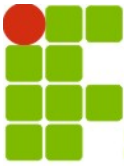

INSTITUTO FED. DE EDUCAÇÃO, CIÊNC. E TEC. DE PERNAMBUCO
CURSO: TEC. EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS
DISCIPLINA: PROGRAMAÇÃO PARA DISPOSITIVOS MÓVEIS
PROFESSOR: RAMIDE DANTAS
ASSUNTO: ACESSO À REDE USANDO RETROFIT

Prática 07

Obs.: esta prática é continuação da prática 06; use controle de Versões

Parte 1: Refatoração e Arrumação

Passo 1: Crie a classe `Repository` no pacote `pdm.weatherapp.repo` como abaixo:

```
class Repository (private var listener : Listener): FBDatabase.Listener {
    private var fbDb = FBDatabase (this)
    //private var weatherService = WeatherService()

    interface Listener {
        fun onUserLoaded(user: User)
        fun onCityAdded(city: City)
        fun onCityRemoved(city: City)
    }

    fun addCity(name: String) {
         fbDb.add(City(name, LatLng(0.0,0.0)))
    }

    fun addCity(lat: Double, lng: Double) {
         fbDb.add(City("Cidade@$lat:$lng", LatLng(lat,lng)))
    }

    fun remove(city: City) {
        fbDb.remove(city)
    }

    fun register(userName: String, email: String) {
        fbDb.register(User(userName, email))
    }

    override fun onUserLoaded(user: User) {
        listener.onUserLoaded(user)
    }

    override fun onCityAdded(city: City) {
        listener.onCityAdded(city)
    }

    override fun onCityRemoved(city: City) {
        listener.onCityRemoved(city)
    }
}
```

Essa classe centralizará o acesso a dados na nossa arquitetura.

Passo 2: Modifique a classe `City` como abaixo:

```
data class City(
    val name: String,
    var location: LatLng? = null,
    var weather: String? = null,
    var img_url: String? = null,
    var bitmap: Bitmap? = null
)
```

Faça as modificações necessárias nas classes da UI.

Passo 3: Faça as seguintes modificações em `MainViewModel`:

```
class MainViewModel : BaseViewModel(), Repository.Listener {

    private val _cities = mutableStateMapOf<String, City>()
    val cities : List<City>
        get() = _cities.values.toList()

    ...

    override fun onUserLoaded(user: User) { _user.value = user  }

    override fun onCityAdded(city: City) { _cities[city.name] = city }

    override fun onCityRemoved(city: City) { _cities.remove(city.name) }
}
```

Em vez uma lista usaremos um mapa de cidades. Isso facilita a atualização das cidades, o que força a atualização da UI (recomposição). Adicionamos os tratadores de eventos para quanto a cidade e usuário são atualizados.

Passo 4: Em `MainActivity`, onde instanciávamos `FBDatabase`, crie uma instância de Repository:

```
val repo = remember { Repository (viewModel) }
```

Passo 5: Em `MainActivity`, onde usávamos a instância de `FBDatabase`, usaremos `Repository`:

```
if (showDialog.value) CityDialog(
    onDismiss = { showDialog.value = false },
    onConfirm = { cityName ->
        if (cityName.isNotBlank())
            repo.addCity(name = cityName)
        showDialog.value = false
    })
```

Passo 5: Mude `MainNavHost` para receber um `Repository` em vez de `FBDatabase`.

Repasse esse parâmetro para `HomePage`, `ListPage` e `MapPage`, substituindo `FBDatabase` por `Repository`.

Passo 6: Em `MapPage`, onde é usado `FBDatabase` mude para `Repository`:

```
onMapClick = {
        repo.addCity(lat = it.latitude, long = it.longitude)
}
```

Remova os marcadores adicionados manualmente, se houver.

Passo 7: Em `ListPage`, onde é usado `FirebaseDB.remove()`, substitua por `Repository.remove()`.

Passo 8: Rode e teste a aplicação. Corrija o que for necessário e faça um *commit*.

Parte 2: Preparação e Configuração

Passo 1: Faça seu cadastro no site **WeatherAPI.com** e depois o login:

URL: https://www.weatherapi.com/

No seu *Dashboard* deve aparecer a chave de API que usaremos na prática.

Passo 2: Copie sua chave (API Key) para o arquivo **local.properties**:

```
sdk.dir=C\:\\Users\\...
MAPS_API_KEY="AIzaFFFFFFFFFFFFFFFFFFFFFFFFFFFFF"
WEATHER_API_KEY="<SUA CHAVE AQUI>"
```

ATENÇÃO: Esse arquivo deve estar listado no **.gitignore** para não ser incluído nos *commits*, evitando que suas chaves sejam publicadas no GitHub.

Passo 3: Modifique o arquivo **build.gradle.kts** (módulo app) como abaixo:

```
import java.util.Properties
...
android {
    ...

    defaultConfig {
        ...
        val keyFile = project.rootProject.file("local.properties")
        val props = Properties()
        props.load(keyFile.inputStream())
        buildConfigField ("String", "WEATHER_API_KEY",
                                    props.getProperty("WEATHER_API_KEY"))
    }
    ...
    buildFeatures {
        ...
        buildConfig = true
    }
    ...
}
...
```

Passo 4: Ainda no **build.gradle.kts**, adicione as dependências para o Retrofit:

```
dependencies {
    ...
    implementation ("com.squareup.retrofit2:retrofit:2.9.0")
    implementation ("com.squareup.retrofit2:converter-gson:2.9.0")
    ...
}
```

O pacote GSON faz a serialização/deserialização das respostas JSON para objetos Kotlin.

Passo 5: Compile e rode o aplicativo.

Não deve haver diferença significativa ainda, mas veja se nada parou de funcionar. Se estiver tudo ok, faça um novo *commit*.

ATENÇÃO: Pode ser necessário dar um *build clean* após modificar o **build.gradle.kts**; só o *sync* no projeto pode não ser suficiente.

Parte 3: Fazendo as requisições com Retrofit

Passo 1: Crie a classe `Location` em `pdm.weatherapp.api`:

```
data class Location (
    var id : String? = null,
    var name : String? = null,
    var region : String? = null,
    var country : String? = null,
    var lat : Double? = null,
    var lon: Double? = null,
    var url : String? = null
)
```

Os objetos desse classe mapeiam o JSON que será recebido da API.

Passo 2: Crie a interface `WeatherServiceAPI` em `pdm.weatherapp.api`:

```
interface WeatherServiceAPI {
    companion object {
        const val BASE_URL = "https://api.weatherapi.com/v1/"
        const val API_KEY = BuildConfig.WEATHER_API_KEY
    }
    // Procura a localização baseado no nome ou coordenadas
    @GET("search.json?key=$API_KEY&lang=pt_br")
    fun search(@Query("q") query: String): Call<List<Location>?>
}
```

Essa interface define as URLs usadas para acessar a API climática, neste caso somente a usada para buscar cidades por nome ou coordenadas. **ATENÇÃO:** Ao importar dependências, veja se é dos pacotes do aplicativo ou Retrofit2.

Passo 3: Crie o objeto `WeatherService` em `pdm.weatherapp.api`:

```
class WeatherService {
    private var weatherAPI: WeatherServiceAPI

    init {
        val retrofitAPI = Retrofit.Builder()
            .baseUrl(WeatherServiceAPI.BASE_URL)
            .addConverterFactory(GsonConverterFactory.create()).build()
        weatherAPI = retrofitAPI.create(WeatherServiceAPI::class.java)
    }

    fun getName(lat: Double, lng: Double, onResponse : (String?) -> Unit ) {
        search("$lat,$lng") { loc ->  onResponse (loc?.name)  }
    }

    fun getLocation(name: String,
                    onResponse: (lat:Double?, long:Double?) -> Unit) {
        search(name) { loc -> onResponse (loc?.lat, loc?.lon)  }
    }

    private fun search(query: String, onResponse : (Location?) -> Unit) {
        val call: Call<List<Location>?> = weatherAPI.search(query)
        call.enqueue(object : Callback<List<Location>?> {
            override fun onResponse(call: Call<List<Location>?>,
                                    response: Response<List<Location>?>) {
                onResponse(response.body()?.get(0))
            }
            override fun onFailure(call: Call<List<Location>?>,t: Throwable) {
                Log.w("WeatherApp WARNING", "" + t.message)
                onResponse(null)
            }
        })
    }
}
```

A classe `WeatherService` realiza as chamadas a API usando o Retrofit. Os métodos `getName()` e `getLocation()` usam do método `search()` para descobrir o nome ou coordenadas de uma cidade, respectivamente.

Passo 4: Modifique a classe `Repository` para acessar a API climática:

```
class Repository (private var listener : Listener): FBDatabase.Listener {
    private var fbDb = FBDatabase (this)
    private var weatherService = WeatherService()

    ...

    fun addCity(name: String) {
        weatherService.getLocation(name) { lat, lng ->
            fbDb.add(City(name = name, weather = "loading...",
                location = LatLng(lat?:0.0, lng?:0.0)))
        }
    }

    fun addCity(lat: Double, lng: Double) {
        weatherService.getName(lat, lng) { name ->
            fbDb.add( City( name = name?:"NOT_FOUND",
                    location = LatLng(lat, lng)))
        }
    }

    ...
}
```

Os métodos `addCity()` realizam uma chamada a API climática para obter as coordenadas ou nome da cidade adicionada, só realizando a inclusão da cidade no Firebase após o retorno dessa chamada.

Passo 5: Rode e teste o aplicativo. Dê um novo *commit* se estiver tudo certo.

Nesse estágio, ao adicionar uma cidade pelo nome (via diálogo), serão buscadas as coordenadas da cidade via API antes da cidade ser adicionada ao Firebase. Ao adicionar a cidade clicando no mapa, será buscado o nome da cidade baseado nas coordenadas do clique antes da cidade ser adicionada ao Firebase.