PROGRAMACIÓN DE APLICACIONES MÓVILES NATIVAS

CodeLab: Persistencia de datos

Raúl Cruz Ortega
Cuarto / Grupo 41
04/11/2023

1. Introducción

En esta práctica, se exploraron los fundamentos de la persistencia de datos en dispositivos móviles utilizando Kotlin y Android Studio. Proporcionando los conocimientos necesarios para almacenar datos de manera local en el dispositivo y hacer que las apps funcionen a pesar de las interrupciones de la red para ofrecer una experiencia al usuario fluida y coherente. Este informe destaca las actividades realizadas durante el codelabs.

2. App Inventory

Se trata de una aplicación la cual muestra una lista de elementos de la base de datos, funcionando como si fuera un inventario. En el cual el usuario tendrá las opciones de agregar un elemento nuevo, actualizar uno existente y borrarlo de la base de datos de inventario. Por ello durante el desarrollo de esta aplicación se ha aprendido lo siguiente:

- Crear un RoomDataBase el cual utilice una Entity y un Dao, haciendo uso de la biblioteca Room.

- Definir tablas de la base de datos como clases de datos anotadas con @Entity

- Definir las propiedades de las clases de datos con @ColumnInfo como columnas en las tablas.

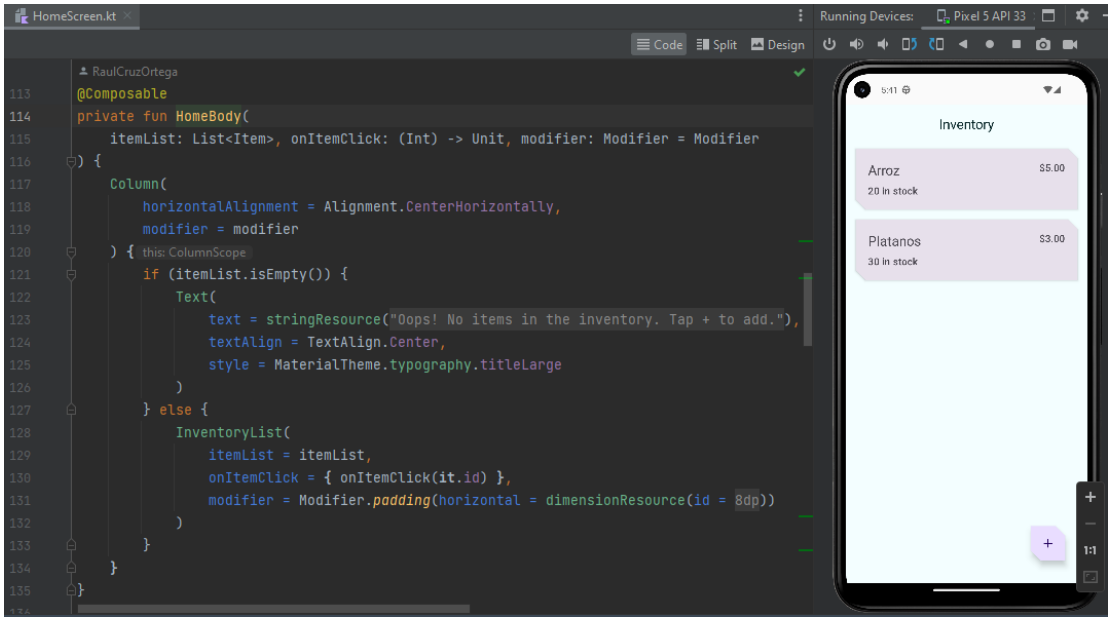
- Definir un objeto de acceso a datos (DAO) como una interfaz anotada con @Dao. El cual asigna funciones de Kotlin a consultas de bases de datos.

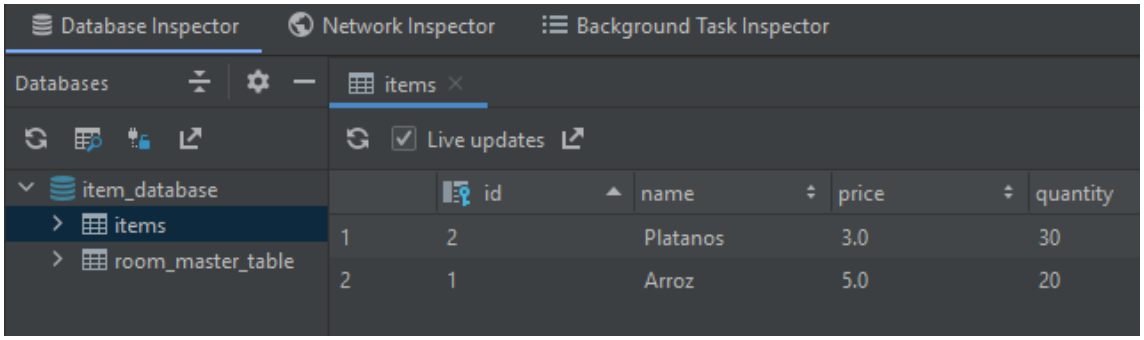
- Aprender a usar las anotaciones de conveniencia de la biblioteca Room como @Insert, @Delete y @Update.

- Realizar consultas SQL más complejas con la anotación @Query.

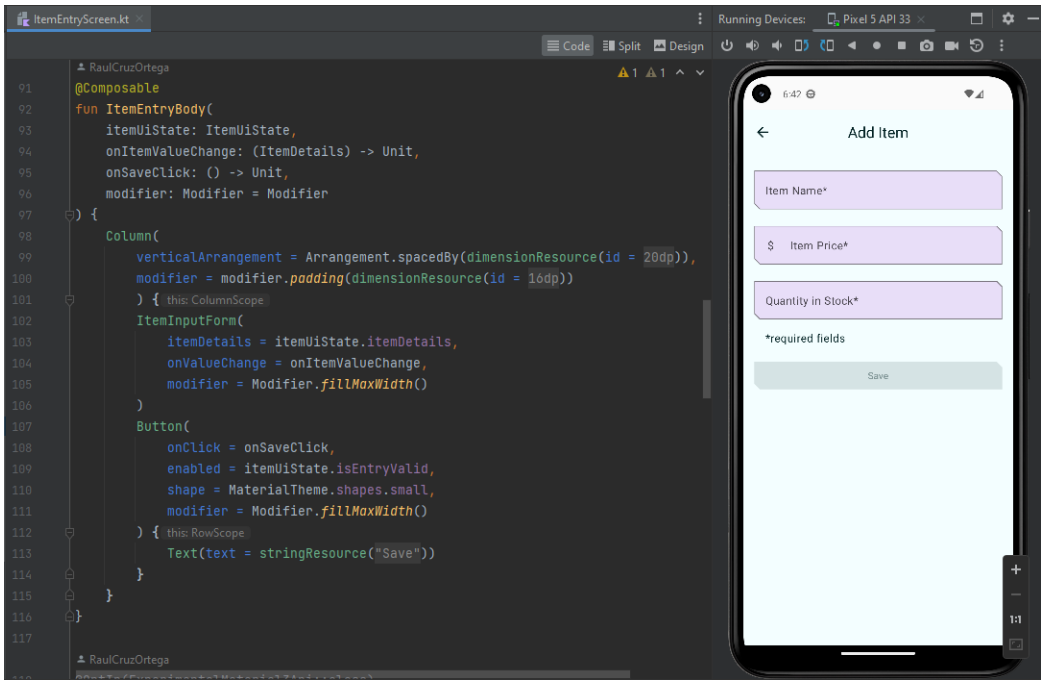
IMÁGENES DE LA APLICACIÓN:

1. HomeBody

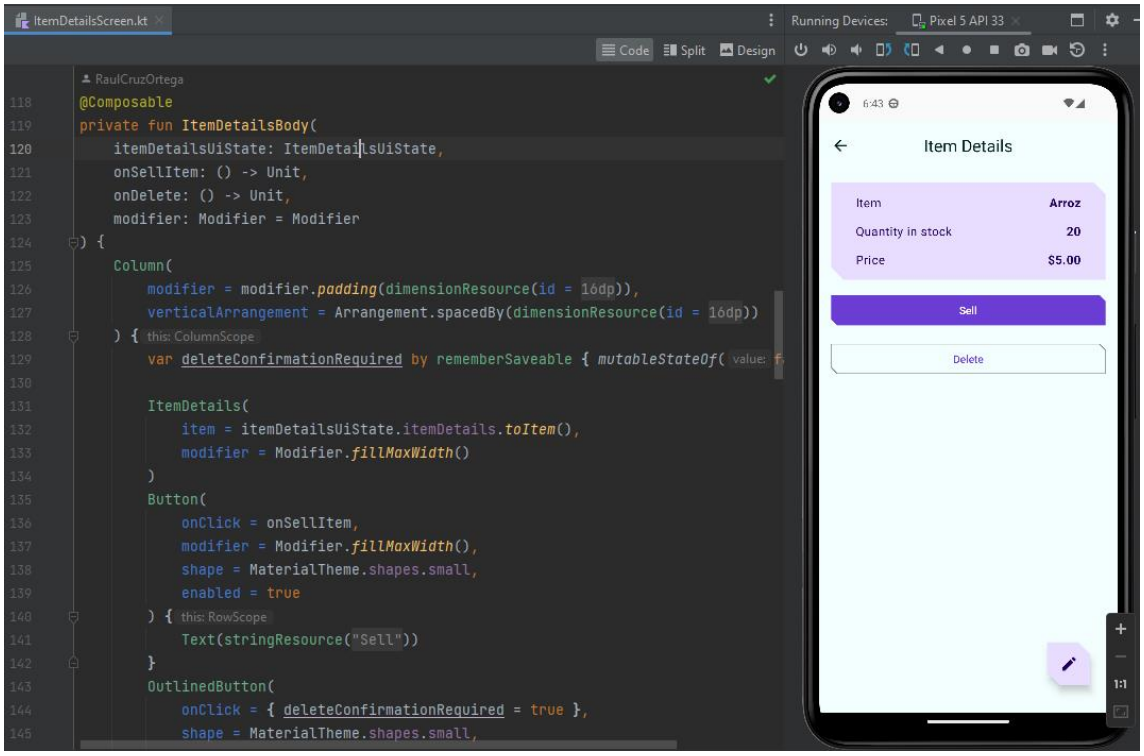




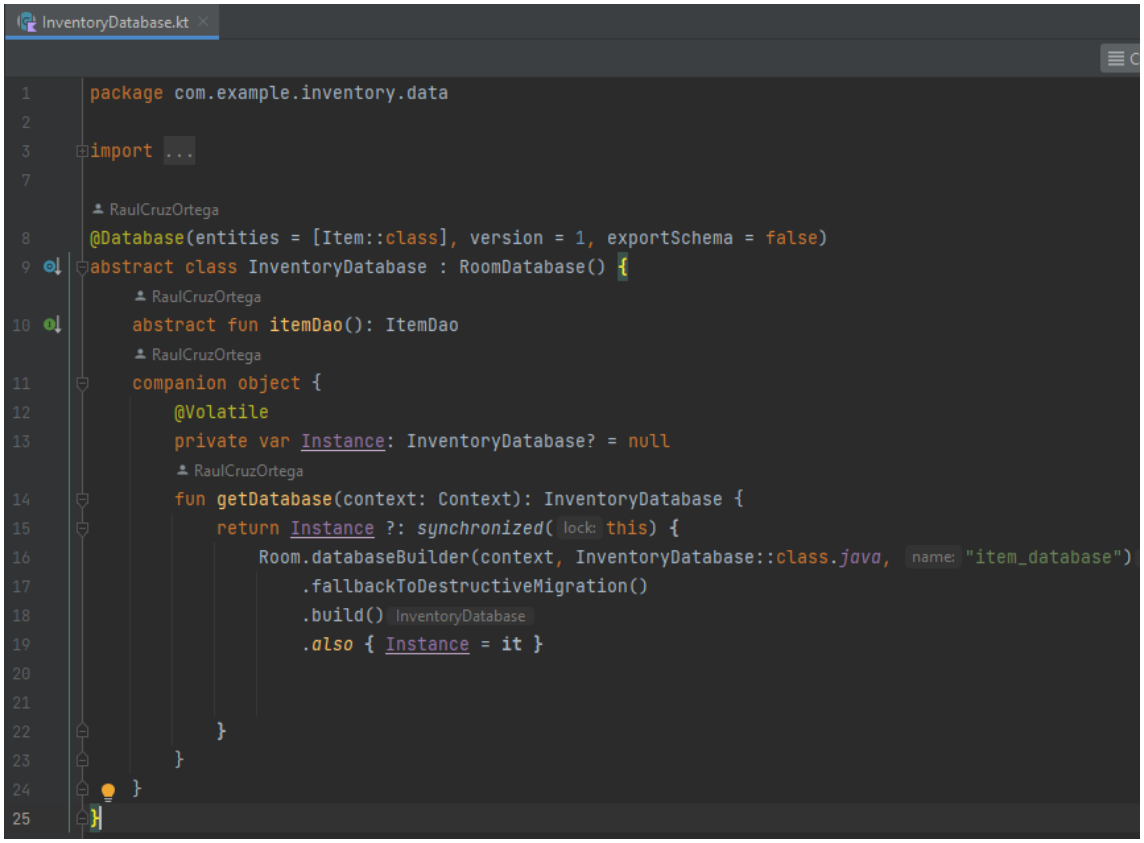
2. ItemEntryScreen



3. ItemDetailsBody



4. Inventory DataBase with Room



5. Entity Item

Item.kt

```
/ Copyright (C) 2023 The Android Open Source Project .../

package com.example.inventory.data

import ...


/**
 * Entity data class represents a single row in the database.
 */
RaulCruzOrtega
@Entity(tableName = "items")
data class Item(
    @PrimaryKey(autoGenerate = true)
    val id: Int = 0,
    val name: String,
    val price: Double,
    val quantity: Int
)
```

6. DAO of DataBase

ItemDao.kt

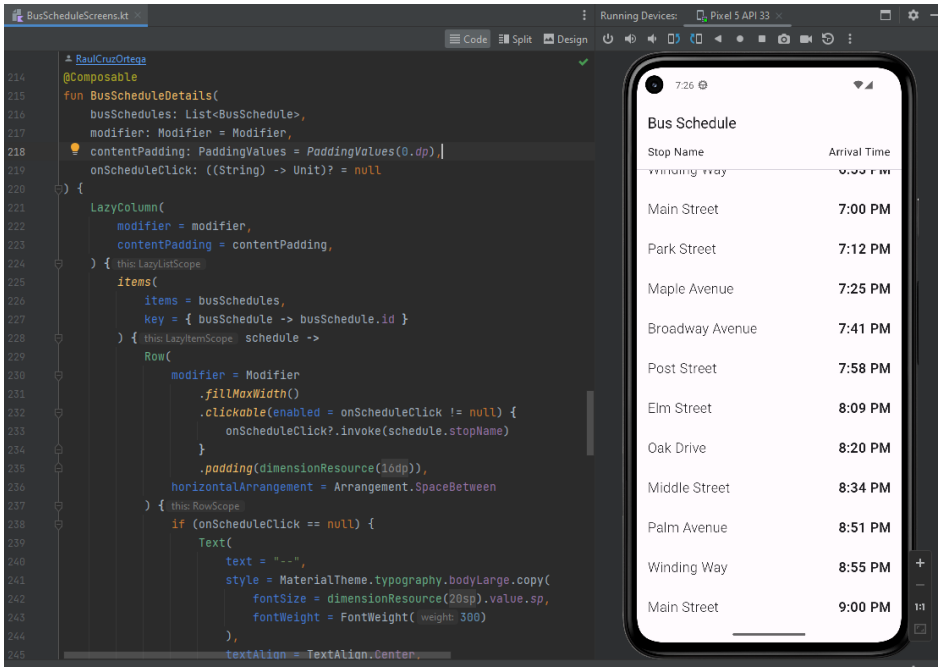
```
package com.example.inventory.data

import ...

RaulCruzOrtega
@Dao
interface ItemDao {
    RaulCruzOrtega
    @Insert(onConflict = OnConflictStrategy.IGNORE)
    suspend fun insert(item: Item)

    RaulCruzOrtega
    @Update
    suspend fun update(item: Item)

    RaulCruzOrtega
    @Delete
    suspend fun delete(item: Item)

    RaulCruzOrtega
    @Query("SELECT * from items WHERE id = :id")
    fun getItem(id: Int): Flow<Item>

    RaulCruzOrtega
    @Query("SELECT * from items ORDER BY name ASC")
    fun getAllItems(): Flow<List<Item>>
}
```

3. App Bus Schedule

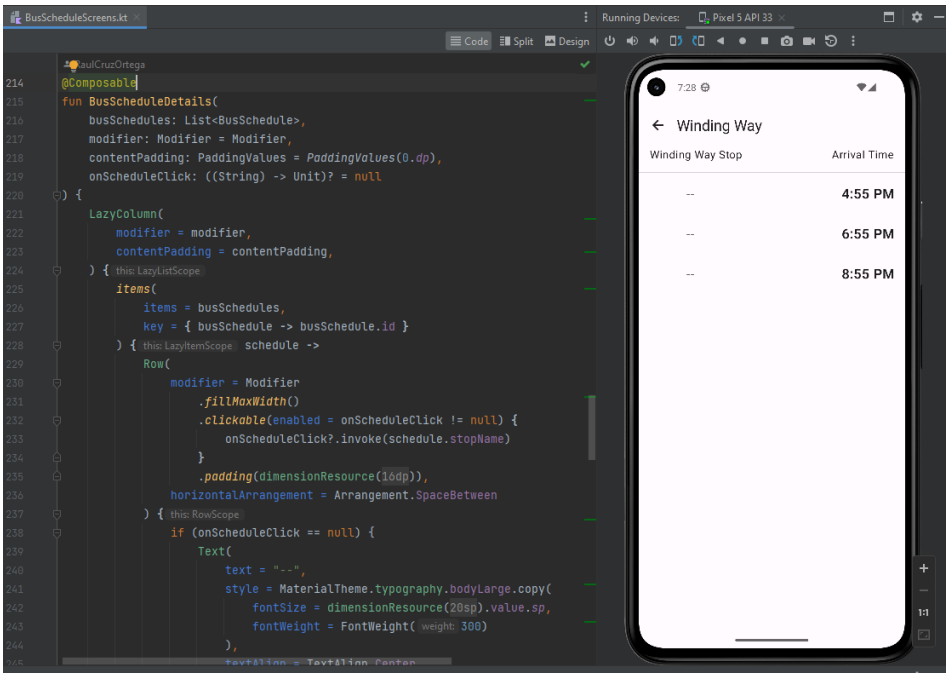
Se trata de una app para informar de los horarios del autobús en sus distintas paradas, por las que pasa. Además, si el usuario selecciona una parada en específico le aparecerá todas las horas a las cuales pasa el autobús por ella. Para desarrollar esta aplicación se ha tenido que implementar una base de datos con la biblioteca Room. Esta base de datos carga los datos de la base de datos de un recurso proporcionado.

IMÁGENES DE LA APLICACIÓN:

1. **BusSheduleHome**



2. **BusSheduleStop**



3. Database with Room

DatabaseSchedule.kt

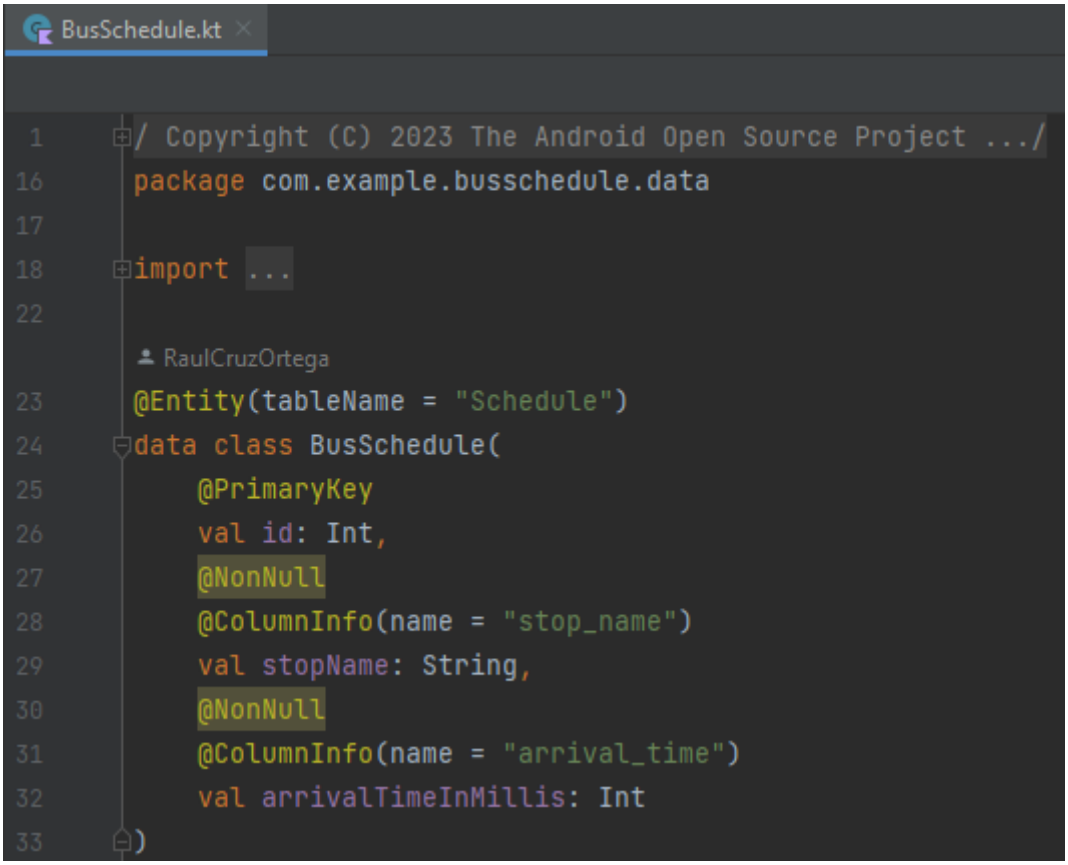
```kotlin
package com.example.busschedule.data

import ...

RaulCruzOrtega
@Database(entities = [BusSchedule::class], version = 1)
abstract class DatabaseSchedule: RoomDatabase() {
    RaulCruzOrtega
    abstract fun busScheduleDao(): BusScheduleDao

    RaulCruzOrtega
    companion object {
        @Volatile
        private var Instance: DatabaseSchedule? = null

        RaulCruzOrtega
        fun getDatabase(context: Context): DatabaseSchedule {
            return Instance ?: synchronized( lock: this) {
                Room.databaseBuilder(context, DatabaseSchedule::class.java, name: "database_schedule")
                    .createFromAsset( databaseFilePath: "database/bus_schedule.db")
                    .fallbackToDestructiveMigration()
                    .build() DatabaseSchedule
                    .also { Instance = it }

            }

        }
    }
}
```

4. DAO of DataBase

BusScheduleDao.kt

```kotlin
package com.example.busschedule.data

import ...

RaulCruzOrtega
@Dao
interface BusScheduleDao {
    RaulCruzOrtega
    @Query ("SELECT * FROM Schedule ORDER BY arrival_time ASC")
    fun getAll(): Flow<List<BusSchedule>>

    RaulCruzOrtega
    @Query ("SELECT * FROM Schedule WHERE stop_name = :stop_name ORDER BY arrival_time ASC")
    fun getByStopName(stop_name: String): Flow<List<BusSchedule>>
}
```

5. **Entity BusShedule**



4. App Dessert Release

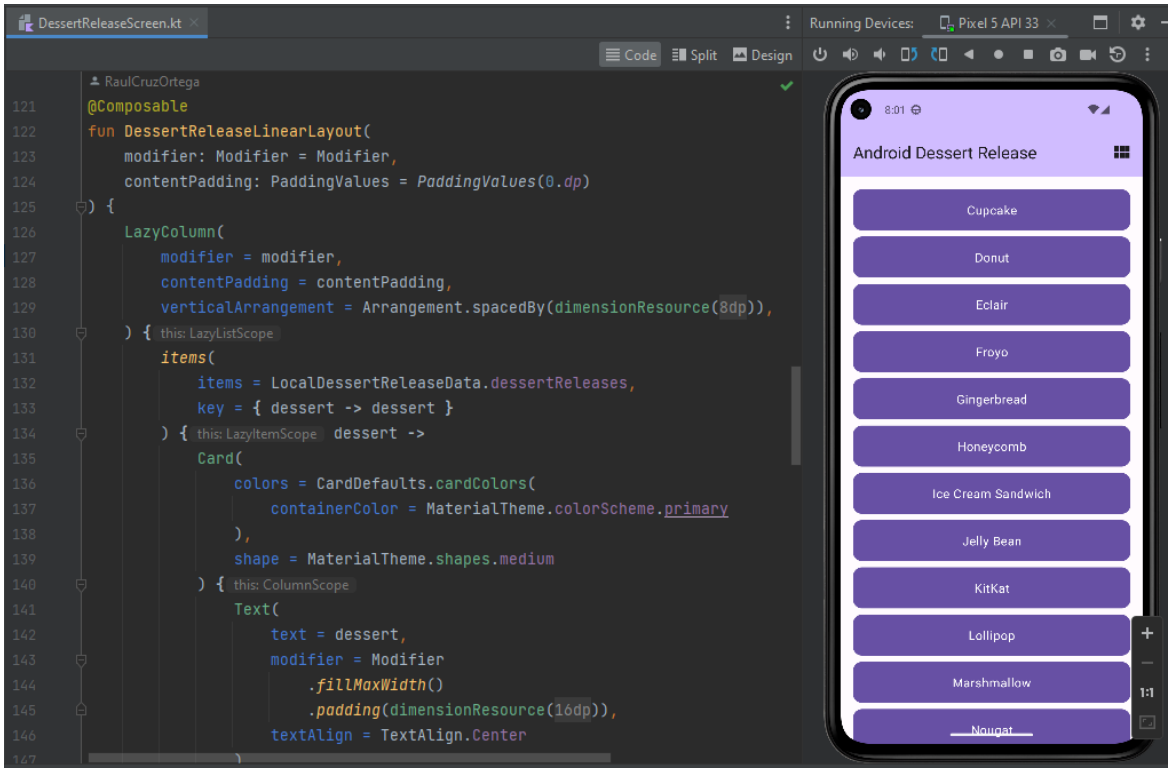
La siguiente aplicación se trata de una lista la cual muestra las distintas versiones de Android, en la cual se puede alternar el diseño de la vista entre cuadriculada o con forma de lista. A esta app se le ha agregado el **DataStore** con el objetivo de conservar el diseño seleccionado a la hora de cerrar la app y al volver a abrirla no vuelva a la selección predeterminada, si no a la guardada.

Durante el desarrolla de esta app se ha aprendido lo siguiente:

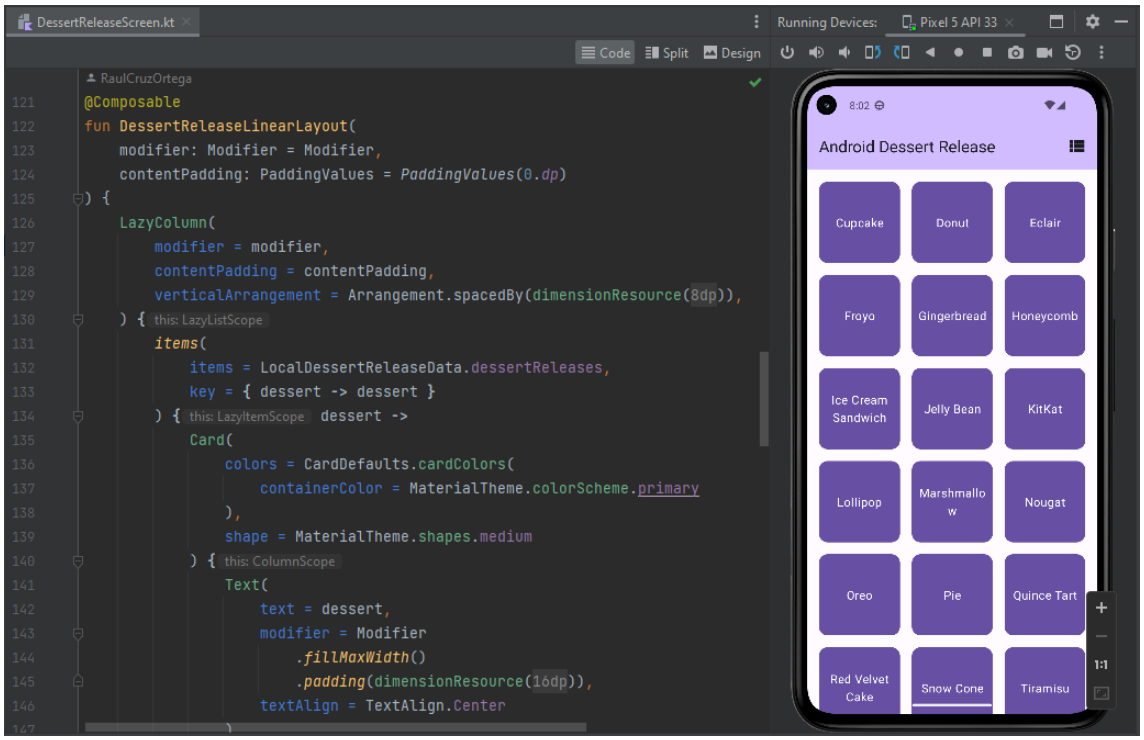
- Crear una clave booleana del **Preferences DataStore**, en la cual podemos escribir y leer un dato.
- Escribir y leer de una clave del DataStore.
- Inicializar el DataStore

IMÁGENES DE LA APLICACIÓN:

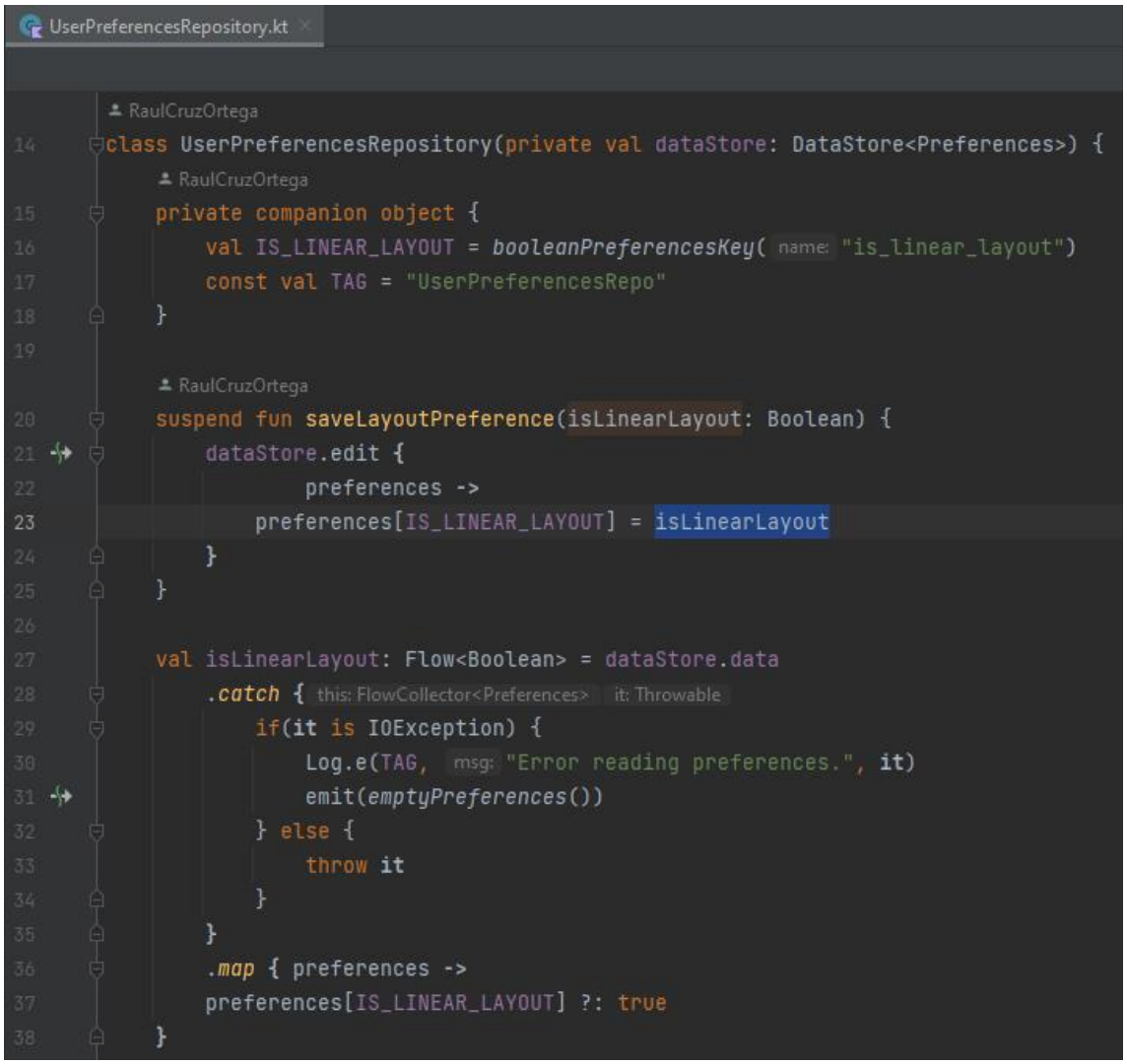
1. Diseño en lista



2. Diseño cuadriculado



3. **Preferences DataStore**



```
UserPreferencesRepository.kt

RaulCruzOrtega
class UserPreferencesRepository(private val dataStore: DataStore<Preferences>) {
    RaulCruzOrtega
    private companion object {
        val IS_LINEAR_LAYOUT = booleanPreferencesKey( name: "is_linear_layout")
        const val TAG = "UserPreferencesRepo"
    }

    RaulCruzOrtega
    suspend fun saveLayoutPreference(isLinearLayout: Boolean) {
        dataStore.edit {
                preferences ->
            preferences[IS_LINEAR_LAYOUT] = isLinearLayout
        }
    }

    val isLinearLayout: Flow<Boolean> = dataStore.data
        .catch { this: FlowCollector<Preferences>  it: Throwable
            if(it is IOException) {
                Log.e(TAG,  msg: "Error reading preferences.", it)
                emit(emptyPreferences())
            } else {
                throw it
            }
        }
        .map { preferences ->
        preferences[IS_LINEAR_LAYOUT] ?: true
    }
```

5. Opinión sobre el CodeLab

La experiencia del codelab ha resultado en una valiosa adquisición de conocimientos acerca de la creación de bases de datos mediante Room y la utilización de DataStore, lo cual ha sido sumamente enriquecedor. Sin embargo, es importante destacar un aspecto crítico a considerar. En algunos momentos, el codelab presenta un nivel de abstracción en la implementación de código que puede resultar bastante avanzado para aquellas personas que están dando sus primeros pasos en el lenguaje de programación Kotlin. Esta complejidad puede, en ocasiones, representar un obstáculo para aquellos que se encuentran en las etapas iniciales de su aprendizaje, lo que podría alejarlos del objetivo fundamental del codelab: aprender a crear y utilizar bases de datos a través de la biblioteca Room y aprovechar el potencial del DataStore. Por lo tanto, es crucial considerar la adecuación del contenido a las necesidades de un público diverso, incluyendo a aquellos que están en el proceso de iniciarse en el mundo de la programación Kotlin.