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: MAPAS E LOCALIZAÇÃO

Prática 04

ATENÇÃO: Continuação da Prática 03; use controle de versões (Git).

Parte 1: Preparação e Configuração

Passo 1: Obtenha a chave de acesso a API do Google.

> **ATENÇÃO:** Durante o procedimento a seguir, o Google pode pedir para criar um conta para faturamento, o que requer um número de cartão de crédito. Cancele se necessário, sem fornecer o número, e verifique se as configurações realizadas foram salvas.
>
> Visite a página do console de desenvolvedores do Google (esteja logado). Crie um projeto chamado "WeatherApp" (a criação pode demorar um pouco). No menu à esquerda, em APIs e serviços, ative a API de Mapas (se não estiver ativada): na lista abaixo, selecione *Maps SDK for Android* e depois Ativar.
>
> No menu, APIs > Credenciais: + Criar credenciais (barra acima) > Chave de API. Sua chave será parecida com: AIzaSyDwSRoRTXZ6btXXXXXXXXXXXXXXX.
> (**Opcional:** restrinja o uso da chave a aplicativos Android. (Clique na chave)).

Passo 2: Verifique se o Google Play Services está instalado no Android Studio; instale se necessário:

> Abra o *SDK Manager* no menu *Tools* (Ferramentas), vá na aba *SDK Tools* e selecione Google Play services. Clique *Apply* ou *OK* e siga as instruções de instalação.

Passo 3: Verifique se o dispositivo usado para testes tem suporte ao *Google Play Services* (isto é, tem a *Play Store*).

> Caso o dispositivo emulado não tenha, será necessário criar um novo dispositivo usando uma versão do Android suporte a *Play Store*.

Passo 4: Adicione uma linha no arquivo **local.properties** (visão Gradle Scripts no projeto) como abaixo usando sua chave:

```
MAPS_API_KEY= "AIzaSyCDgBOqYlz6cXXXXXXXXXXXXXXXXXXXX"
```

> **ATENÇÃO:** O arquivo **local.properties** por *default* não é commitado no git (.gitignore). Isto é de propósito, para evitar que a chave seja publicada por acidente num repositório público (e.g., GitHub).

Passo 5: Adicione as permissões necessárias no **AndroidManifest.xml**:

> Adicione as linhas a seguir antes da *tag* `<application>` (caso não existam):

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

A permissão para localização “fina” ou precisa (`ACCESS_FINE_LOCATION`) usa GPS, Wi-fi e rede celular para determinar a posição do usuário. O mapa em si não precisa dessa permissão, mas ela é necessária para acessar a localização atual. A localização aproximada (`ACCESS_COARSE_LOCATION`) usa apenas Wi-fi e rede celular e tem a precisão no nível de quarteirão.

Passo 6: Ainda no **AndroidManifest.xml**, adicionar o conteúdo a seguir em `<Application>`:

```
   <meta-data
       android:name="com.google.android.geo.API_KEY"
       android:value="${MAPS_API_KEY}" />
```

Passo 7: Adicione o trecho abaixo ao começo do **build.script.kts** do projeto WeatherApp.

```
buildscript {
    repositories {
        google()
        mavenCentral()
    }
    dependencies {
        classpath ("com.google.android.libraries.mapsplatform.secrets-gradle-
plugin:secrets-gradle-plugin:2.0.1")
    }
}

plugins { ... }
```

Passo 8: Modifique o **build.script.kts** do módulo *app* com os plugins e as dependências novas:

```
plugins {
    ...

    // Deve vir depois dos outros plugins (Bug)
    id ("com.google.android.libraries.mapsplatform.secrets-gradle-plugin")
}

...

dependencies {

    // Google maps
    implementation("com.google.android.gms:play-services-maps:18.2.0")
    implementation("com.google.android.gms:play-services-location:21.2.0")

    // Google maps for compose
    implementation("com.google.maps.android:maps-compose:2.8.0")

    ...
}
```

Esse dois últimos passos configuram as bibliotecas necessárias (`play-services-maps`, etc.) mas também permitem esconder a chave da API de mapas no arquivo **local.properties** e importá-la no manifesto (usando `secrets-gradle-plugin`).

Passo 9: Sincronize e rode o aplicativo.

Esse passo é para verificar se nada foi quebrado. Não deve haver mudança no comportamento. Faça um novo commit se estiver tudo certo.

Parte 2: Refatoração e Permissões

Passo 1: Em **MainViewModel.kt**, faça as seguintes refatorações:

Adicione o atributo `location` em `FavorityCity`:

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

Modifique o `MainViewModel` para ficar como abaixo:

```
class MainViewModel : ViewModel() {
    private val _cities = getCities().toMutableStateList()
    val cities : List<City>
        get() = _cities

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

    fun add(city: String, location: LatLng? = null) {
        _cities.add(City(city, "Carregando clima...", location))
    }
}
```

As mudanças incluem o esconder a lista de cidades (`_cities`) atrás de uma propriedade *read-only* (`cities`), para evitar mudanças indesejadas, e o parâmetro `location` ao adicionar uma nova cidade.

Passo 2: Refatore `HomePage` de forma a ter os parâmetros a seguir:

```
fun HomePage(
    modifier: Modifier = Modifier,
    viewModel: MainViewModel,
    context: Context
) {  ...  }
```

Faça o mesmo em `ListPage` e `MapPage`. Em `ListPage`, a variável activity deixa de ser necessária, com `context` a substituindo onde necessário. (A inclusão de parâmetros faz o `@Preview` parar de funcionar; opcionalmente, crie uma versão sem parâmetros desses `@Composables` para visualizar o preview).

Passo 3: Modifique `MainNavHost` de forma a receber um parâmetro do tipo Context e passe esse parâmetro para as chamadas de `HomePage`, `ListPage` e `MapPage`.

Também modifique as chamadas a `HomePage` e `MapPage` para incluir o parâmetro `ViewModel`, assim como já existe para `ListPage`.

Passo 4: Na classe `MainActivity`, crie as variável abaixo dentro de `setContent(){...}`, depois das variáveis existentes:

```
    val context = LocalContext.current
    val currentRoute = navController.currentBackStackEntryAsState()
    val showButton = currentRoute.value?.destination?.route !=
                                    BottomNavItem.MapPage.route
    val launcher = rememberLauncherForActivityResult(contract =
            ActivityResultContracts.RequestPermission(), onResult = {} )
```

Passe `context` como parâmetro a chamada de `MainNavHost()`.

A variável `launcher` será usada para pedir a permissão de localização ao usuário.

Mova também a variável `viewModel` para fora (antes) de `setContent()`.

Passo 5: Em `MainActivity`, mude o código associado com `floatingActionButton`:

```
floatingActionButton = {
    if (showButton) {
        FloatingActionButton(onClick = { showDialog = true }) {
            Icon(Icons.Default.Add, contentDescription = "Adicionar")
        }
    }
}
```

Esse código, junto com as variáveis `currentRoute` e `showButton`, faz com que o botão flutuante não apareça no página do mapa (`MapPage`).

Passo 6: Ainda em `MainActivity`, adicione a linha abaixo para chamar o diálogo de pedido de permissão.

```
Scaffold(...) { innerPadding ->
    Box(modifier = Modifier.padding(innerPadding)) {

        launcher.launch(Manifest.permission.ACCESS_FINE_LOCATION)

        MainNavHost(...)
    }
}
```

Caso a permissão já tenha sido dada pelo usuário (ou rejeitada permanentemente), o diálogo não aparecerá. Em caso de problemas nessa linha, importe explicitamente o `Manifest` (há várias classes com esse nome, incluindo uma no próprio pacote do projeto, o que causa problemas na compilação):

```
import android.Manifest
```

Passo 7: Rode o aplicativo.

A única mudança visível será o pedido de permissão. Faça um novo commit.

Parte 3: Adicionando e Testando o Mapa

Passo 1: Em **MapPage.kt**, substitua o `Column()` (com `Text()` dentro) com o trecho abaixo:

```
GoogleMap (modifier = Modifier.fillMaxSize()) {}
```

Passo 2: Declare as localizações abaixo dentro de `MapPage(){...}`, antes de `GoogleMap()`:

```
val recife = LatLng(-8.05, -34.9)
val caruaru = LatLng(-8.27, -35.98)
val joaopessoa = LatLng(-7.12, -34.84)
```

Passo 3: Para cada uma da localizações acima, crie manualmente marcadores (pinos) no mapa como no exemplo abaixo (escolhas cores diferentes para cada um):

```
GoogleMap( modifier = modifier.fillMaxSize()) {

    Marker(
        state = MarkerState(position = recife),
        title = "Recife",
        snippet = "Marcador em Recife",
        icon = BitmapDescriptorFactory.defaultMarker(
                   BitmapDescriptorFactory.HUE_BLUE)
    )

    ...
}
```

Passo 4: Rode e teste o aplicativo.

Navegue para a página do mapa e veja se funciona. Navegue para os marcadores e clique neles.

Passo 5: Modifique o código para ficar com abaixo (não é preciso retirar os marcadores colocados anteriormente).

```
GoogleMap(
    modifier = modifier.fillMaxSize(),
    onMapClick = { viewModel.add("Nova cidade", location = it) }
) {

    viewModel.cities.forEach {
        if (it.location != null) {
            Marker( state = MarkerState(position = it.location!!),
                title = it.name, snippet = "${it.location}")
        }
    }

    ...
}
```

Esse código adiciona marcadores para as cidades favoritas no mapa, mas somente aquelas que possuem uma localização definida. O clique no mapa (`onMapClick = {...}`) é configurado para adicionar uma nova cidade na lista de favoritas usando a localização do clique. Veja que a nova cidade aparece ao final da lista em `ListPage`, e quando removida, também some do mapa.

Passo 6: Modifique o corpo da função `MapPage()` como abaixo:

```
...
val camPosState = rememberCameraPositionState ()
...
GoogleMap( ...,
    cameraPositionState = camPosState
) { ... }
```

Essa modificação salva a posição da câmera do mapa (onde estamos olhando, nível de zoom, etc.), de forma que o mapa não retorna à posição *default* se sairmos e voltarmos a ele.

Passo 7: Rode e teste o aplicativo.

Navegue para o mapa e teste. Adicione marcadores e veja a lista de cidades favoritas. Se estiver tudo certo, faça um novo *commit*.

Parte 4: Trabalhando com a localização do usuário

Passo 1: Adicione a variável `hasLocationPermission` antes da chamada a `GoogleMap(...)`:

```
val hasLocationPermission by remember {
    mutableStateOf(
        ContextCompat.checkSelfPermission(context,
                Manifest.permission.ACCESS_FINE_LOCATION) ==
                        PackageManager.PERMISSION_GRANTED
    )
}
```

Essa variável verifica se temos a permissão de acessar a localização precisa do usuário.

Passo 2: Modifique novamente a chamada a `GoogleMap()` como abaixo:

```
GoogleMap( ..., // <- parâmetros anteriores aqui
    properties = MapProperties(isMyLocationEnabled = hasLocationPermission),
    uiSettings = MapUiSettings(myLocationButtonEnabled = true)
) { ... }
```

Essa modificação habilita o ponto azul com posição atual do usuário e botão que leva a câmera do mapa para essa posição.

Passo 3: Rode e teste o aplicativo.

Navegue para a página do mapa e veja se funciona. No emulador, a posição atual do usuário pode ser na sede do Google na California (certifique-se que a localização está ativada no celular/emulador).