
ubique

VIScon 2024

Android Map Workshop

Maps on Android with Open Mobile Maps

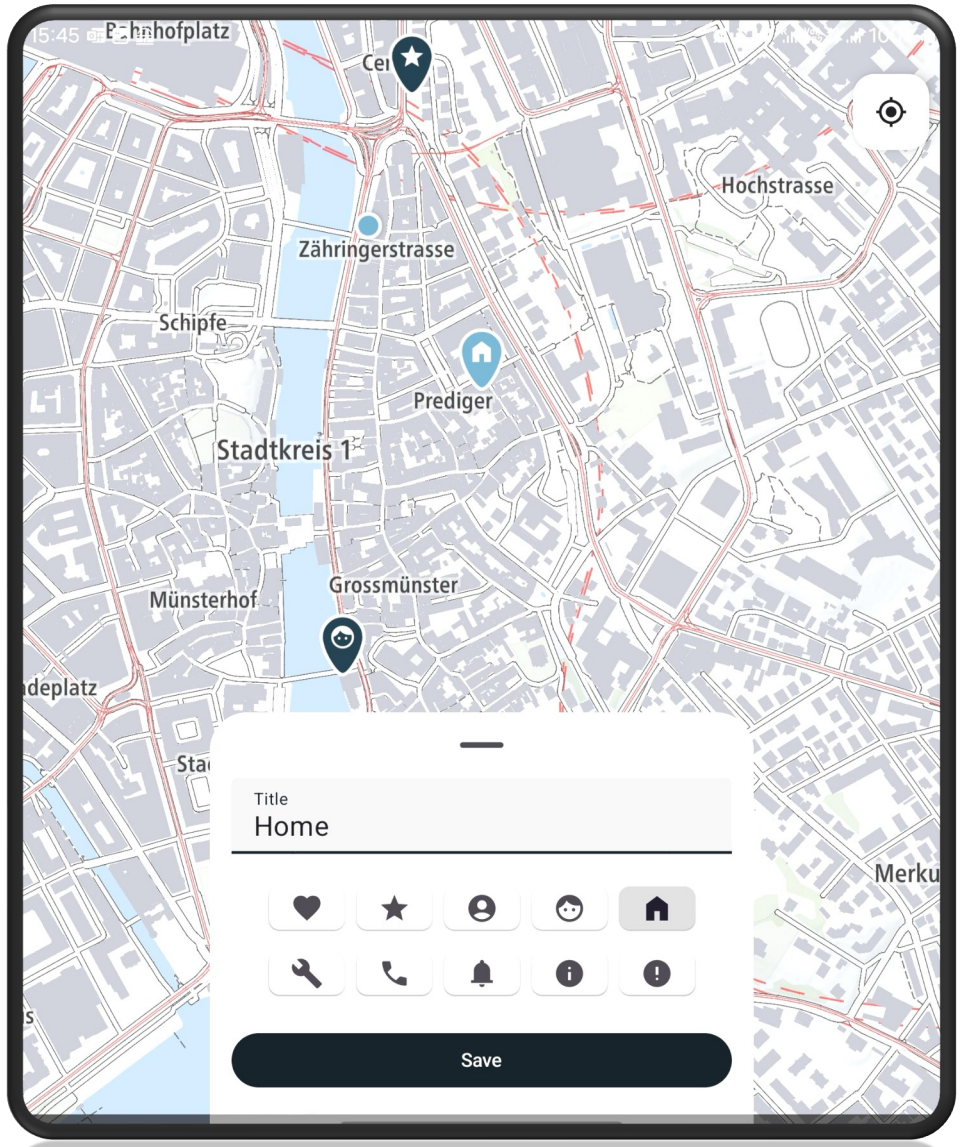
ubique

Goal

App using Map SDK: **Open Mobile Maps**
https://openmobilemaps.io/

Workshop content

1. App and SDK setup
2. Load and show map
3. Style map
4. Map interaction
5. Show data on map
6. GPS position



Open Mobile Maps

Collection of open-source SDKs for maps Android and iOS: **maps-core**, **layer-gps**
Convenience wrapper and default Swisstopo layers: **Open Swiss Maps**
Developed by Ubique

On Android: Integrated via MapView
Shared code in C++, customizable by using exposed interfaces
Different layer types: line, polygon, icon, tiled raster or tiled vector
Handles:

- Rendering
- Coordinate system conversions
- Map interaction and camera
- Networking (via OkHttp)

Used in mobile apps of: swisstopo, SwitzerlandMobility, SAC-CAS, ...

Kotlin

- Official language for Android Development
- JVM-based & fully **interoperable with Java**
- **Multiplatform** (JVM, Android, iOS, JavaScript, Web, Native)
- Open Source by JetBrains: https://kotlinlang.org/

Kotlin

Modern & Concise & Safe

- Data classes
- Null safety
- Type inference
- Extension functions

Data classes

```
data class Garden(val appleTree: AppleTree?)
```

Null safety

```
val appleTree: AppleTree? = garden.appleTree
val apples: List<Apple> = appleTree?.fruits ?: emptyList()
```

Type inference

```
val appleTree = garden.appleTree
```

Extension functions

```
fun Garden.numberOfApples() = fruits.size
```

Kotlin

Modern & Concise & Safe

- Data classes
- Null safety
- Type inference
- Extension functions
- Coroutines
- Sealed classes
- Delegated properties

Coroutines

```
scope.launch(Dispatchers.IO) {
    // Do some async IO
}
```

ViewModel & Flows

State host

ViewModel.kt

```
private val gardenMutable = MutableStateFlow<Garden?>(null)
val garden = gardenMutable.asStateFlow()
```

State collector

Composable.kt

```
val garden: State<Garden?> = viewModel.garden.collectAsState()
Log.v(tag, garden.value)
```

Project Structure

github.com/UbiqueInnovation/viscon24-android

Git repository

- Workshop **starting point: newest commit** on main
- Each commit is a sample solution for a workshop step
- **Prefixed commits** [WORKSHOP STEP NAME]

Android project

- Open in Android Studio
- Commit prefixes can be found as comments in the project (Ctrl + shift + f)

Setup Map

Add BestMapView (wrapper for Open Mobile Maps MapView):

App.kt

```
fun App(innerPadding: PaddingValues) {
  ...
  AndroidView(
    ...
  )
}
```

Setup Map

Setup map and specify a coordinate system (EPSG 2056):

BestMapView.kt

```
init {
    setupMap(MapConfig(CoordinateSystemFactory.getEpsg2056System()))
}
```

Setup Map

Register the view lifecycle to the map when available:

BestMapView.kt

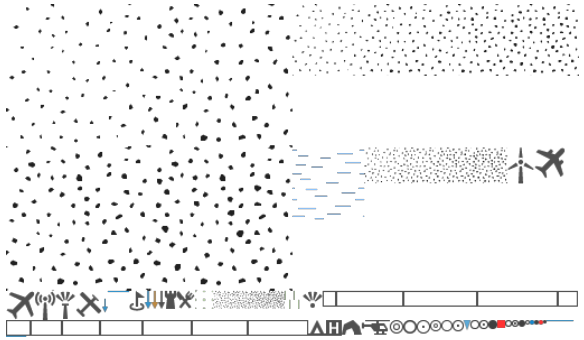
```
override fun onAttachedToWindow() {
    …
    registerLifecycle(lifecycleOwner.lifecycle)
}
```

Test your map by setting a background color:

BestMapView.kt

```
setBackgroundColor(Color(1.0f, 0.0f, 0.0f, 1.0f))
```

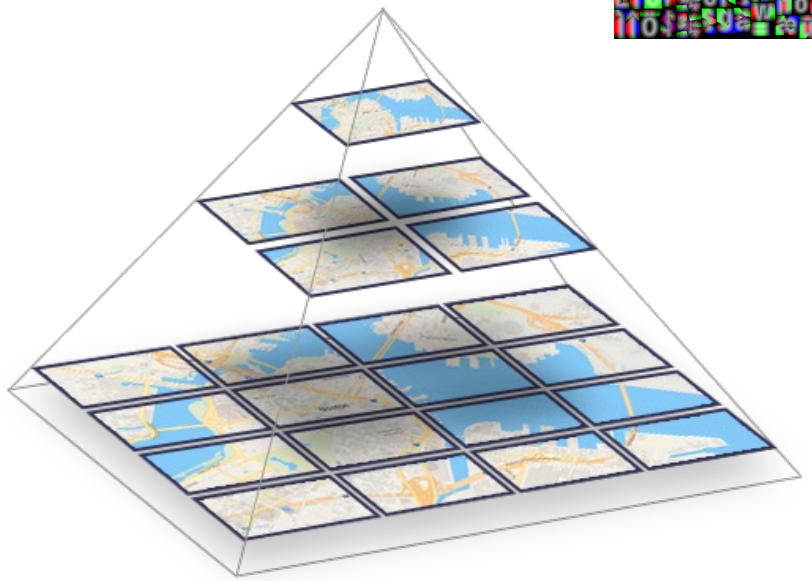
Tiled Vector Layers



Vector tile standard defined by mapbox (style and vector tile format)
In development in Open Mobile Maps
Style and other data needed specified in style.json (styling, data sources, sprites, ...)
Data:

- Layered geometry (points, lines and polygons) and metadata
- Tiled (usually EPSG 3857)
- Provided as protobuf files

Vector styles may contain raster or geojson data sources



Vector Layer style.json

simple_base_map_style.json

```
{
  ...
  "sources": { ... },
  "sprite": "https://vectortiles.geo.admin.ch/styles/ch.swisstopo...",
  "glyphs": "https://vectortiles.geo.admin.ch/fonts/{fontstack}/{range}.pbf",
  ...
  "layers": [ ]
}
```

Vector Layer style.json

simple_base_map_style.json

```
"sources": {
  "base_v1.0.0": {
    "url": "https://vectortiles.geo.admin.ch/tiles/ch.swisstopo.base.vt/v1.0.0/tiles.json",
    "type": "vector"
  },
  "swisstopo_relief_mono": {
    "type": "raster",
    "tiles": [ "https://wmts.geo ... monodirektional/default/current/3857/{z}/{x}/{y}.png” ],
    "minzoom": 0,
    "maxzoom": 19,
    "scheme": "xyz"
  }
},
```

Vector Layer style.json

simple_base_map_style.json

```
{
  "id": "water_polygon",
  "type": "fill",                                    // or raster, line, symbol
  "source": "base_v1.0.0",                           // data/tile source
  "source-layer": "water",                           // layer name in vector tiles
  "layout": { "visibility": "visible” },             // "structural" properties
  "paint": { "fill-color": "rgb(199, 224, 245)” }    // "visual" properties
},
```

Vector Layer style.json

simple_base_map_style.json

```
{
  "id": "place_city", ... "layout": {
    "text-size": [
        "interpolate",
        ["exponential", 1.2],    // interpolation curve properties (linear, exponential, ...)
        ["zoom"],                // input value
        1,                       // first x-step
        12,                      // value (text size) at first step
        16,                      // second x-step
        48                       // value at second step
      ], ... }
  }
}
```

Add Vector Layer

Load simple base map style.json from assets

BestMapView.kt

```
fun addVectorLayer() {
  ...
  val styleJson = context.assets.open("map/styles/simple_base_map_style.json")
    .bufferedReader(StandardCharsets.UTF_8).use { it.readText() }
  ...
}
```

Add Vector Layer

Create the vector layer (need the loaders provided in the BestMapView):

BestMapView.kt

```
fun addVectorLayer() {
  ...
  val vectorLayer: Tiled2dMapVectorLayerInterface? = Tiled2dMapVectorLayerInterface.create(
    layerName = "SwisstopoBaseMap",
    styleJson = styleJson,
    dataLoader = dataLoader,
    fontLoader = fontLoader
  )
  ...
}
```

Add Vector Layer

Add the layer to the map:

BestMapView.kt

```
fun addVectorLayer() {
  ...
  if (vectorLayer != null) {
     insertLayerAt(vectorLayer.asLayerInterface(), BASE_LAYER_INDEX)
     ...
  }
  ...
}
```

No initial camera adjustment: remember to **zoom out**!

Initial Camera Position

Set a useful initial position and zoom (Hint: CoordinateSystemIdentifiers with EPSG 4326):

App.kt

```
if (mapViewState?.value == MapViewState.RESUMED && !cameraPositionInitialized.value) {
    LaunchedEffect(Unit) {
        mapView.value?.mapInterface?.getCamera()?.moveToCenterPositionZoom(
            centerPosition = Coord(
                systemIdentifier = CoordinateSystemIdentifiers.EPSG4326(),
                x = 8.5404656,      // longitude
                y = 47.377858,      // latitude
                z = 0.0
            ), zoom = 30000.0,
            animated = false
        ) ...
}}
```

Change map style

Change the style to the more complex base_map_style.json:

BestMapView.kt

```
fun addVectorLayer() {
  val styleJson = context.assets.open("map/styles/base_map_style.json")
    .bufferedReader(StandardCharsets.UTF_8).use { it.readText() }
  ...
}
```

Maybe explore the new style.json a bit.

Icon Layer

Create the icon layer:

BestMapView.kt

```
private val iconLayer = IconLayerInterface.create()
```

Icon Layer

Add the icon layer to the map:

BestMapView.kt

```
private fun addIconLayer() {
  ...
  insertLayerAt(iconLayer.asLayerInterface(), ICON_LAYER_INDEX)
  ...
}
```

Icon Layer

Test your layer with the test icon provided with the commented sample code:

BestMapView.kt

```
private fun addIconLayer() {
  ...
  val testIcon = IconFactory.createIcon(
    identifier = "Test-Icon",
    coordinate = Coord(CoordinateSystemIdentifiers.EPSG4326(), 8.54378, 47.37568, 0.0),
    texture = BitmapTextureHolder(
      drawable = requireNotNull(ContextCompat.getDrawable(context, R.drawable.ic_star))
    ),
    iconSize = Vec2F(84f, 84f),
    scaleType = IconType.INVARIANT,
    blendMode = BlendMode.NORMAL
  )
  iconLayer.add(testIcon)
}
```

Clickable Icon Layer

Make the icon layer clickable and forward clicks to the map view's clickListener:

BestMapView.kt

```
private fun addIconLayer() {
  iconLayer.apply {
      setLayerClickable(true)
      setCallbackHandler(object : IconLayerCallbackInterface() {
       override fun onClickConfirmed(icons: ArrayList<IconInfoInterface>): Boolean {
          clickListener?.onClickIcons(icons.map { it.getIdentifier() })
          return true
       }
       override fun onLongPress(icons: ArrayList<IconInfoInterface>): Boolean {
          return false // not used
       }
    })
  } ...
}
```

Clickable Icon Layer

In the click listener of the BestMapView, forward the clicked icons into the ViewModel and test the callback by displaying a toast:

App.kt

```
AndroidView( ...
    BestMapView(context).also {  ...
           override fun onClickIcons(icons: List<String>) {
              mainViewModel.
              GlobalScope.launch(Dispatchers.Main) {
                Toast.makeText(context, "Icons clicked: $icons", Toast.LENGTH_SHORT).show()
              }
              ...
           } ...
    }
)
```

Clickable Icon Layer

Complete the setIcons method of the BestMapView:

BestMapView.kt

```
fun setIcons(icons: List<IconInfoInterface>) {

}
```

Add New POI

React to long press on map. Get interaction coordinate from screen position:

BestMapView.kt

```
private fun addTouchListener() { ...
          override fun onLongPress(posScreen: Vec2F): Boolean {
             val coordinate =
               requireMapInterface().getCamera().coordFromScreenPosition(posScreen)
             ...
             return true
       }
    })
}
```

Add New POI

Forward coordinate to the map view's click listener:

BestMapView.kt

```
private fun addTouchListener() { ...
          override fun onLongPress(posScreen: Vec2F): Boolean {
              ...
              clickListener?.onLongPressBaseMap(coordinate4326)
              ...
          }
      })
}
```

Add New POI

In the click listener of the BestMapView, forward the long press to the ViewModel and clear clicked icon identifiers / the long press coordinate when an icon is clicked:

App.kt

```
AndroidView( ...
    BestMapView(context).also {  ...
       override fun onLongPressBaseMap(coordinate: Coord) {
          mainViewModel.clearClickedIcons()
          mainViewModel.onLongPress(coordinate)
       }
       override fun onClickIcons(icons: List<String>) {
          mainViewModel.clearLongPressCoordinates()
          ...
       } ...
    }
)
```

Add New POI

For testing purposes, add a pin via ViewModel on each collected long press (and clear the coordinates):

MapOverlay.kt

```
longPressCoordinate.value?.let { pinCoordinate ->
    mainViewModel.clearLongPressCoordinates()
    mainViewModel.addPin("Poi", null, pinCoordinate)
    ...
}
```

The pin is added to the database referenced in the ViewModel.

Add New POI

Pins from the database are loaded and transformed in the ViewModel. Add the collected pin icons to the map:

App.kt

```
val icons = mainViewModel.icons.collectAsState(emptyList())
mapView.value?.setIcons(icons.value)
```

Remember to remove the test code for creating a pin.

New POI Sheet

New long press behavior: Open bottom sheet to customize Pin

Propagate changes from sheet

MapOverlay.kt

```
mainViewModel.setNewIconProperties(
    Pin(id = "__NEW_PIN__", title = "", iconName = tempIcon, coord = pinCoordinate)
)
```

Insert into db on save

MapOverlay.kt

```
mainViewModel.clearLongPressCoordinates()
mainViewModel.addPin(title, icon, pinCoordinate)
mainViewModel.clearNewPin()
```

POI Content Sheet

Design your own POI sheet content in the `PoiBottomSheetContent.kt`.
The data to display is in the `pin: Pin`.

Add a button to your sheet to delete a pin from the database

MapOverlay.kt

```
mainViewModel.removePin(pin.id)
```

Bonus: GPS Layer

Add a default location provider:

BestMapView.kt

```
private val locationProvider = GpsProviderType.GOOGLE_FUSED.getProvider(context)
```

And create the gps layer:

```
private val gpsLayer = GpsLayer(context, GpsStyleInfoFactory.createDefaultStyle(context),
locationProvider).apply {
    setMode(GpsMode.STANDARD)
    setHeadingEnabled(false)
}
```

Bonus: GPS Layer

Add the layer to the map:

BestMapView.kt

```
private fun addGpsLayer() {
    insertLayerAt(gpsLayer.asLayerInterface(), GPS_LAYER_INDEX)
}
```

Bonus: GPS Layer

Register the gps layer to the map view's lifecycle:

BestMapView.kt

```
override fun onAttachedToWindow() {
    ...
    gpsLayer.registerLifecycle(lifecycleOwner.lifecycle)
    ...
}
```

Bonus: GPS Layer

Forward the notification of the granted gps permission to the location provider

BestMapView.kt

```
fun notifyLocationPermissionGranted() {
    locationProvider.notifyLocationPermissionGranted()
}
```

Bonus: GPS Layer

Add the call to create a location button in the map overlay:

MapOverlay.kt

```
Box(
    modifier = modifier.padding(
        PaddingValues(0.dp, innerPadding.calculateTopPadding(), 0.dp,
           innerPadding.calculateBottomPadding())
    )
) {
    GpsButton(mapView)
}
```

Handles the check for the location permission.

Bonus: GPS Layer

Add moving to the current location on click of the GpsButton:

MapOverlay.kt

```
private fun BoxScope.GpsButton(mapView: BestMapView?) { ...
  FloatingActionButton(
    onClick = {
      if (context.hasLocationPermission()) {
        mapView?.apply {
          currentLocation?.let {
            getCamera().moveToCenterPositionZoom(
              centerPosition = Coord(systemIdentifier = CoordinateSystemIdentifiers.EPSG4326(),
                x = it.longitude, y = it.latitude, z = it.altitude),
              zoom = 5000.0, animated = true)
          }
        }
      } ...
    } ...
```

Bonus: GPS Layer

Need to register with the location provider in the BestMapView to receive calls to onLocationUpdate:

BestMapView.kt

```
override fun onAttachedToWindow() {
    ...
    locationProvider.registerLocationUpdateListener(this)
}

override fun onDetachedFromWindow() {
    ...
    locationProvider.unregisterLocationUpdateListener(this)
}
```