**Sesión 4-03 Clase del 13 de febrero**

En esta sesión vamos a realizar una programación básica para obtener la ubicación del dispositivo. Para hacer esto en **Jetpack Compose**, necesitamos usar la API de ubicación de Android (**FusedLocationProviderClient**).

Puedes consultar información sobre servicios de localización o ubicación en la documentación oficial de **Android Developers** en el siguiente enlace:

<https://developer.android.com/training/location/index.html>

Un servicio de localización o de ubicación proporciona información sobre las coordenadas de posicionamiento de un dispositivo (latitud y longitud) y otros datos relacionados como altitud, precisión de la localización, etc.

*Aunque se puede obtener más información, el servicio de localización proporciona la información sobre el posicionamiento geográfico del dispositivo. El posicionamiento o ubicación se establece por la latitud (norte o sur respecto del ecuador de la tierra y comprendida entre -90.0 y +90.0) y la longitud (este u oeste respecto del meridiano de Greenwich y comprendida entre -180.0 y +180.0). También permite obtener la altitud medida en metros.*

1.- Por ahora, simplemente debemos diseñar un Composable **HomeView** dentro de **MainActivity** que tenga un **Button INICIAR\_LOCALIZACION** y dos Text para que se escriba en ellos la información sobre la ubicación actual. Debes crear en el composable dos variables de estado **textLatitud** y **textLongitud** destinadas a contener el valor que se escribirán en los dos **Text**.

2.- Debemos declarar en Manifest estos dos permisos de acceso a la ubicación:

ACCESS\_COARSE\_LOCATION (menor precisión, ubicación aproximada)

ACCESS\_FINE\_LOCATION (mayor precisión, ubicación precisa)

3.- Esos dos permisos son de los permisos “riesgosos”, permisos que comprometen la privacidad del usuario. Por ello, se debe meter código en la aplicación que controle que el usuario conoce esos permisos y los ha autorizado en la primera ejecución de la app.

3.1.- Añade la siguiente función para gestionar la autorización de los permisos en la app:

@Composable  
//La función recibirá una función onGranted

fun RequestLocationPermission(onGranted: () -> Unit) {  
 val context = *LocalContext*.current  
 //llamada a funcion para solicitar permisos  
 val permissionLauncher = rememberLauncherForActivityResult(  
 contract = ActivityResultContracts.RequestPermission()  
 ) **{** isGranted **->** //si el usuario ha dado permisos se ejecuta la funcion onGranted  
 if (isGranted) onGranted()  
 **}** //Codigo que se carga la primera vez que se inicia el composable  
 LaunchedEffect(Unit) **{** //Comprobamos si el usuario ha dado permisos  
 if (ContextCompat.checkSelfPermission(context, Manifest.permission.*ACCESS\_FINE\_LOCATION*) !=  
 PackageManager.*PERMISSION\_GRANTED*) {  
 //Si el usuario no ha dado permisos se solicitan  
 permissionLauncher.launch(Manifest.permission.*ACCESS\_FINE\_LOCATION*)  
 } else {  
 //Si el usuario ha dado permisos se ejecuta la funcion onGranted  
 onGranted()  
 }  
 **}**}

Si tienes error en **Manifest.permission.ACCESS\_FINE\_LOCATION**, debes añadir el **import andoid.Manifest**.

*No hace falta pedir el permiso ACCESS\_COARSE\_LOCATION si ya se pide el permiso ACCESS\_FINE\_LOCATION (es de más precisión y sirve para lo mismo).*

4.- Volvemos a editar **HomeView** para empezar a programar la gestión de la ubicación.

4.1.- Añadimos una variable para el contexto y otra de estado que nos permitirá acceder a la información de ubicación del dispositivo.

val context = *LocalContext*.current  
//cliente que se usa para acceder a la ubicación del dispositivo  
val fusedLocationClient = remember **{** LocationServices.getFusedLocationProviderClient(context) **}**

4.-2.- Creamos otra variable de estado para controlar si el cliente de ubicación estará obteniendo datos

// Estado para saber si está obteniendo la ubicación  
val isTracking = remember **{** *mutableStateOf*(false) **}**

4.3.- Creamos y configuramos una variable de estado para la solicitud de ubicación:

val locationRequest = remember **{** // Usa GPS para obtener la ubicación más precisa.

// Cada 5 segundos actualiza la ubicación  
 LocationRequest.Builder(Priority.*PRIORITY\_HIGH\_ACCURACY*, 5000)   
 .build()  
**}**

Podemos pasar más datos de configuración a esa variable usando varias funciones **set.** Por ejemplo, si queremos que las actualizaciones de ubicación se realicen si pasan 5 segundos o si el dispositivo se mueve más de 5 metros:

LocationRequest.Builder(Priority.*PRIORITY\_HIGH\_ACCURACY*, 5000)  
 .setMinUpdateDistanceMeters(4f) // Cuando se mueva 4 metros  
 .build()

4.4.- Creamos el objeto que se encargará de recibir las actualizaciones de ubicación:

//objeto que recibirá las actualizaciones de ubicación  
val locationCallback = remember **{** //El objeto implementa el interface LocationCallback  
 object : LocationCallback() {  
 // Si se recibe una actualización de ubicación se hace esto  
 override fun onLocationResult(locationResult: LocationResult) {  
 // si se recibió una actualización se actualizan latitud y longitud  
 locationResult.*lastLocation*?.*let* **{** location **->** textLatitud.value = "Latitud: ${location.*latitude*}"  
 textLongitud.value = "Longitud: ${location.*longitude*}"  
 **}** }  
 }  
**}**

4.5.- Añadimos la siguiente función que sirve para que se inicie la información de actualización de la ubicación.

//Función para iniciar la actualización de ubicación  
fun startLocationUpdates() {  
 //Verificamos si el usuario ha dado permisos  
 if (ContextCompat.checkSelfPermission(context, Manifest.permission.*ACCESS\_FINE\_LOCATION*) ==  
 PackageManager.*PERMISSION\_GRANTED*) {  
 // Si ha dado permisos se inicia la actualización de ubicación  
 //se pasa la solicitud y el callback  
 //y se pasa el Looper para que se ejecute en el hilo principal  
 fusedLocationClient.requestLocationUpdates(locationRequest,  
 locationCallback,  
 Looper.getMainLooper())  
 //se cambia el estado a true (para que se use en el Button)  
 isTracking.value = true  
 }  
}

4.6.- Ya sólo te queda a ti programar el Button para que al clicarlo se inicie la actualización de la ubicación y que el Button quede desactivado.

5.- Puedes comprobar que esto funciona correctamente.

Pero hay un problema que no apreciamos: Una vez iniciado un actualizador de ubicación, sigue funcionando y consumiendo recursos y batería, aunque la app se cierre. Tal como lo hemos programado, sólo si se apaga el dispositivo se cerraría este actualizador de ubicación.

Para solucionar este problema, hay que trabajar en el ciclo de vida del composable o de la activity en que se inició la actualización de ubicación.

Aunque se puede hacer de varias formas, esta vez lo vamos a ver con un mecanismo novedoso en **Jetpack Compose.**

Seguimos programando en el composable **HomeView.**

5.1.- Añadimos una variable de estado para controlar el ciclo de vida de la Activity de la app:

//estado que obtiene el ciclo de vida de la Activity   
// que lanzo el composable (mainactivity)  
val lifecycleOwner = *LocalLifecycleOwner*.current

5.2.- Añadimos una función para poder detener la actualización de la ubicación:

//detiene la actualización de ubicación  
fun stopLocationUpdates() {  
 fusedLocationClient.removeLocationUpdates(locationCallback)  
}

5.3.- Y añadimos el código para controlar el ciclo de vida y responder a alguno de los estados del ciclo de vida:

//se ejecuta cuando el composable se crea y se destruye  
// cuando se sale de la pantalla  
DisposableEffect(lifecycleOwner) **{** //observador que escucha los cambios en el ciclo de vida del owner  
 // (de la mainactivity)  
 // \_ es el LifecycleOwner (no lo usamos, lo ignoramos)  
 val observer = *LifecycleEventObserver* **{** \_, event **->** when (event) {  
 // Reiniciar ubicación cuando entra pantalla  
 Lifecycle.Event.*ON\_RESUME* -> startLocationUpdates()  
 // Detener ubicación cuando sale de pantalla  
 Lifecycle.Event.*ON\_STOP* -> stopLocationUpdates()  
 else -> {}  
 }  
 **}** //añadir el observador al ciclo de vida del owner  
 lifecycleOwner.lifecycle.addObserver(observer)  
 //cuando se sale de la pantalla se elimina el observador  
 onDispose **{** lifecycleOwner.lifecycle.removeObserver(observer)  
 **}  
}**

6.- Pero puede ser muy normal que estas aplicaciones de ubicación necesiten tener activa la ubicación en segundo plano. Para este caso, solo deberíamos programar que se detenga la actualización cuando se destruye la activity:

DisposableEffect(lifecycleOwner) **{** //observador que escucha los cambios en el ciclo de vida del owner  
 // (de la mainactivity)  
 // \_ es el LifecycleOwner (no lo usamos, lo ignoramos)  
 val observer = *LifecycleEventObserver* **{** \_, event **->** when (event) {  
 // Detener ubicación al terminar la app  
 Lifecycle.Event.*ON\_DESTROY* -> stopLocationUpdates()  
 else -> {}  
 }  
 **}** //añadir el observador al ciclo de vida del owner  
 lifecycleOwner.lifecycle.addObserver(observer)  
 //cuando se sale de la pantalla se elimina el observador  
 onDispose **{** lifecycleOwner.lifecycle.removeObserver(observer)  
 **}  
}**