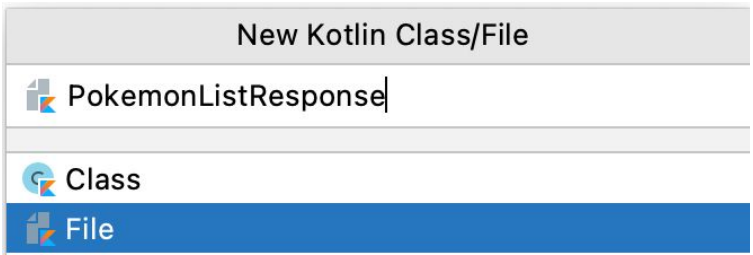
```
data class PokemonResponse(
    @SerializedName("number") val number: String,
    @SerializedName("name") val name: String,
    @SerializedName("imageURL") val imageURL: String,
    @SerializedName("ps") var ps: Int,
    @SerializedName("attack") var attack: Int,
    @SerializedName("defense") var defense: Int,
    @SerializedName("velocity") var velocity: Int
)
```

MODEL DA API

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



Dê o nome da classe de **PokemonListResponse**



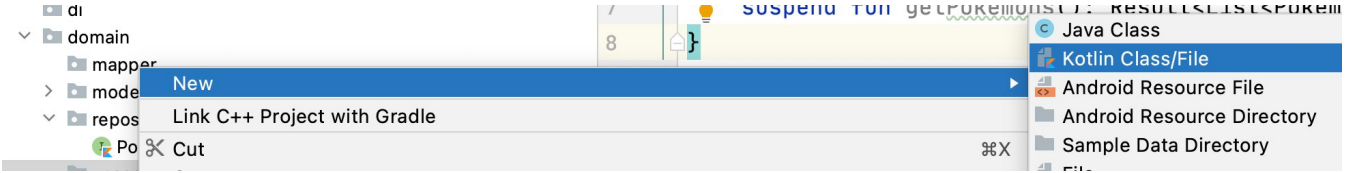
MODEL DA API

Adicione o seguinte código na classe **PokemonListResponse**

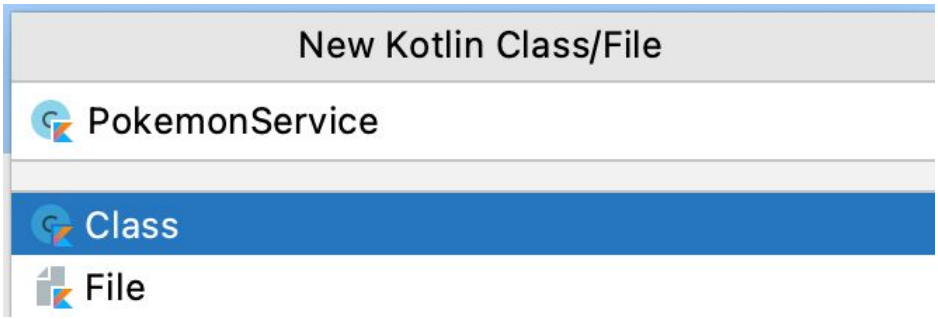
```
data class PokemonListResponse(
    @SerializedName("content")val pokemons: List<PokemonResponse>
)
```

INTERFACE DA API

Dentro do pacote **data/remote/api** clique com o botão direito **New → Kotlin Class/File.**



Dê o nome da classe de **PokemonService**



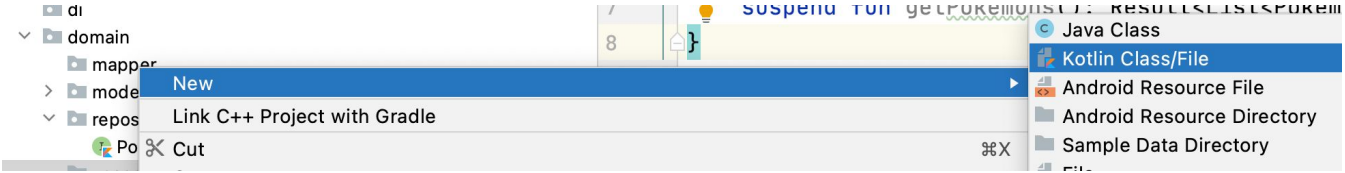
INTERFACE DA API

Adicione o seguinte código na classe **PokemonService**

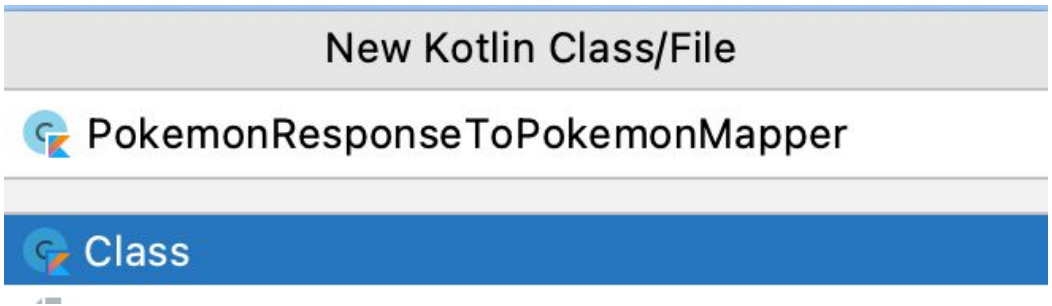
```
interface PokemonService {

    @GET("/api/pokemon")
    suspend fun getPokemons(
        @Query("size") size: Int,
        @Query("sort") sort: String): PokemonListResponse
}
```

IMPLEMENTANDO O MAPPER

Dentro do pacote **data/remote/mapper** clique com o botão direito **New → Kotlin Class/File.**



Dê o nome da classe de **PokemonResponseToPokemonMapper**



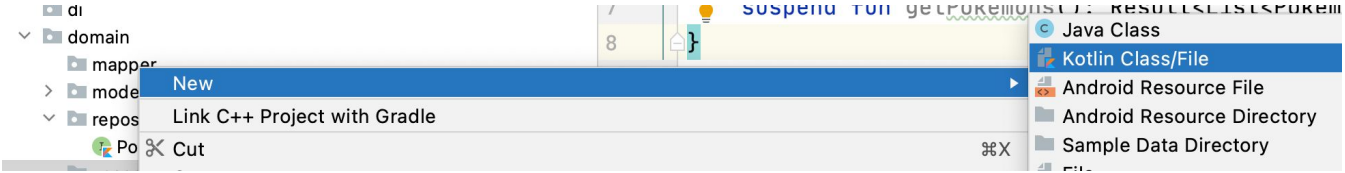
IMPLEMENTANDO O MAPPER

Adicione o seguinte código na classe **PokemonResponseToPokemonMapper**

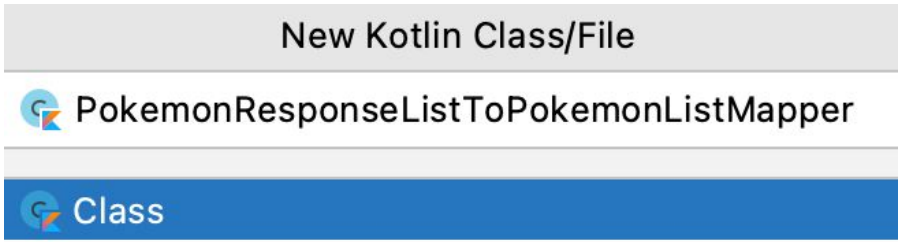
```
class PokemonResponseToPokemonMapper : Mapper<PokemonResponse,
Pokemon> {

    override fun map(source: PokemonResponse): Pokemon {
        return Pokemon(
            number = source.number,
            name = source.name,
            imageURL = source.imageURL,
            ps = source.ps,
            attack = source.attack,
            defense = source.defense,
            velocity = source.velocity,
        )
    }
}
```

IMPLEMENTANDO O MAPPER

Dentro do pacote **data/remote/mapper** clique com o botão direito **New → Kotlin Class/File.**



Dê o nome da classe de **PokemonResponseListToPokemonListMapper**



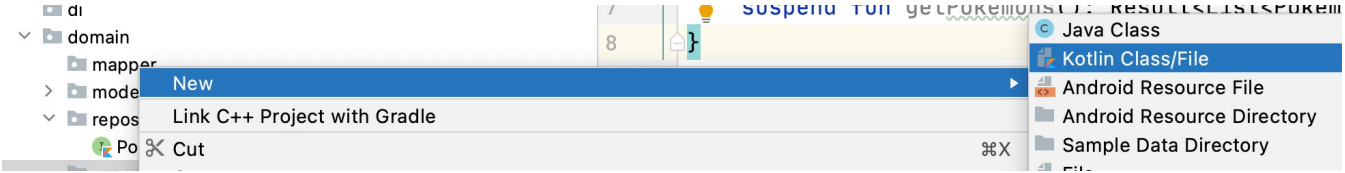
IMPLEMENTANDO O MAPPER

Adicione o seguinte código na classe
PokemonResponseListToPokemonListMapper

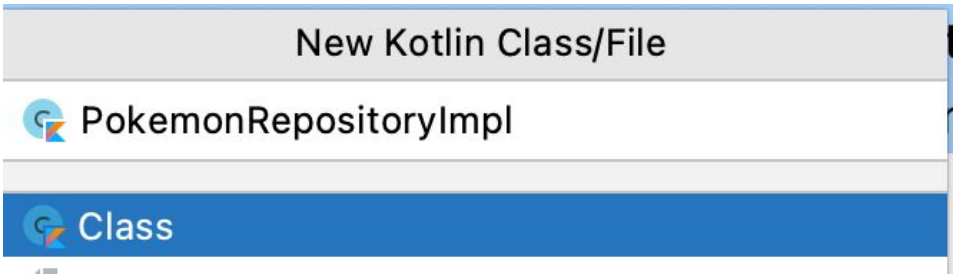
```
class PokemonResponseListToPokemonListMapper :
    Mapper<List<PokemonResponse>, List<Pokemon>> {

    private val mapper = PokemonResponseToPokemonMapper()

    override fun map(source: List<PokemonResponse>): List<Pokemon> =
        source.map { mapper.map(it) }

}
```

IMPLEMENTAÇÃO DO REPOSITÓRIO

Dentro do pacote **data/repository** clique com o botão direito **New → Kotlin Class/File.**



Dê o nome da classe de **PokemonRepositoryImpl**



UTILIZANDO O MAPPER

Volte a classe **PokemonRepositoryImpl** e adicione o seguinte código:

```
class PokemonRepositoryImpl(
   private val pokemonService: PokemonService
) : PokemonRepository {

   private val pokemonListMapper:
PokemonResponseListToPokemonListMapper =
       PokemonResponseListToPokemonListMapper()

   override suspend fun getPokemons(size: Int, sort: String):
Result<List<Pokemon>> {
       return
Result.success(pokemonListMapper.map(pokemonService.getPokemons(size
, sort).pokemons))
   }
}
```

CRIANDO O RETROFIT

Dentro da pacote **data/remote/retrofit** e adicione uma classe chamada **CacheInterceptor.** Classe responsável em implementar cache no exemplo de 5 minutos.

```
const val CACHE_CONTROL_HEADER = "Cache-Control"
const val CACHE_TIME = 5

object CacheInterceptor : Interceptor {

    override fun intercept(chain: Interceptor.Chain): Response {
        val request = chain.request()
        val originalResponse = chain.proceed(request)

        val cacheControl = CacheControl.Builder()
            .maxAge(CACHE_TIME, TimeUnit.MINUTES)
            .build()

        return originalResponse.newBuilder()
            .header(CACHE_CONTROL_HEADER, cacheControl.toString())
            .build()
    }
}
```

CRIANDO O RETROFIT

Dentro da pacote **data/remote/retrofit** e adicione uma classe chamada **AuthInterceptor.** Classe responsável em injetar o token nas chamadas.

```
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
    }
}
```

CRIANDO O RETROFIT

Dentro da pacote **data/remote/retrofit** e adicione uma classe chamada **HttpClient.** Classe utilizada para que o retrofit seja intejado, poupando criação de instancias do mesmo cada vez que uma api for chamada. O código fica desacoplado da chamada do Serviço.

```
class HttpClient(private val retrofit: Retrofit) {

    fun <T> create(service: Class<T>): T {
        return retrofit.create(service)
    }
}
```

CRIANDO O RETROFIT

Dentro da pacote **data/remote/retrofit** e adicione uma classe chamada **RetrofitClient.**

```
private const val BASE_URL = "https://pokedexdx.herokuapp.com"
private const val CACHE_SIZE = 5 * 1024 * 1024L // 5 MB de cache
class RetrofitClient(
    private val application: Context
) {
    private val gson: Gson by lazy { GsonBuilder().create() }

    private val okHttp: OkHttpClient by lazy {
        OkHttpClient.Builder()
            .cache(cacheSize())
            .addNetworkInterceptor(CacheInterceptor)
            .addInterceptor(AuthInterceptor())
            .build()
    }
```

CRIANDO O RETROFIT

Dentro da pacote **data/remote/retrofit** e adicione uma classe chamada **RetrofitClient.**

```
    fun newInstance(): Retrofit {
        return Retrofit.Builder()
            .baseUrl(BASE_URL)
            .client(okHttp)
            .addConverterFactory(GsonConverterFactory.create(gson))
            .build()
    }

    private fun cacheSize(): Cache {
        return Cache(application.cacheDir, CACHE_SIZE)
    }
}
```

CRIANDO O PICASSO

Dentro da pacote **data/remote/picasso** e adicione uma classe chamada **PicassoClient**

```
private val CACHE_SIZE = 5 * 1024 * 1024L // 5 MB de cache

class PicassoClient(
   private val application: Context
) {

   private val okHttp: OkHttpClient by lazy {
       OkHttpClient.Builder()
           .cache(cacheSize())
           .addNetworkInterceptor(CacheInterceptor)
           .addInterceptor(AuthInterceptor())
           .build()
   }
```

CRIANDO O PICASSO

Dentro da pacote **data/remote/picasso** e adicione uma classe chamada **PicassoClient**

```
    fun newInstance(): Picasso {
        return Picasso
            .Builder(application)
            .downloader(OkHttp3Downloader(okHttp))
            .build()
    }

    private fun cacheSize(): Cache {
        return Cache(application.cacheDir, CACHE_SIZE)
    }
}
```

CRIANDO A MODEL

Dentro do pacote **domain/model** crie uma classe chamada **ViewState**

```
sealed class ViewState<out T> {
    object Loading : ViewState<Nothing>()
    data class Success<T>(val data: T) : ViewState<T>()
    data class Failure(val throwable: Throwable) :
ViewState<Nothing>()
}
```

CRIANDO A MODEL

Dentro do pacote **presentation/listpokemons** crie uma classe chamada **ListPokemonsViewModel:**

```
class ListPokemonsViewModel(
    val getFirstGenerationPokemonsUseCase:
GetFirstGenerationPokemonsUseCase
) : ViewModel() {

    private val _pokemonResult =
MutableLiveData<ViewState<List<Pokemon>>>()

    val pokemonResult : LiveData<ViewState<List<Pokemon>>>
        get() = _pokemonResult
```

CRIANDO A MODEL

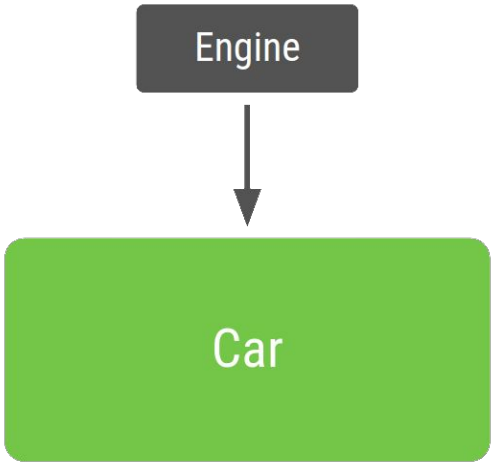
Dentro do pacote **presentation/listpokemons** crie uma classe chamada **ListPokemonsViewModel:**

```
    fun getPokemons() {

        _pokemonResult.postValue(ViewState.Loading)

        viewModelScope.launch(Dispatchers.IO) {
            runCatching {
                getFirstGenerationPokemonsUseCase()
            }.onSuccess {

_pokemonResult.postValue(ViewState.Success(it.getOrDefault(listOf())))

            }.onFailure {
                _pokemonResult.postValue(ViewState.Failure(it))
            }
        }
    }
}
```

Injeção de Dependência

DEPENDÊNCIA

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. No exemplo abaixo, a classe Car precisa de Engine.



INJEÇÃO DE DEPENDÊNCIA

A injeção de dependência **(DI, na sigla em inglês)** é uma técnica amplamente usada na programação e adequada para o desenvolvimento do Android. Ao seguir os princípios de DI, você cria a base para uma boa arquitetura do app.

INJEÇÃO DE DEPENDÊNCIA

As classes geralmente exigem referências a outras classes, ou seja, injeção de dependência é a técnica que delega a responsabilidade de inicializar dependências para o software.

Ao invés instanciarmos as dependências em algum momento do código, o próprio framework de injeção de dependência realizará esses passos pra gente.

Por exemplo, uma classe **Car** pode precisar de uma referência a uma classe **Engine**.

Essas classes solicitadas são chamadas de **dependências** e, neste exemplo, a classe **Car** depende de ter uma instância da classe **Engine** para ser executada.

VANTAGENS DO USO DE INJEÇÃO DE DEPENDÊNCIA

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).

A implementação da injeção de dependência oferece as seguintes vantagens:

- Reutilização do código
- Facilidade de refatoração
- Facilidade de teste

COMO OBTER OS OBJETOS QUE PRECISAMOS?

Exemplo 1: A classe cria a dependência necessária. No exemplo, **Car** criará e inicializará a própria instância de **Engine**.

```
class Car {

    private val engine = Engine()

    fun start() {
        engine.start()
    }
}

fun main(args: Array) {
    val car = Car()
    car.start()
}
```

COMO OBTER OS OBJETOS QUE PRECISAMOS?

Exemplo 1: Este não é um exemplo de injeção de dependência porque a classe **Car** está criando o próprio **Engine**. Isso pode ser problemático porque:

Car e **Engine** estão fortemente acoplados, ou seja, uma instância de **Car** usa um tipo de **Engine**, e nenhuma subclasse ou implementação alternativa pode ser facilmente usada.

Caso o **Car** construa o próprio **Engine**, você precisaria criar dois tipos de **Car** em vez de apenas reutilizar o mesmo Car para mecanismos de tipo **Gas** e **Electric**.

A forte dependência de **Engine** dificulta os testes. **Car** usa uma instância real de **Engine**. Isso evita o uso de um teste duplo para modificar **Engine** para diferentes casos.

COMO OBTER OS OBJETOS QUE PRECISAMOS?

Exemplo 2: Forneça-o como um parâmetro. O app pode disponibilizar essas dependências quando a classe é criada ou transmiti-las nas funções que precisam de cada dependência.

```
class Car(private val engine: Engine) {
    fun start() {
        engine.start()
    }
}

fun main(args: Array) {
    val engine = Engine()
    val car = Car(engine)
    car.start()
}
```

COMO OBTER OS OBJETOS QUE PRECISAMOS?

A função main usa **Car**. Como **Car** depende de **Engine**, o app cria uma instância de **Engine** e a usa para criar uma instância de **Car**. Os benefícios dessa abordagem baseada na injeção de dependência são:

Reutilização de Car. É possível transmitir implementações diferentes de Engine para Car:

Por exemplo, é possível definir uma nova subclasse de **Engine** chamada **ElectricEngine** que você quer que **Car** use. Se usar a injeção de dependência, tudo o que você precisará fazer é transmitir uma instância da subclasse ElectricEngine atualizada, e Car ainda funcionará sem precisar de mudanças.

Teste fácil de Car. É possível transmitir cópias de teste para testar diferentes cenários. Por exemplo, você pode criar um teste duplo de Engine chamado **FakeEngine** e configurá-lo para testes diferentes.

COMO OBTER OS OBJETOS QUE PRECISAMOS?

Há duas maneiras principais de fazer a injeção de dependência no Android:

Injeção de construtor. Esta é a maneira descrita acima. Você transmite as dependências de uma classe para o construtor.

Injeção de campo (ou injeção de setter) Algumas classes de framework do Android, como atividades e fragmentos, são instanciadas pelo sistema, e por isso, a injeção de construtor não é possível. Com a injeção de campo, as dependências são instanciadas após a criação da classe.

ALTERNATIVA À INJEÇÃO DE DEPENDÊNCIA

Uma alternativa à **injeção de dependência** é usar um **localizador de serviços.** O padrão de design do localizador de serviços também melhora o desacoplamento de classes de dependências concretas. Você cria uma classe conhecida como localizador de serviços que cria e armazena dependências e as disponibiliza sob demanda.

ALTERNATIVA À INJEÇÃO DE DEPENDÊNCIA

```
object ServiceLocator {
    fun getEngine(): Engine = Engine()
}

class Car {
    private val engine = ServiceLocator.getEngine()

    fun start() {
        engine.start()
    }
}

fun main(args: Array) {
    val car = Car()
    car.start()
}
```

ALTERNATIVA À INJEÇÃO DE DEPENDÊNCIA

O padrão do localizador de serviços é diferente da injeção de dependência na forma como os elementos são consumidos. Com o padrão do localizador de serviços, as classes têm controle e pedem objetos para serem injetados. Com a injeção de dependência, o app tem controle e injeta os objetos necessários de forma proativa.

Em comparação com a injeção de dependência:

- A coleta de dependências exigida por um localizador de serviços dificulta o teste porque todos os testes precisam interagir com o mesmo localizador de serviços global.
- As dependências são codificadas na implementação da classe, não na superfície da API. Como resultado, é mais difícil saber de fora o que uma classe precisa. Como resultado, mudanças no Car ou nas dependências disponíveis no localizador de serviços podem resultar em problemas no tempo de execução ou no teste, fazendo com que as referências falhem.
- O gerenciamento de ciclos de vida de objetos é mais difícil se você quer ter um escopo que não seja a vida útil de todo o app.

INJETANDO DEPENDÊNCIAS COM O KOIN

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

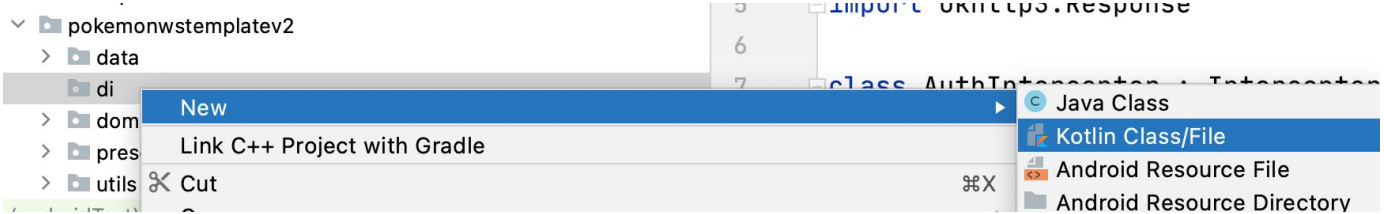
get(): é usado dentro de construtores para resolver instâncias necessárias.

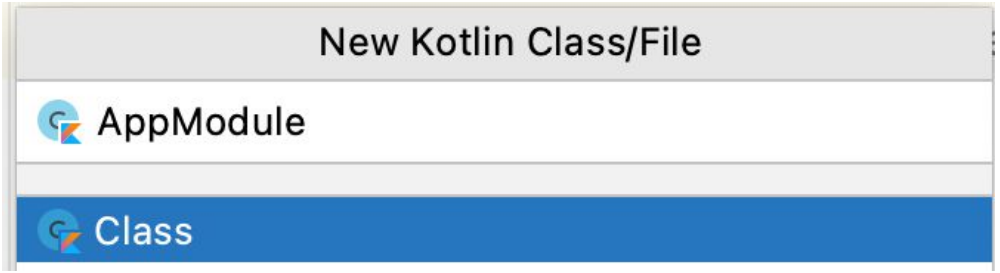
factory: é usado para indicar que uma nova instância deve ser criada sempre que for injetada.

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

KOIN MÓDULOS

Dentro do pacote **di** adicione o **AppModule:**





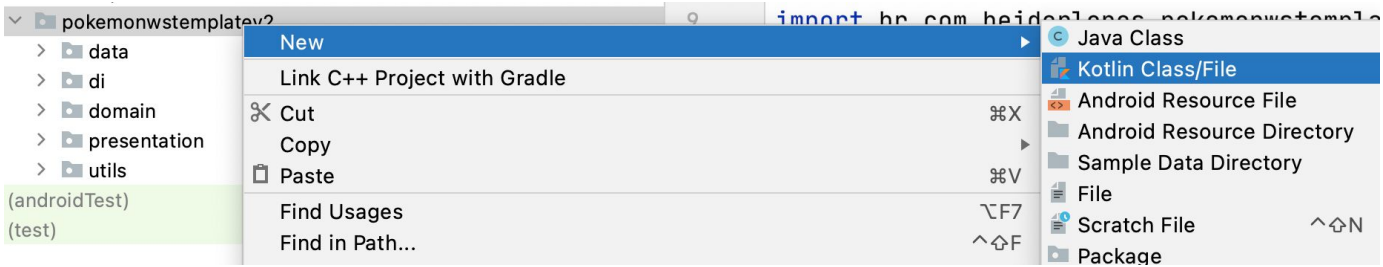
KOIN MÓDULOS

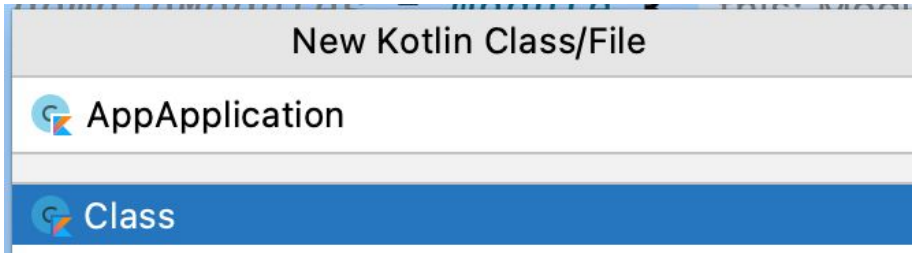
Dentro do pacote **di** adicione o **AppModule:**

```
val domainModules = module {
    factory { GetFirstGenerationPokemonsUseCase(pokemonRepository = get()) }
}

val presentationModules = module {
    viewModel { ListPokemonsViewModel(getFirstGenerationPokemonsUseCase =get()) }
}

val dataModules = module {
    factory<PokemonRepository> { PokemonRepositoryImpl(pokemonService = get()) }
}

val networkModules = module {
    single { RetrofitClient(application = androidContext()).newInstance() }
    single { HttpClient(get()) }
    factory { get<HttpClient>().create(PokemonService::class.java) }
    single { PicassoClient(application = androidContext()).newInstance()}
}
```

KOIN MÓDULOS

Na raíz do projeto crie a para inicializar o koin chamada **AppApplication**





KOIN MÓDULOS

Adicione o seguinte código dentro da classe **AppApplication** para inicializar o Koin:

```
class AppApplication : Application() {
    override fun onCreate() {
        super.onCreate()
        startKoin {
            androidContext(this@AppApplication)
            modules(domainModules)
            modules(dataModules)
            modules(presentationModules)
            modules(networkModules)
        }
    }
}
```

KOIN MÓDULOS

Abra o **AndroidManifest.xml** e adicione a permissão de acesso à Internet e associe seu **AppApplication** ao app:

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

<application
   android:name=".AppApplication"

//....
```

EXIBINDO OS POKÉMONS

Dentro do pacote **presentation/listpokemons** crie a classe **ListPokemonsAdapter** e adicione o seguinte código:

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

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
PokemonViewHolder {
        return PokemonViewHolder(
            PokemonListItemBinding.inflate(
                LayoutInflater.from(parent.context),
                parent,
                false
            )
        )
    }
```

EXIBINDO OS POKÉMONS

Dentro do pacote **presentation/listpokemons** crie a classe **ListPokemonsAdapter** e adicione o seguinte código:

```
    override fun onBindViewHolder(holder: PokemonViewHolder, position:
Int) {
        val pokemon = pokemons[position]
        holder.binding.tvPokemonName.text = pokemon.name
        holder.binding.tvPokemonNumber.text = pokemon.number

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

        holder.binding.containerPokemon.setOnClickListener {
            clickListener(pokemon)
        }
    }
    override fun getItemCount() = pokemons.size
}
```

EXIBINDO OS POKÉMONS

Exibindo os dados na **ListPokemonsActivity:**

```
class ListPokemonsActivity : AppCompatActivity() {

    private val listPokemonsViewModel: ListPokemonsViewModel by viewModel()

    val picasso: Picasso by inject()

    private val viewBinding by lazy {
        ActivityListPokemonsBinding.inflate(layoutInflater)
    }

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

        listPokemonsViewModel.getPokemons()

        registerObserver()
    }
```

EXIBINDO OS POKÉMONS

Exibindo os dados na **ListPokemonsActivity:**

```
    fun registerObserver() {
        listPokemonsViewModel.pokemonResult.observe(this, {
            when(it) {
                is ViewState.Success -> {
                    viewBinding.loading.containerLoading.visibility = View.GONE
                    viewBinding.rvPokemons.adapter = ListPokemonsAdapter(it.data, picasso) {
                        val intent = Intent(this, FormPokemonActivity::class.java)
                        intent.putExtra("POKEMON", it.number)
                        startActivity(intent)
                    }
                    viewBinding.rvPokemons.layoutManager = GridLayoutManager(this, 3)
                }
                is ViewState.Loading -> {
                    viewBinding.loading.containerLoading.visibility = View.VISIBLE
                }
                is ViewState.Failure -> {
                    viewBinding.loading.containerLoading.visibility = View.GONE
                    Toast.makeText(this, it.throwable.message, Toast.LENGTH_LONG).show()
                }
            }
        })
    }
}
```

ABRINDO A TELA DE LISTAGEM

ABRINDO A TELA DE LISTAGEM

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

```
class MainActivity : AppCompatActivity() {

    private val viewBinding by lazy {
        ActivityMainBinding.inflate(layoutInflater)
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(viewBinding.root)
        setUpListeners()
    }

    private fun setUpListeners() {
        viewBinding.btPokemonList.setOnClickListener {
            startActivity(Intent(this, ListPokemonsActivity::class.java))
        }
    }
}
```

EXERCÍCIO: IMPLEMENTE A PESQUISA PELO NÚMERO

EXIBINDO OS POKÉMONS

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

```
suspend fun getPokemon(
    number: String
): Result<Pokemon>
```

EXIBINDO OS POKÉMONS

Dentro do pacote **domain/usecases** adicione a classe **GetPokemonUseCase:**

```
class GetPokemonUseCase(
    private val pokemonRepository: PokemonRepository
) {

    suspend operator fun invoke(number: String) =
pokemonRepository.getPokemon(
        number
    )
}
```

EXIBINDO OS POKÉMONS

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

```
@GET("/api/pokemon/{number}")
suspend fun getPokemon(
    @Path("number") number: String
) : PokemonResponse
```

EXIBINDO OS POKÉMONS

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

```
private val pokemontMapper: PokemonResponseToPokemonMapper =
    PokemonResponseToPokemonMapper()

override suspend fun getPokemon(number: String): Result<Pokemon> {
    return
Result.success(pokemontMapper.map(pokemonService.getPokemon(number))
)
}
```

PESQUISANDO O POKEMON

Dentro do pacote **presentation/form** crie a classe **FormPokemonViewModel:**

```
class FormPokemonViewModel(
    val getPokemonUseCase: GetPokemonUseCase
) : ViewModel() {

    private val _pokemonResult =
MutableLiveData<ViewState<Pokemon>>()

    val pokemonResult : LiveData<ViewState<Pokemon>>
        get() = _pokemonResult
```

PESQUISANDO O POKEMON

Dentro do pacote **presentation/form** crie a classe **FormPokemonViewModel:**

```
    fun getPokemon(number: String) {

        _pokemonResult.postValue(ViewState.Loading)

        viewModelScope.launch(Dispatchers.IO) {
            runCatching {
                getPokemonUseCase(number)
            }.onSuccess {

_pokemonResult.postValue(ViewState.Success(it.getOrDefault(Pokemon("",
"", "", 0,0,0,0))))

            }.onFailure {
                _pokemonResult.postValue(ViewState.Failure(it))
            }
        }
    }
}
```

PESQUISANDO O POKEMON

Abra o arquivo **AppModule** e adicione o código em negrito:

```
val domainModules = module {
    factory { GetFirstGenerationPokemonsUseCase(pokemonRepository =
get()) }
    factory { GetPokemonUseCase(pokemonRepository = get()) }
}

val presentationModules = module {
    viewModel {
ListPokemonsViewModel(getFirstGenerationPokemonsUseCase = get()) }
    viewModel { FormPokemonViewModel(getPokemonUseCase = get()) }
}
```

EXIBINDO O POKÉMON

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

```
class FormPokemonActivity : AppCompatActivity() {

    private val formPokemonViewModel: FormPokemonViewModel by viewModel()
    val picasso: Picasso by inject()
    private lateinit var pokemon: Pokemon

    private val viewBinding by lazy {
        ActivityFormPokemonBinding.inflate(layoutInflater)
    }

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

        val pokemonNumber = intent.getStringExtra("POKEMON") ?: ""
        formPokemonViewModel.getPokemon(pokemonNumber)

        registerObserver()
    }
```

EXIBINDO O POKÉMON

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

```
fun registerObserver() {
    formPokemonViewModel.pokemonResult.observe(this, {
        when(it) {
            is ViewState.Success -> {
                setValues(it.data)
            }
            is ViewState.Loading -> {
            }
            is ViewState.Failure -> {
                Toast.makeText(this, it.throwable.message,
Toast.LENGTH_LONG).show()
            }
        }
    })
}
```

EXIBINDO O POKÉMON

```
private fun setValues(pokemon: Pokemon) {
    this.pokemon = pokemon
    viewBinding.tvPokemonNameForm.text = pokemon.name

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

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

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

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

EXIBINDO O 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?) {}
        })
    }
}
```

ATUALIZANDO O POKÉMON

ATUALIZANDO O POKEMON

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

```
suspend fun update(
    pokemon: Pokemon
): Result<Pokemon>
```

ATUALIZANDO O POKEMON

Dentro do pacote **domain/usecases** adicione a classe **UpdatePokemonUseCase:**

```
class UpdatePokemonUseCase(
    private val pokemonRepository: PokemonRepository
) {

    suspend operator fun invoke(pokemon: Pokemon) =
pokemonRepository.update(
        pokemon
    )
}
```

ATUALIZANDO O POKEMON

Dentro do pacote **data/model** adicone a classe **UpdatePokemonRequest:**

```
data class UpdatePokemonRequest(
    @SerializedName("number") val number: String,
    @SerializedName("name") val name: String,
    @SerializedName("imageURL") val imageURL: String,
    @SerializedName("ps") var ps: Int,
    @SerializedName("attack") var attack: Int,
    @SerializedName("defense") var defense: Int,
    @SerializedName("velocity") var velocity: Int
)
```

ATUALIZANDO O POKEMON

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

```
@PUT("/api/pokemon")
suspend fun updatePokemon(
    @Body pokemon: UpdatePokemonRequest
) : PokemonResponse
```

ATUALIZANDO O POKEMON

Dentro do pacote **data/mapper** adicione a classe **PokemonToUpdatePokemonRequestMapper:**

```
class PokemonToUpdatePokemonRequestMapper : Mapper<Pokemon,
UpdatePokemonRequest> {

    override fun map(source: Pokemon): UpdatePokemonRequest {
        return UpdatePokemonRequest(
            number = source.number,
            name = source.name,
            imageURL = source.imageURL,
            ps = source.ps,
            attack = source.attack,
            defense = source.defense,
            velocity = source.velocity,
        )
    }
}
```

ATUALIZANDO O POKEMON

Abra a classe **PokemonRepositoryImpl** e adicione o seguinte método:

```
private val pokemonToUpdatePokemonRequestMapper:
PokemonToUpdatePokemonRequestMapper =
    PokemonToUpdatePokemonRequestMapper()


override suspend fun update(pokemon: Pokemon): Result<Pokemon> {
    return
Result.success(pokemontMapper.map(pokemonService.updatePokemon(
        pokemonToUpdatePokemonRequestMapper.map(pokemon)
    )))
}
```

ATUALIZANDO O POKEMON

Abra a classe **FormPokemonViewModel** e adicione o seguinte código em negrito:

```
class FormPokemonViewModel(
    val getPokemonUseCase: GetPokemonUseCase,
    val updatePokemonUseCase: UpdatePokemonUseCase
) : ViewModel() {

    private val _pokemonResult =
MutableLiveData<ViewState<Pokemon>>()
    private val _pokemonUpdateResult =
MutableLiveData<ViewState<Pokemon>>()

    val pokemonUpdateResult : LiveData<ViewState<Pokemon>>
        get() = _pokemonUpdateResult
```

ATUALIZANDO O POKEMON

```
fun update(pokemon: Pokemon) {

    _pokemonUpdateResult.postValue(ViewState.Loading)

    viewModelScope.launch(Dispatchers.IO) {
        runCatching {
            updatePokemonUseCase(pokemon)
        }.onSuccess {

_pokemonUpdateResult.postValue(ViewState.Success(it.getOrDefault(Pok
emon("", "", "", 0,0,0,0))))

        }.onFailure {
            _pokemonUpdateResult.postValue(ViewState.Failure(it))
        }
    }
}
```

ATUALIZANDO O POKEMON

Dentro do pacote **presentation/form** crie a classe **FormPokemonActivity:**

```
fun registerObserver() {

    formPokemonViewModel.pokemonUpdateResult.observe(this, {
        when(it) {
            is ViewState.Success -> {
                Toast.makeText(this, "Pokémon atualizado com
sucesso", Toast.LENGTH_LONG).show()
            }
            is ViewState.Loading -> {
            }
            is ViewState.Failure -> {
                Toast.makeText(this, it.throwable.message,
Toast.LENGTH_LONG).show()
            }
        }
    })
```

ATUALIZANDO O POKEMON

Dentro do pacote **presentation/form** crie a classe **FormPokemonActivity** dentro do **onCreate** adicione o seguinte código:

```
viewBinding.btSaveForm.setOnClickListener {
    pokemon.attack = viewBinding.sbAttack.progress
    pokemon.defense = viewBinding.sbDefense.progress
    pokemon.velocity = viewBinding.sbVelocity.progress
    pokemon.ps = viewBinding.sbPS.progress
    formPokemonViewModel.update(
        pokemon
    )
}
```

ATUALIZANDO O POKEMON

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

```
val domainModules = module {
    factory { GetFirstGenerationPokemonsUseCase(pokemonRepository =
get()) }
    factory { GetPokemonUseCase(pokemonRepository = get()) }
    factory { UpdatePokemonUseCase(pokemonRepository = get()) }
}


val presentationModules = module {
    viewModel {
ListPokemonsViewModel(getFirstGenerationPokemonsUseCase = get()) }
    viewModel { FormPokemonViewModel(getPokemonUseCase = get(),
updatePokemonUseCase = get()) }
}
```

EXTRA: LENDO O QRCODE

LENDO O QRCODE

Dentro da pasta **presentation** adicione uma classe chamada **BaseScanActivity** e adicione o seguinte código:

```
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)
        }
    }
```

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)
        }
    }
}
```

LENDO O QRCODE

Abra a classe **ScanActivity** e adicione o seguinte código:

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

    private val viewBinding by lazy {
        ActivityScanBinding.inflate(layoutInflater)
    }
```

LENDO O QRCODE

Abra a classe **ScanActivity** e adicione o seguinte código:

```
@RequiresApi(Build.VERSION_CODES.M)
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)

    setContentView(viewBinding.root)

    viewBinding.permissions.btPermission.setOnClickListener {
        val intent = Intent(
            Settings.ACTION_APPLICATION_DETAILS_SETTINGS,
            Uri.parse("package:$packageName")
        )
        intent.addCategory(Intent.CATEGORY_DEFAULT)
        startActivity(intent)
    }

    super.requestPermission()
}
```

LENDO O QRCODE

Abra a classe **ScanActivity** e adicione o seguinte código:

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

LENDO O QRCODE

Abra a classe **ScanActivity** e adicione o seguinte código:

```
    override fun onPermissionDenied() {
        viewBinding.permissions.containerPermission.visibility =
View.VISIBLE
    }
    override fun onPermissionGranted() {
        viewBinding.permissions.containerPermission.visibility =
View.GONE
    }
    override fun handleResult(rawResult: Result?) {
        val pokemonNumber = rawResult?.text
        val intent = Intent(this, PokedexActivity::class.java)
        intent.putExtra("POKEMON", pokemonNumber)
        startActivity(intent)
        finish()
    }
}
```

LENDO O QRCODE

Abra a classe **MainActivity.kt** e adicione o seguinte código:

```
private fun setUpListeners() {
    viewBinding.btPokemonList.setOnClickListener {
        startActivity(Intent(this, ListPokemonsActivity::class.java))
    }

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

HORA DO EXERCÍCIO

EXERCÍCIO

Faça a implementação para exibir os dados do **Pokemon** obtido através do QRCode na activity **PokedexActivity.**

Importante: Você já tem o caso de uso necessário para pesquisar o Pokémon

OBRIGADO

 /heider.lopes

 /in/heider-lopes-a06b2869/

FIAP


FIAP