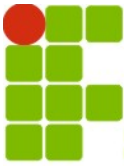

INSTITUTO FED. DE EDUCAÇÃO, CIÊNC. E TEC. DE PERNAMBUCO
CURSO: TEC. EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS
DISCIPLINA: PROGRAMAÇÃO PARA DISPOSITIVOS MÓVEIS
PROFESSOR: RAMIDE DANTAS
ASSUNTO: ACESSO À REDE USANDO RETROFIT 2

Prática 08

Obs.: esta prática é continuação da prática 07; use controle de Versões

Parte 1: Mostrando as condições climáticas atuais na lista de cidades

Passo 1: Renomeie a classe `api.Location` para `api.APILocation`, para evitar confusão com as classes do sistema.

Passo 2: Crie as classes abaixo no pacote `api`:

```
data class APICondition (
    var text : String? = null,
    var icon : String? = null
)

data class APIWeather (
    var last_updated: String? = null,
    var temp_c : Double? = 0.0,
    var maxtemp_c: Double? = 0.0,
    var mintemp_c: Double? = 0.0,
    var condition : APICondition? = null
)

data class APICurrentWeather (
    var APILocation : APILocation? = null,
    var current : APIWeather? = null
)
```

Nos nomes dos atributos e a organização dos objetos e sub-objetos são usados para deserializar as respostas JSON que recebemos da API climática, o que é feito automaticamente pela biblioteca GSON usada pelo Retrofit.

Passo 3: Adicione a função abaixo à declaração de `api.WeatherServiceAPI`:

```
  @GET("current.json?key=$API_KEY&lang=pt")
  fun currentWeather(@Query("q") query: String): Call<APICurrentWeather?>
```

Esse novo *endpoint* retorna as condições climáticas atuais da cidade.

Passo 4: Em `api.WeatherService`, adicione a função genérica abaixo:

```
private fun <T> enqueue(call : Call<T?>, onResponse : ((T?) -> Unit)? = null){
    call.enqueue(object : Callback<T?> {
        override fun onResponse(call: Call<T?>, response: Response<T?>) {
            val obj: T? = response.body()
            onResponse?.invoke(obj)
        }

        override fun onFailure(call: Call<T?>, t: Throwable) {
            Log.w("WeatherApp WARNING", "" + t.message)
        }
    })
}
```

Essa função genérica faz o enfileiramento das requisições do Retrofit, evitando a repetição de código. Refatore a funções `search()` em `WeatherService` para utilizar essa função.

Passo 5: Ainda em `api.WeatherService`, adicione a função abaixo:

```
fun getCurrentWeather(name: String, onResponse: (APICurrentWeather?) -> Unit){
    val call: Call<CurrentWeather?> = weatherAPI.currentWeather(name)
    enqueue(call) { onResponse.invoke(it) }
}
```

Essa função aciona o nomo *endpoint* para buscar as condições climáticas atuais da cidade, usando a nossa função genérica do Retrofit.

Passo 6: Crie a classe `model.Weather` abaixo:

```
data class Weather (
    val date: String,
    val desc: String,
    val temp: Double,
    val imgUrl: String,
    var bitmap: Bitmap? = null
)
```

Essa classe modela as informações climáticas que usaremos na UI. É uma versão simplificada e independente de provedor das classes que usamos para buscar as informações da API climática.

Passo 7: Modifique a classe `model.City` como abaixo:

```
data class City(
    val name: String,
    var location: LatLng? = null,
    var weather: Weather? = null,
    //var forecast: List<Forecast>? = null, // Usada mais a frente
)
```

Passo 8: Em `repo.Repository`, adicione o tratador de eventos `onCityUpdated` na interface do `Listener`:

```
class Repository (private var listener : Listener): FBDatabase.Listener {
    ...
    interface Listener {
        ...
        fun onCityUpdated(city: City)
    }
    ...
}
```

Esse tratador é chamado quando a cidade é atualizada e será implementado pelo `MainViewModel` para atualizar a lista de cidades e atualizar a UI.

Passo 9: Em `repo.Repository`, adicione o método abaixo:

```
fun loadWeather(city: City) {
    weatherService.getCurrentWeather(city.name) { apiWeather ->
        city.weather = Weather (
            date = apiWeather?.current?.last_updated?:"...",
            desc = apiWeather?.current?.condition?.text?:"...",
            temp = apiWeather?.current?.temp_c?:-1.0,
            imgUrl = "https:" + apiWeather?.current?.condition?.icon
        )
        listener.onCityUpdated(city)
    }
}
```

Esse código dispara uma busca pelo clima atual. Quando a resposta é retornada, o tratador `onCityUpdated()` é chamado.

Passo 10: No `MainViewModel`, implemente `onCityUpdated`:

```
override fun onCityUpdated(city: City) {
    _cities.remove(city.name)
    _cities[city.name] = city.copy()
}
```

Com esse código, quando a cidade é atualizada (por ex.: clima atual retornado pela API), o *map* contendo as cidades é modificado, fazendo a UI atualizar.

Passo 11: Em `ListPage`, adicione o código abaixo:

```
@Composable
fun ListPage(...) {
    val activity = LocalContext.current as? Activity

    val cityList = viewModel.cities
    LazyColumn ( ... ) {
        items(cityList) { city ->
            if (city.weather == null) {
                repo.loadWeather(city)
            }

            CityItem(city = city, onClick= { ... }, onClose = { ... })
        }
    }
}
```

Esse código dispara a carga das condições climáticas da cidade somente quando a cidade é exibida na lista e ainda não contém essas informações.

Passo 12: Ainda em `ListPage`, modifique `CityItem` como mostrado abaixo:

```
@Composable
fun CityItem(...) {
    Row(...) {
        ...
        Column(modifier = modifier.weight(1f)) {
            Text(modifier = Modifier,
                text = city.name,
                fontSize = 24.sp)
            Text(modifier = Modifier,
                text = city.weather?.desc?:"carregando...",
                fontSize = 16.sp)
        }
        ...
    }
}
```

Passo 13: Em `MapPage`, modifique o código da chamada ao `GoogleMap()`:

```
GoogleMap ( ... ) {
    viewModel.cities.forEach {
        if (it.location != null) {
            Marker( state = MarkerState(position = it.location!!),
                title = it.name,
                snippet = it.weather?.desc?:"Carregando...")
        }
    }
}
```

Essa mudança faz com o que as condições climáticas atuais pareçam no *snippet* ao clicar em um marcador no mapa.

Passo 14: Rode e teste o aplicativo. Se estiver tudo certo, faça um novo *commit*.

Parte 2: Mostrando a previsão do tempo ao selecionar uma cidade da lista

Passo 1: Adicione as seguintes classes no pacote `api`:

```
data class APIWeatherForecast (
    var location: APILocation? = null,
    var current: APIWeatherForecast? = null,
    var forecast: APIForecast? = null
)

data class APIForecast (
    var forecastday: List<APIForecastDay>? = null
)

data class APIForecastDay (
    var date: String? = null,
    var day: APIWeather? = null
)
```

Passo 2: Adicione a função abaixo a declaração de `service.WeatherForecastAPI`:

```
    @GET("forecast.json?key=$API_KEY&days=10&lang=pt")
    fun forecast(@Query("q") name: String): Call<APIWeatherForecast?>
```

Esse *endpoint* aciona a API de previsão do tempo para os próximos 10 dias.

Passo 3: Em `service.WeatherService`, adicione a função abaixo:

```
fun getForecast(name: String, onResponse : (APIWeatherForecast?) -> Unit) {
    val call: Call<APIWeatherForecast?> = weatherAPI.forecast(name)
    enqueue(call) { onResponse.invoke(it) }
}
```

Essa função aciona o novo *endpoint* da API.

Passo 4: Crie a classe `model.Forecast`:

```
data class Forecast (
    val date: String,
    val weather: String,
    val tempMin: Double,
    val tempMax: Double,
    val imgUrl: String,
)
```

Passo 5: Modifique `model.City` para incluir o atributo do tipo `model.Forecast`.

Passo 6: Em `repo.Repository`, adicione a função a seguir.

```
fun loadForecast(city : City) {
    weatherService.getForecast(city.name) { result ->
        city.forecast = result?.forecast?.forecastday?.map {
            Forecast(
                date = it.date?:"00-00-0000",
                weather = it.day?.condition?.text?:"Erro carregando!",
                tempMin = it.day?.mintemp_c?:-1.0,
                tempMax = it.day?.maxtemp_c?:-1.0,
                imgUrl = ("https:" + it.day?.condition?.icon)
            )
        }
        listener.onCityUpdated(city)
    }
}
```

Essa função carrega a previsão do tempo na cidade e aciona `onCityUpdate()`.

Passo 7: Em `MainViewModel`, adicione a propriedade `city` como abaixo:

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

    private var _city = mutableStateOf<City?>(null)
    var city: City?
        get() = _city.value
        set(tmp) { _city = mutableStateOf(tmp?.copy()) }

    ...

    override fun onCityUpdated(city: City) {
        _cities.remove(city.name)
        _cities[city.name] = city.copy()

        if (_city.value?.name == city.name) {
            _city.value = city.copy()
        }
    }
}
```

Essa propriedade registra a cidade atualmente selecionada na lista, cuja previsão será exibida na página *Home*.

Passo 8: Atualize `HomePage` para conter o código abaixo:

```
@Composable
fun HomePage(...) {
    Column {
        Row {
            Icon(
                imageVector = Icons.Filled.AccountBox,
                contentDescription = "Localized description",
                modifier = Modifier.size(130.dp)
            )
            val format = DecimalFormat("#.0")

            Column {
                Spacer(modifier = Modifier.size(20.dp))
                Text(text = viewModel.city?.name?:"Selecione uma cidade...",
                     fontSize = 24.sp)
                Spacer(modifier = Modifier.size(10.dp))
                Text(text = viewModel.city?.weather?.desc?:"...",
                      fontSize = 20.sp)
                Spacer(modifier = Modifier.size(10.dp))
                Text(text = "Temp: " + viewModel.city?.weather?.temp + "°C",
                      fontSize = 20.sp)
            }
        }

        if (viewModel.city == null ||
            viewModel.city!!.forecast == null) return

        LazyColumn {
            items(viewModel.city!!.forecast!!) { forecast ->
                ForecastItem(forecast, onClick = { }, modifier = modifier )
            }
        }
    }
}
```

Nesse código novo, estamos exibindo as condições climáticas atuais da cidade selecionada (`viewModel.city`) no topo, seguida de uma lista de previsões para os próximos dias (próximo passo).

Passo 9: Crie o composable `ForecastItem` que será usado na `HomePage`:

```
@Composable
fun ForecastItem(
    forecast: Forecast,
    onClick: (Forecast) -> Unit,
    modifier: Modifier = Modifier
) {
    val format = DecimalFormat("#.0")
    val tempMin = format.format(forecast.tempMin)
    val tempMax = format.format(forecast.tempMax)

    Row(
        modifier = modifier.fillMaxWidth().padding(8.dp)
            .clickable( onClick = { onClick(forecast) }),
        verticalAlignment = Alignment.CenterVertically
    ) {
        Icon( imageVector = Icons.Filled.LocationOn,
              contentDescription = "Localized description",
              modifier = Modifier.size(40.dp) )
        Spacer(modifier = Modifier.size(12.dp))
        Column {
            Text(modifier = Modifier, text = forecast.weather, fontSize = 20.sp)
            Row {
             Text(modifier = Modifier, text = forecast.date, fontSize = 16.sp)
             Spacer(modifier = Modifier.size(12.dp))
             Text(modifier = Modifier, text = "Min: $tempMin°C", fontSize = 14.sp)
             Spacer(modifier = Modifier.size(12.dp))
             Text(modifier = Modifier, text = "Max: $tempMax°C", fontSize = 14.sp)
            }
        }

    }
}
```

Esse *composable* representa um item na lista de previsões, contendo data da previsão, temperaturas mínima e máxima, e um texto descritivo. Estamos usando um *placeholder* para as imagens de previsão de tempo.

Passo 10: Em `ListPage`, adicione o parâmetro `navCtrl` como abaixo:

```
@Composable
fun ListPage(
    modifier: Modifier = Modifier,
    viewModel: MainViewModel,
    context: Context,
    navCtrl: NavHostController
) {
   ...
}
```

ATENÇÃO: Modifique `ui.nav.MainNavHost` passando o parâmetro adequado a esse *composable*.

Passo 11: Em `ListPage`, modifique o `onClick` do Item como abaixo:

```
@Composable
fun ListPage( ...,
    navControl : NavHostController
) {
    val activity = LocalContext.current as? Activity
    val cityList = viewModel.cities
    LazyColumn ( ... ) {
        items(cityList) { city ->
            if (city.weather == null) {
                repo.loadWeather(city)
            }
            CityItem(city = city, onClick= {
                viewModel.city = city
                repo.loadForecast(city)
                navController.navigate(BottomNavItem.HomePage.route) {
                    navController.graph.startDestinationRoute?.let {
                        popUpTo(it) {  saveState = true  }
                        restoreState = true
                    }
                    launchSingleTop = true
                }
            }, onClose = { ... })
        }
    }
}
```

Esse código faz a cidade clicada ser selecionada no `MainViewModel`, dispara a busca pela previsão climática e navega para `HomePage`.

Passo 12: Rode e teste a aplicação. Se estiver tudo certo, dê um novo *commit*.

Nesse ponto, ao clicar numa cidade da lista de favoritas, a previsão do tempo para essa cidade será carregada e exibida na página *Home*.