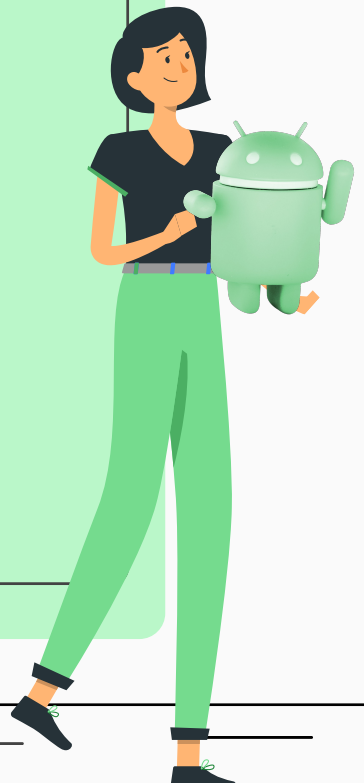
AULA 15

Retrofit e projeto final



01
COMUNICAÇÃO
INTERNET

02
RETROFIT

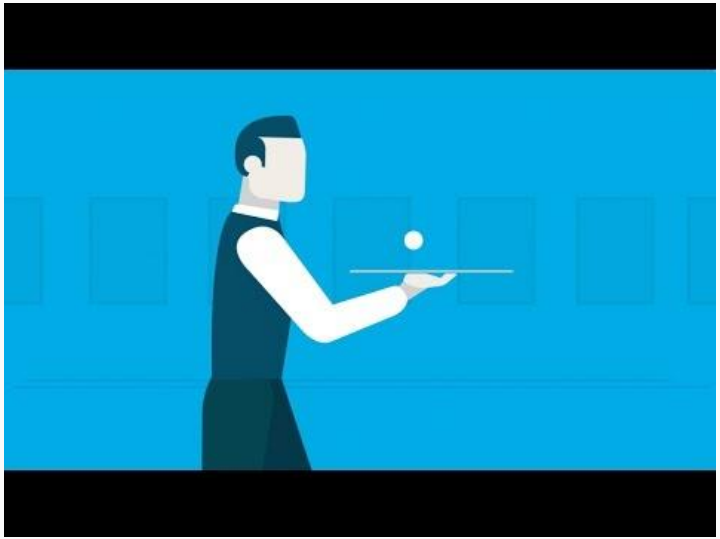
03
CARREGANDO IMAGENS

COMUNICAÇÃO COM A INTERNET

Os apps android são um frontend. Isso significa que muitas vezes, as informações que precisamos para fazer nossa aplicação funcionar vem de outra aplicação. Esse fluxo de informação é quase sempre mediado pela internet.

API

API: Application Programming Interface
é uma interface de computação que define as **interações entre vários intermediários de software**.



API REST

Os dados dos coquetéis que mostraremos na nossa app, armazenados em um servidor web, e expostos a partir de uma **API REST**. As APIS que usam a arquitetura REST são construídos usando componentes e protocolos da Web padrão.

https://www.thecocktaildb.com/api.php

JSON

A resposta de um serviço da web é comumente formatada em **JSON**, que é um formato para representar dados estruturados. Um objeto JSON é uma coleção de pares de valores-chave, às vezes chamados de dicionário, mapa de hash ou matriz associativa. Uma coleção de objetos JSON é um **array JSON, e é o array que você recebe como resposta de um serviço da web.**

JSON


```
{
  "drinks": [
    {
      "idDrink": "11007",
      "strDrink": "Margarita",
      "strDrinkAlternate": null,
      "strDrinkES": null,
      "strDrinkDE": null,
      "strDrinkFR": null,
      "strDrinkZH-HANS": null,
      "strDrinkZH-HANT": null,
      "strTags": "IBA,ContemporaryClassic",
      "strVideo": null,
      "strCategory": "Ordinary Drink",
      "strIBA": "Contemporary Classics",
      "strAlcoholic": "Alcoholic",
      "strGlass": "Cocktail glass",
      "strInstructions": "Rub the rim of the glass with the li
      of the imbiber and never mix into the cocktail. Shake th
      "strInstructionsES": null,
```

JSON

A resposta de um serviço da web é comumente formatada em **JSON**, que é um formato para representar dados estruturados. Um objeto JSON é uma coleção de pares de valores-chave, às vezes chamados de dicionário, mapa de hash ou matriz associativa. Uma coleção de objetos JSON é um **array JSON, e é o array que você recebe como resposta de um serviço da web.**

Exemplo: https://www.decolar.com

RETROFIT

Para acessar dados externos no aplicativo, é necessário uma conexão de rede e se comunicar com um servidor. Além disso, receber e analisar os dados de resposta em um formato que o aplicativo possa usar. O Retrofit é uma biblioteca que nos auxilia a fazer essa conexão.

DEPENDENCIAS

Crie o projeto Today's Cocktail a partir de uma activity vazia (empty activity)
Adicione as seguintes dependencias:

https://gist.github.com/marcellalcs/dfabe01c089ed05d3c2238a270fd233c

DEPENDENCIAS

Adicione as versões no arquivo build.gradle do módulo, confira se as opções abaixo estão configuradas

```
compileOptions {
    sourceCompatibility JavaVersion.VERSION_1_8
    targetCompatibility JavaVersion.VERSION_1_8
}
kotlinOptions {
    jvmTarget = '1.8'
}
```

O QUE O RETROFIT FAZ?

O retrofit cria uma API de rede para o aplicativo com base no conteúdo do serviço da web.

Ele busca dados do serviço da web e os encaminha por meio de uma biblioteca conversora que sabe como decodificar os dados e retorná-los na forma de objetos úteis.

O retrofit inclui suporte integrado para formatos de dados da web populares, como XML e JSON. Por fim, o Retrofit cria a maior parte da camada de rede, incluindo detalhes críticos, como a execução de solicitações em threads.

COCKTAIL DATA OBJECT

Quando fazemos uma chamada à nossa API, o retorno será assim:


```
{
  "drinks": [
    {
      "strDrink": "'57 Chevy with a White License Plate",
      "strDrinkThumb": "https://www.thecocktaildb.com/images/media/drink/qyyvtu1468878544.jpg",
      "idDrink": "14029"
    },
    {
      "strDrink": "1-900-FUK-MEUP",
      "strDrinkThumb": "https://www.thecocktaildb.com/images/media/drink/uxywyw1468877224.jpg",
      "idDrink": "15395"
    },
    {
      "strDrink": "110 in the shade",
      "strDrinkThumb": "https://www.thecocktaildb.com/images/media/drink/xxyywq1454511117.jpg",
      "idDrink": "15423"
    },
    {
      "strDrink": "151 Florida Bushwacker",
      "strDrinkThumb": "https://www.thecocktaildb.com/images/media/drink/rvwrvv1468877323.jpg",
      "idDrink": "14588"
    },
```

COCKTAIL DATA OBJECT

A resposta JSON é uma matriz, indicada pelos colchetes.

A matriz contém objetos JSON, que são cercados por chaves.
Cada objeto contém um conjunto de pares chave-valor, separados por dois pontos.

As chaves estão entre aspas, os valores podem ser números ou strings, e as strings também estão entre aspas.

COCKTAIL DATA OBJECT

A lib Moshi analisa esses dados JSON e os converte em objetos Kotlin.

Para fazer isso, é necessário ter uma classe de dados Kotlin para armazenar os resultados retornados, portanto, a próxima etapa é criar a **data classe Cocktail** no pacote network.

```
data class Cocktail(
    val id: Int,
    val name: String,
    val thumbUrl: String
)
```

COCKTAIL DATA OBJECT

Observe que cada uma das variáveis na classe Cocktail corresponde a uma propriedade no objeto JSON.

Quando Moshi analisa o JSON, ele combina as **chaves por nome e preenche os objetos de dados com os valores apropriados.**

COCKTAIL DATA OBJECT

Para a combinação acontecer automaticamente, os nomes dos campos do data class deveriam serguir o mesmo nome da chave no objeto json.
Às vezes, os nomes de chave em uma resposta JSON podem tornar as propriedades Kotlin confusas ou podem não corresponder ao seu estilo de codificação

Por isso usaremos a anotação **@Json (name = "nome_da_chave_no_JSON")** para ajudar o Moshi a entender qual campo deve ser preenchido com qual valor do JSON.

COCKTAIL DATA OBJECT

```
data class Cocktail(
    @Json(name = "idDrink")
    val id: Int,
    @Json(name = "strDrink")
    val name: String,
    @Json(name = "strDrinkThumb")
    val thumbUrl: String
)
```

COCKTAILRESPONSE DATA OBJECT


```
"drinks": [
    {
      "strDrink": "'57 Chevy",
      "strDrinkThumb":
"https:\/\/www.thecocktaildb.com\/images\/media\/drink\/qyyvtu1468878544.jp
g",
      "idDrink": "14029"
    },
    {
      "strDrink": "1-900-FUK-MEUP",
      "strDrinkThumb":
"https:\/\/www.thecocktaildb.com\/images\/media\/drink\/uxywyw1468877224.jp
g",
      "idDrink": "15395"
    }
]
```

COCKTAILRESPONSE DATA OBJECT

Resposta = [cocktail1, cocktail2, cocktail3 ...]

COCKTAILRESPONSE DATA OBJECT

```
data class CocktailsResponse (
        @Json(name = "drinks")
        val cocktailsList: List<Cocktail>)
```

Começaremos estabelecendo a comunicação com nosso serviço.
Crie um pacote network e dentro dele o arquivo CocktailsApiService.

Nesse arquivo defina a url base:

```
private const val BASE_URL =
"https://www.thecocktaildb.com/api/json/v1/1/"
```

Crie um objeto moshi

```
private val moshi = Moshi.Builder()
    .add(KotlinJsonAdapterFactory())
    .build()
```

Crie um objeto retrofit

```
private val retrofit = Retrofit.Builder()
   .addConverterFactory(ScalarsConverterFactory.create())
    .baseUrl(BASE_URL)
    .build()
```

O retrofit precisa de pelo menos duas coisas disponíveis para construir uma API de serviços da web:
o URI básico para o serviço da web e uma fábrica de conversores.

O conversor informa ao Retrofit o que fazer com os dados que recebe do serviço da web.
O retrofit usará o Moshi para realizar a conversão dos dados do JSON para nosso objeto cocktail.
Por isso adicionamos à chamada addConverterFactory() no construtor com uma instância de Moshi.

Ainda no arquivo, complete o código com a interface:

```
interface CocktailsApiService {
    @GET( "filter.php?a=Alcoholic")
    suspend fun getCocktails():CocktailResponse
}
```

Por enquanto queremos uma de coquetéis. Essas informações serão obtidas a partir da resposta do serviço da web. Para acessar essa resposta dentro da nossa aplicação, usaremos o método **getCocktails()**.

Para informar ao Retrofit o que esse método deve fazer, usamos uma anotação **@GET** e especificamos o caminho, para o serviço da web correspondente.

O Retrofit anexa o estado real do caminho passado à URL base (que você definiu no construtor Retrofit) e cria um **objeto Call**. Esse objeto Call é usado para iniciar a solicitação.

Ainda no mesmo arquivo, defina o objeto CocktailApi:

```
object CocktailsApi {
    val retrofitService: CocktailsApiService by lazy {
        retrofit.create(CocktailsApiService::class.java)
    }
}
```

O método Retrofit **create()** cria o próprio serviço Retrofit com a interface CoktailsApiService.

Essa criação é custosa e nosso app só precisa de **uma instância** de serviço de Retrofit. Por isso, expomos o serviço CoktailsApi para restante do aplicativo usando um objeto público.

Ele é inicializado *preguiçosamente*, o que signfica que o atributo *retrofitService* do objeto só será realmente preenchido quando alguém precisar.

Toda a configuração está feita! Cada vez que seu aplicativo chamar ***CocktailsApi.retrofitService***, ele obterá um objeto Retrofit **singleton** que implementa **CocktailsApiService**.

REPOSITORY

A classe CocktailsListRepository será responsável por fornecer os dados dos drinks para a ViewModel.

Ela fará a comunicação com o serviço externo (e futuramente salvará essas infos num banco de dados).

Crie um novo pacote e adicione a classe CocktailsListRepository

REPOSITORY

O código da classe:
https://gist.github.com/marcellalcs/1559dae1f4cada4c6f55a9c09e56ed09

VIEW MODEL

A Classe CocktailsListViewModel será responsável exclusivamente por recuperar as informações da lista de drinks e repassá-la para activity

Crie o pacote viewmodel e adicione esta classe:

```
class CocktailsListViewModel(private val repository:
CocktailsListRepository): ViewModel() {
    val  cocktailList: LiveData<List<Cocktail>>
        get() = repository.cocktailList
}
```

VIEW MODEL FACTORY

Como nossa viewmodel precisa de repository para se construir, criaremos um ConctailsListViewModelFactory

```
class CocktailsListViewModelFactory(private val repository:
CocktailsListRepository):  ViewModelProvider.Factory {
    override fun <T : ViewModel?> create(modelClass: Class<T>): T {
        if (modelClass.isAssignableFrom(CocktailsListViewModel::class.java)) {
            return CocktailsListViewModel(repository) as T
        }
        throw IllegalArgumentException("Unknown ViewModel class")
    }
}
```

RESULTADO NA ACTIVITY

Por fim, nossa activity mostrará informações na tela:

```
val viewModelFactory =
CocktailsListViewModelFactory(CocktailsListRepository())
val viewModel = ViewModelProvider(this,
viewModelFactory).get(CocktailsListViewModel::class.java)
val list = viewModel.cocktailList
list.observe(this, Observer {
    findViewById<TextView>(R.id.textView).text = it[0].name
})
```

PALAVRAS CHAVE

- RETROFIT
- API
- JSON
- LIVEDATA
- SUSPEND
- POSTVALUE