{desafío}
latam_

Persistencia y bases de datos _

Parte 1

¿Qué aprenderemos?

Aprenderemos qué opciones presenta Android para persistir datos en una aplicación.:

- Aprenderemos la diferencia entre data privada y pública
- Opciones de persistencia de datos simples, estructurados y archivos
- El uso de SharedPreferences como almacenamiento de datos simples
- El uso de una capa de abstracción sobre SQLite, Room, como ORM en Android
- Algunas opciones NoSQL para almacenar datos en Android

Todos los ejemplos y código de esta presentación se encuentran escritos en Kotlin.

Persistencia de Datos en Android

Objetivos

Aprenderemos:

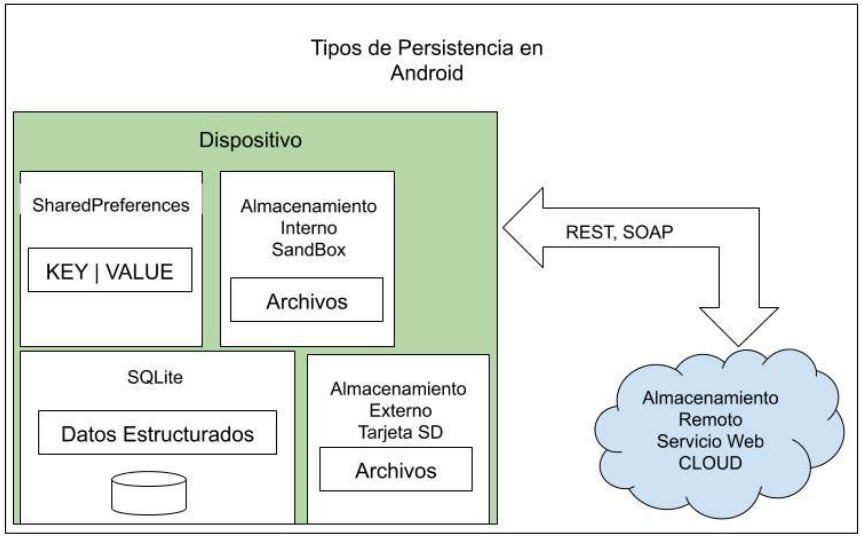
- Entender qué opciones nos entrega Android para persistir datos en nuestra Aplicación
- Elegir la mejor opción para almacenar datos en nuestra aplicación, considerando la disponibilidad y privacidad de los mismos
- Conocer las opciones que ofrece Android para compartir datos entre aplicaciones

Es recomendable tener claro que es considerado datos o data en una aplicación, y que es un dato privado o no.

Tipos de persistencia en Android

En el diagrama a la derecha vemos un mapa conceptual de qué tipos de persistencia nos entrega Android como sistema:

- SharedPreferences
- SQLite
- App Sandbox
- Externos: SD
- Remoto: WS o CLOUD



Tipos de persistencia en Android

SharedPreferences:

- Almacenamiento de primitivas (Int, Float, etc)
- Formato Clave - Valor
- Seguridad mínima
- Puede ser eliminada por el usuario
- Ideal para almacenar preferencias, datos e información volátil.

Almacenamiento Interno

- Almacenamiento en el Sandbox de la aplicación
- Almacenamiento de archivos.
- Sólo la aplicación puede acceder a este espacio. El usuario root también.
- Se puede exponer información con un ContentProvider
- Seguridad media, mejora con encriptación

Tipos de persistencia en Android

Almacenamiento externo:

- Normalmente una memoria SD
- Público por definición
- No se recomienda almacenar información privada en este lugar
- Reservado para almacenar contenido de gran tamaño

Base de datos SQLite

- Almacenamiento de información estructurada
- Privado, a menos que se exponga un ContentProvider con acceso a ella
- Eficiente para almacenar relaciones entre datos, en tablas.
- Mucho código es necesario si no se cuenta con un ORM para manejar la base de datos

Tipos de persistencia en Android

Almacenamiento remoto:

- Podemos alojar información en un servicio remoto.
- Normalmente se utiliza un conjunto de APIs que implementa el protocolo REST, lo que facilita su uso debido a la robustez del estándar.
- Seguridad depende del tipo de servicio, normalmente se tiene autenticación por token autorenovable.

Ejercicio

Debo crear una aplicación que maneje una lista de contactos, y que pueda sincronizar entre diferentes dispositivos, por ejemplo un teléfono y un tablet.

¿Qué tipo de almacenamiento me conviene en el dispositivo?.

Ejercicio - Solución

En este caso la respuesta es bastante sencilla, me conviene utilizar una base de datos. Al utilizar este tipo de almacenamiento puedo manejar mis datos de manera estructurada, persistente y más confiable a largo plazo. Al ser un sistema estructurado me será más fácil integrar un servicio web para enviar los datos a un servicio remoto y mantener mis contactos sincronizados entre dispositivos.

Datos y Seguridad

Debemos considerar con qué tipo de datos estamos tratando, para poder decidir con qué tipo de persistencia es el que más nos conviene en una aplicación. Debemos dar respuesta a las siguientes preguntas:

- ¿Qué tipo de información o datos estamos almacenando?.
- ¿Los datos son públicos o privados?
- ¿Qué disponibilidad debe tener esa información o datos?
- ¿Quién tiene acceso a esa información o datos ?

Veamos algunos ejemplos

Aplicación Cámara teléfono

- Los datos son imágenes o videos de gran tamaño
- Las imágenes y videos de la cámara principal del teléfono son mayormente públicos. Si el usuario desea tener contenido privado, la responsabilidad es del usuario, no de la aplicación.
- Las imágenes deben estar disponibles hasta que el usuario las elimine. La cámara sólo avisa cuando se está acabando el almacenamiento disponible
- Las imágenes y videos son accesibles a otras aplicaciones y a los usuarios que el dueño del teléfono estime conveniente.

Por lo tanto el sistema de persistencia más recomendado para este caso es el almacenamiento externo. Las tarjetas sd son de gran tamaño y están disponibles para las otras aplicaciones del dispositivo si tienen los permisos necesarios para usarla.

Configuración de preferencias en una aplicación

- Los datos almacenados son pequeños, normalmente números, cadenas de texto o booleanos. Nada muy complejo ni de gran tamaño
- Las preferencias pueden ser privadas a una aplicación, por ejemplo sus notificaciones push.
- La información debe estar disponible, pero si se elimina o pierde no representa un gran daño o problema, salvo un inconveniente menor para el usuario.
- El usuario tiene acceso a esa información, la aplicación también.

En este caso como no estamos almacenando estructuras complejas, no tenemos problemas de privacidad o disponibilidad, lo mejor es usar SharedPreferences.

Compartiendo Datos en Android

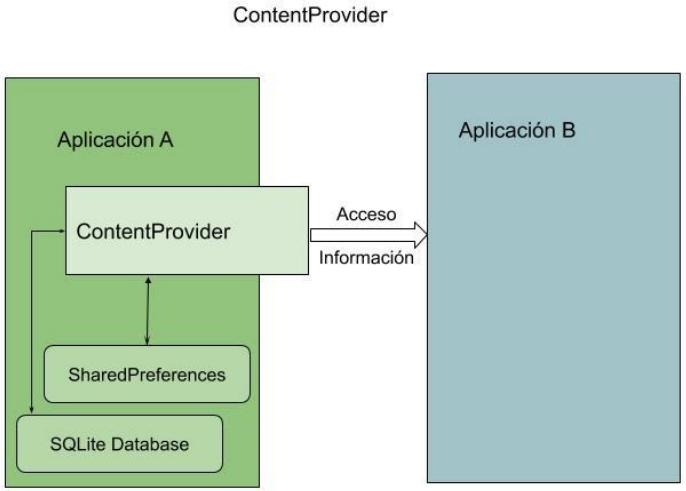
En Android podemos compartir datos de dos maneras:

- Compartir SharedPreferences: Se puede compartir el archivo de preferencias con otras apps
- ContentProvider: Es una interfaz que se expone de una aplicación a otras para que puedan acceder a información privada, de forma segura y confiable.

ContentProvider

Un ContentProvider nos permite hacer lo siguiente:

- Compartir información de una aplicación A a una aplicación B, de forma segura y confiable
- Nos permite controlar qué puede y que no puede acceder la aplicación B
- Nos permite exponer una interfaz que incluso puede permitir modificar información en A desde B.



SharedPreferences

Objetivos

Aprenderemos:

- Entender que es SharedPreferences y como funciona en una aplicación Android.
- Uso correcto de SharedPreferences, Que y que no podemos almacenar en ella
- Compartir información entre aplicaciones a través de SharedPreferences

SharedPreferences es la herramienta más sencilla y utilizada para almacenar información en Android.

SharedPreferences

- Nos permite almacenar información sencilla en formato clave - valor
- Almacena primitivas: String, Integer, Float, Boolean, Long y Sets de Strings
- Se crea un archivo de preferencias al que sólo tiene acceso la aplicación
- El archivo está normalmente relacionado con el nombre del paquete de la aplicación
- El usuario puede eliminar los datos de este archivo cuando lo desee.
- Existe Preference, que es una conjunto de apis para construir una pantalla de preferencias, que almacena los datos en un archivo de SharedPreferences

SharedPreferences

Para crear un archivo de SharedPreferences en una aplicación, en modo privado, debemos ejecutar el siguiente código, debemos tener un Context para utilizar getSharedPreferences:

```
lateinit var sharedPreferences: SharedPreferences
   override fun onCreate(savedInstanceState: Bundle?) {
       super.onCreate(savedInstanceState)
       setContentView(R.layout.activity_main)
       val fileName = "cl.desafiolatam.persistenciaandroid"
       sharedPreferences = getSharedPreferences(fileName, Context.MODE_PRIVATE)
   }
```

SharedPreferences

Existen otros modos para inicializar el archivo de preferencias, pero ninguno está recomendado por temas de seguridad, estos tres modos están deprecados:

```
 val fileName = "cl.desafiolatam.persistenciaandroid"
//Modo recomendado, privado
 sharedPreferences = getSharedPreferences(fileName, Context.MODE_PRIVATE)
//otros modos
//Modo lectura para todas las otras aplicaciones, deprecado
sharedPreferences = getSharedPreferences(fileName, Context.MODE_WORLD_READABLE)
//Modo escritura para todas las otras aplicaciones, deprecado
sharedPreferences = getSharedPreferences(fileName, Context.MODE_WORLD_WRITEABLE)
//Modo lectura/escritura para todas las otros procesos de una aplicación, deprecado
sharedPreferences = getSharedPreferences(fileName, Context.MODE_MULTI_PROCESS)
```

SharedPreferences.Editor

Un Editor nos permite almacenar información en el archivo de SharedPreferences:

```
sharedPreferences = getSharedPreferences(fileName, Context.MODE_PRIVATE)
//edit no se puede utilizar sólo, se debe ejecutar algún método y luego aplicar commit o
//apply
sharedPreferences.edit().putString(“TestString”,”Hola”).commit()
sharedPreferences.edit().putString(“TestString2”,”Hola 2”).apply()
```

Los métodos apply y commit funcionan de la siguiente manera:

- apply se ejecuta asíncronamente contra el archivo de preferencias
- commit se ejecuta síncronamente contra el archivo de preferencias
- Para ambos aplica que el último que escribe, sobrescribe los datos finales.

SharedPreferences.Editor

Otros métodos de Editor:

```
//el método put almacena el tipo que acompaña al método
sharedPreferences.edit().putString("TestString", "Hola").apply()
sharedPreferences.edit().putBoolean("TestBoolean", true).apply()
sharedPreferences.edit().putFloat("TestFloat", 1.0f).apply()
sharedPreferences.edit().putInt("TestInt", 1).apply()
sharedPreferences.edit().putLong("TestLong", 1L).apply()
val setString = setOf("one", "two", "three", "four")
sharedPreferences.edit().putStringSet("TestStringSet", setString).apply()
```

SharedPreferences

SharedPreferences nos ofrece otros métodos para interactuar con los datos almacenados:

```
//para saber si contiene cierta clave
sharedPreferences.contains("TestString")
//si quiero todos los valores y claves contenidos, uso all gracias a kotlin, en Java es
//getAll
sharedPreferences.all
//obtener un boolean
sharedPreferences.getBoolean("TestBoolean", false)
//obtener un int
sharedPreferences.getInt("TestInt", -1)
//obtener un long
sharedPreferences.getLong("TestLong", -1L)
//obtener un Float
sharedPreferences.getFloat("TestFloat", -1.0f)
//obtener un String
sharedPreferences.getString("TestString", "NotFound")
//obtener un Set<String>
sharedPreferences.getStringSet("TestStringSet", setOf())
```

SharedPreferences.OnSharedPreferencesChangeListene

Esta interfaz nos permite “escuchar” los cambios en el archivo de preferencias:

```
class MainActivity:AppCompatActivity(),SharedPreferences.OnSharedPreferenceChangeListener {
   override fun onSharedPreferenceChanged(changedSharedPreferences: SharedPreferences?,
                                          changedKey: String?) {

    if (changedKey.equals("UserNickName", true)) {
           val newNickName =  changedSharedPreferences?.getString(changedKey, "")
           displayText.text = newNickName
       }
   }
   private lateinit var sharedPreferences: SharedPreferences
   private lateinit var displayText:TextView
...
//En este caso tenemos una Actividad que escucha los cambios en el archivo de preferencias.
//Al implementar la interfaz debemos implementar el método onSharedPreferenceChanged. En
//ese método recibimos la notificación y los cambios del archivo.
//para registrar el listener utilizamos el siguiente método
sharedPreferences.registerOnSharedPreferenceChangeListener(listener)
```

SharedPreferences.OnSharedPreferencesChangeListener

Podemos implementar el Listener como un lambda y registrarlo de la siguiente manera:v

```
val listener = SharedPreferences
    .OnSharedPreferenceChangeListener { preferences, key ->
     if (key.equals("TestString", true)) {
        displayText.text = preferences.getString(key, "")
     }
}

sharedPreferences.registerOnSharedPreferenceChangeListener(listener) // en onResume
sharedPreferences.unregisterOnSharedPreferenceChangeListener(listener) // en onPause
```

El listener debe ser una referencia fuerte en la clase interesada en escuchar, las referencias en SharePreferences son weak, para evitar memory leaks. Normalmente registramos el listener en onResume, y lo deregistramos en onPause. Se destruye en onDestroy

{desafío} latam_

Academia de talentos digitales

www.desafiolatam.com