CSCI 448 – Lab 10B
Wednesday, March 29, 2023
LAB IS DUE BY **Friday, April 07, 2023 11:59 PM**!!

Lab10B is more straightforward than Lab10A was, but setting up the Map and interacting with it can be error prone when requesting an API key.

Step 0 – Get Google Maps API Access

Part 0.I – Add Google Maps Dependency

First, in the project level `build.gradle` file, add the following dependency block inside of the `buildscript` block:

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

Now, in the `build.gradle` file, add the following plugin:

```
plugins {
   ...
   id 'com.google.android.libraries.mapsplatform.secrets-gradle-plugin'
}
```

And add the dependency for Google Maps.

```
dependencies {
   ...
   implementation 'com.google.android.gms:play-services-maps:18.1.0'
   implementation 'com.google.maps.android:maps-compose:2.1.0'
   ...
}
```

Part 0.II – Get an API Key

This could be the most error prone part of this entire lab.

We're going to follow the instructions at https://developers.google.com/maps/documentation/android-sdk/start#get-key for the "`slightly less fast way`".

Follow the link to the `Google Cloud Platform Console` https://console.cloud.google.com/ .

Once logged in, here are the steps to follow to generate an API key (listed here as well https://developers.google.com/maps/documentation/android-sdk/get-api-key):

1. Create a new Project named Geolocatr
2. Go to APIs & Services
3. Go to Credentials

4. Click Create Credentials > API key
5. Click Save
6. Copy the API Key

Really, we'd want to restrict the key – but we're trying to avoid any potentials for problems. Additionally, we won't have access to all Map and Google Cloud features because we have not set up a billing account for the full free trial.

Part 0.III – Add the Google Maps API Key to the Manifest

Now, in your `local.properties` file add a new line to the end with

`MAPS_API_KEY=#yourAPIkeygoeshere#`

And paste in your API key in place of `#yourAPIkeygoeshere#`

Jump over to the manifest where we'll add the key. Inside of the application tag, add a meta-data tag

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

With a little luck, everything worked and we can move on. We'll find out if it didn't work shortly.

Step 1 – Display the Map

We only need to make a few minor changes to `LocationScreen` in order to display the map.

First, remember the `cameraPositionState` that will be used to move around the map.

```
@Composable
fun LocationScreen(location: Location?,
                   isLocationAvailable: Boolean,
                   onGetLocation: () -> Unit,
                   address: String) {

  val cameraPositionState = rememberCameraPositionState {
    position = CameraPosition.fromLatLngZoom(LatLng(0,0, 0.0), 0f)
  }

  Column {
    ...
    // TODO map will go here at the end
  }
}
```

Now we can fill in the TODO.

We'll first create the map using the `GoogleMap` composable.

```
GoogleMap(
  modifier = Modifier.fillMaxSize(),
  cameraPositionState = cameraPositionState
) {
  // marker goes here
}
```

And now, we'll only display a marker if our location is not null (otherwise we'd see the marker in the ocean off the coast of Africa). We'll set the marker to be at our location position and when clicked have it display the address and lat/long.

```
if(location != null) {
  val markerState = MarkerState().apply {
    position = LatLng(location.latitude, location.longitude)
  }
  Marker(
    state = markerState,
    title = address,
    snippet = "${location.latitude} / ${location.longitude}"
  )
}
```

And that's it

Run your app, press the location button, and you should see a marker added to the map at the location. We now want the map to auto zoom to that location.

Step 2 – Go to Location

Add a new resource file of type `Values` named `dimens.xml`. We will put into this file dimensions that should be used for various view components.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <dimen name="map_inset_padding">100dp</dimen>
</resources>
```

We will use the `map_inset_padding` to provide a `100dp` margin around our location when we zoom in.

In `LocationScreen`, before calling the `Column` composable add a `LaunchedEffect` to monitor whenever the location changes - we will want to update the camera. The camera movement is given below.

```
val context = LocalContext.current
LaunchedEffect(location) {
  if(location != null) {
    // include all points that should be within the bounds of the zoom
    // convex hull
    val bounds = LatLngBounds.Builder()
        .include(LatLng(location.latitude, location.longitude)
        .build()
    // add padding
    val padding = context.resources
                         .getDimensionPixelSize(R.dimen.map_inset_padding)
    // create a camera update to smoothly move the map view
    val cameraUpdate = CameraUpdateFactory.newLatLngBounds(bounds, padding)
    // move our camera!
    cameraPositionState.animate(cameraUpdate)
  }
}
```

Run your program and once the map has loaded, press the get location button. Zoom zoom.

Step 3 - Submission

Be sure to be working with a fresh install. Uninstall the app from the device if necessary. Submit a video with the following features for the final sign off:

- Open the app
- Click the location button
- Give location permission
- Lat/Long, Address, and Map update (map has marker and zooms to marker)
- Click the marker to see the title

Please name this video file `<your_user_name>_L10`. For instance, my submission would be `jpaone_L10.webm`.

LAB IS DUE BY **Friday, April 07, 2023 11:59 PM**!!