Lecture 13: SQLite
IN721: Design and Development of Applications for Mobile Devices
Semester One, 2020

Kaiako: Grayson Orr

Te Kura Matatini ki Otago, Ōtepoti, Aotearoa

Friday, 17 April

Lecture 11: Localization Topics

- Localization
- RTL & LTR

Lecture 13: SQLite Topics

- SQLite
- Formatting date & time

SQLite

- What is SQLite?
  - Open-source relational database
  - Used to perform different database operations
    - Storing, manipulating or retrieving persistent data from the database
  - Embedded in your Android device by default
- In production, you wouldn't use SQLite. This is sufficient for the assignment
- If you are interested in alternative solutions, please privately message me on MS Teams

Playlist

- Create a class called Playlist - in the helpers directory
- Three fields
  - ID
  - Name
  - Date & time
- In today's practical, feel free to add more

```
class Playlist {
    var id: Int = 0
    var name: String? = null
    var dateTime: String? = null
}
```

Playlist ViewHolder

- Create a view holder class - in the helpers directory
- Extends from RecyclerView.ViewHolder
  - Two text views
  - One image button

```
class PlaylistViewHolder(view: View) : RecyclerView.ViewHolder(view)  {
    var txvName: TextView = view.findViewById(R.id.txvPlaylistName)
    var txvDateTime: TextView = view.findViewById(R.id.txvPlaylistDateTime)
    var imgBtnMenu: ImageButton = view.findViewById(R.id.imgBtnPlaylistMenu)
}
```

IItemClick Interface

- Create an interface class called IItemClick - in the interfaces directory
- Create a function called onItemClick - takes a Playlist object & image button as its arguments

```
interface IItemClick {
    fun onItemClick(playlist: Playlist, imgBtn: ImageButton)
}
```

Playlist RecyclerView Adapter

- Create a class called PlaylistRecyclerViewAdapter - in the helpers directory
- Extends from RecyclerView.Adapter
- Similar code structure to LastFmRecyclerViewAdapter
- Override functions that we must implement:
  - getItemCount
  - onBindViewHolder
  - onCreateViewHolder
- loadNewData - notifies any data changes
- Inner class called MenuOnButtonClickListener - called the IItemClick's onItemClick function

Playlist RecyclerView Adapter

- PlaylistRecyclerViewAdapter class

```
class PlaylistRecyclerViewAdapter(var listener: IItemClick, private var playlists: ArrayList<Playlist>) :
    RecyclerView.Adapter<PlaylistViewHolder>() {

    override fun getItemCount(): Int {
        return if (playlists.isNotEmpty()) playlists.size else 0
    }

    fun notifyData(newPlaylists: ArrayList<Playlist>) {
        playlists = newPlaylists
        notifyDataSetChanged()
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): PlaylistViewHolder {
        val view: View =
            LayoutInflater.from(parent.context).inflate(R.layout.custom_playlist_list_item, parent,  attachToRoot: false)
        return PlaylistViewHolder(view)
    }

    override fun onBindViewHolder(viewHolder: PlaylistViewHolder, position: Int) {
        val playlist: Playlist = playlists[position]
        viewHolder.txvName.text = "Name: " + playlist.name
        viewHolder.txvDateTime.text = "Date: " + DateTime.formatDateTime(playlist.dateTime!!)
        viewHolder.imgBtnMenu.setOnClickListener(MenuOnButtonClickListener(playlist, viewHolder.imgBtnMenu))
    }

    inner class MenuOnButtonClickListener(var playlist: Playlist, var imgBtn: ImageButton) : View.OnClickListener {
        override fun onClick(v: View) {
            listener.onItemClick(playlist, imgBtn)
        }
    }
}
```

Playlist Menu Layout

- Create a new menu layout with two items
- One for updating a playlist & one for deleting a playlist

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/update"
        android:title="Update" />
    <item
        android:id="@+id/delete"
        android:title="Delete" />
</menu>
```

DB Helper - SQLiteOpenHelper

- Create a class called DBHelper - in the helpers directory
- The class takes context as its argument
- Extends from SQLiteOpenHelper

```
class DBHelper(context: Context) :
    SQLiteOpenHelper(context, DATABASE_NAME,  factory: null, DATABASE_VERSION)
```

DB Helper

- Override functions that we must implement:
  - onCreate
  - onUpgrade
- We want to implement basic CRUD functionality. Create four functions:
  - insert
  - selectAll
  - update
  - delete

DB Helper - Override Functions

- onCreate - creates the table. Specified in the companion object
- onUpgrade - drop table if it exists

```
override fun onCreate(db: SQLiteDatabase) {
    db.execSQL(DATABASE_CREATE)
}

override fun onUpgrade(db: SQLiteDatabase, p1: Int, p2: Int) {
    db.execSQL( sql: "DROP TABLE IF EXISTS $TABLE_NAME")
    onCreate(db)
}
```

DB Helper - CRUD

- insert & selectAll function

```
fun insert(msg: String): Long {
    val db: SQLiteDatabase = this.writableDatabase
    val values = ContentValues()
    values.put(COLUMN_NAME, msg)
    values.put(COLUMN_DATE_TIME, DateTime.currentDateTime())
    val id: Long = db.insert(TABLE_NAME,  nullColumnHack: null, values)
    db.close()
    return id
}

fun selectAll(): ArrayList<Playlist> {
    val playlists = ArrayList<Playlist>()
    val selectQuery = "SELECT * FROM $TABLE_NAME ORDER BY $COLUMN_DATE_TIME ASC"
    val db: SQLiteDatabase = this.writableDatabase
    val cursor: Cursor = db.rawQuery(selectQuery,  selectionArgs: null)
    if (cursor.moveToFirst()) {
        do {
            val playlist = Playlist()
            playlist.id = cursor.getInt(cursor.getColumnIndex(COLUMN_ID))
            playlist.name = cursor.getString(cursor.getColumnIndex(COLUMN_NAME))
            playlist.dateTime = cursor.getString(cursor.getColumnIndex(COLUMN_DATE_TIME))
            playlists.add(playlist)
        } while (cursor.moveToNext())
    }
    cursor.close()
    db.close()
    return playlists
}
```

DB Helper - CRUD

- update & delete function

```
fun update(id: Long, msg: String): Int {
    val db: SQLiteDatabase = this.writableDatabase
    val values = ContentValues()
    values.put(COLUMN_NAME, msg)
    return db.update(
        TABLE_NAME, values, whereClause: "$COLUMN_ID = ?",
        arrayOf(id.toString())
    )
}

fun delete(id: Long) {
    val db: SQLiteDatabase = this.writableDatabase
    db.delete(
        TABLE_NAME, whereClause: "$COLUMN_ID = ?",
        arrayOf(id.toString())
    )
    db.close()
}
```

DB Helper - Companion Object

- Companion object
- Contains the following information:
  - Database version
  - Database name
  - Table name
  - Column id
  - Column name
  - Column date & time
  - CREATE TABLE SQL query

```
companion object {
    const val DATABASE_VERSION = 1
    const val DATABASE_NAME = "db_playlist"
    const val TABLE_NAME = "playlists"
    const val COLUMN_ID = "id"
    const val COLUMN_NAME = "name"
    const val COLUMN_DATE_TIME = "date_time"
    const val DATABASE_CREATE: String =
        "CREATE TABLE $TABLE_NAME($COLUMN_ID INTEGER PRIMARY KEY AUTOINCREMENT, $COLUMN_NAME TEXT, $COLUMN_DATE_TIME TEXT)"
}
```

DB Helper - Formatting Date & Time

- Two functions that format date & time
- By default, it is yyyy-mm-dd hh:mm:ss
- In today's practical, you are going to use dd/MM/yyyy kk:mm:ss

```
object DateTime {
    fun currentDateTime(): String {
        val date: Date = Calendar.getInstance().time
        val outputPattern = "dd/MM/yyyy kk:mm:ss"
        val outputFormat = SimpleDateFormat(outputPattern, Locale.ENGLISH)
        return outputFormat.format(date)
    }

    fun formatDateTime(dateTime: String): String {
        val inputPattern = "dd/MM/yyyy kk:mm:ss"
        val outputPattern = "dd MMM yyyy hh:mm:ss a"

        val inputFormat = SimpleDateFormat(inputPattern, Locale.ENGLISH)
        val outputFormat = SimpleDateFormat(outputPattern, Locale.ENGLISH)

        lateinit var date: Date
        lateinit var str: String

        try {
            date = inputFormat.parse(dateTime)
            str = outputFormat.format(date)
        } catch (e: ParseException) {
            e.printStackTrace()
        }
        return str
    }
}
```

Playlist Activity - Class Declaration

- Create a new empty activity via the wizard
- Extends from BaseActivity & IItemClick
- Declare necessary fields

```
class PlaylistActivity : BaseActivity(), IItemClick {
    private lateinit var playlists: ArrayList<Playlist>
    private lateinit var dbHelper: DBHelper
    private lateinit var btnAdd: Button
    private lateinit var recyclerView: RecyclerView
    private lateinit var playlistRecyclerViewAdapter: PlaylistRecyclerViewAdapter
```

Playlist Activity - OnCreate

- onCreate code

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_playlist)
    displayToolbar(isHomeEnabled = true, isTitleEnabled = false)

    playlists = ArrayList()
    dbHelper = DBHelper( context: this@PlaylistActivity)
    playlists = dbHelper.selectAll()

    btnAdd = findViewById(R.id.btnAddPlaylist)
    btnAdd.setOnClickListener { it: View!
        addNewPlaylistDialog(DatabaseStatus.INSERT, id: 0, txt: "")
    }

    recyclerView = findViewById(R.id.rcvPlaylists)
    val layoutManager = LinearLayoutManager( context: this@PlaylistActivity)
    recyclerView.layoutManager = layoutManager
    playlistRecyclerViewAdapter = PlaylistRecyclerViewAdapter( listener: this, playlists)
    recyclerView.adapter = playlistRecyclerViewAdapter

    readDatabase()
}
```

Playlist Activity - Read Database

- Create a function called readDatabase
- Returns all records in the playlist's table
- Notifies the recycler view of any data changes
- Updates the recycler view appropriately

```
private fun readDatabase() {
    playlists = dbHelper.selectAll()
    playlistRecyclerViewAdapter.notifyData(playlists)
}
```

Playlist Activity - Dialog

- A little different to what we have done previously
- Good to see different implementations
- Set the content view to the fragment layout file
  - This layout file contains a text view, edit text & two buttons
- Two button click listeners - one for inserting/updating & one for deleting

Playlist Activity - ReadDatabase

- addNewPlaylistDialog code

```
private fun addNewPlaylistDialog(status: DatabaseStatus, id: Int, txt: String) {
    val dialog = Dialog( context: this@PlaylistActivity, R.style.DialogFullScreen)
    dialog.setCancelable(true)
    dialog.setCanceledOnTouchOutside(true)
    dialog.setContentView(R.layout.fragment_add_playlist)

    val edtAddPlaylist: EditText = dialog.findViewById(R.id.edtAddPlaylist)
    val btnClosePlaylist: Button = dialog.findViewById(R.id.btnClosePlaylist)
    val btnAddPlaylist: Button = dialog.findViewById(R.id.btnAddPlaylist)
    val txvAddPlaylist: TextView = dialog.findViewById(R.id.txvAddPlaylist)

    if (status == DatabaseStatus.UPDATE) {
        edtAddPlaylist.setText(txt)
        btnAddPlaylist.text = "Update"
        txvAddPlaylist.text = "Update Playlist"
    } else {
        edtAddPlaylist.setText("")
        btnAddPlaylist.text = "Add"
        txvAddPlaylist.text = "Add New Playlist"
    }

    btnAddPlaylist.setOnClickListener { it: View!
        if (status == DatabaseStatus.UPDATE) {
            dbHelper.update(id.toLong(), edtAddPlaylist.text.toString().trim())
            readDatabase()
        } else if (status == DatabaseStatus.INSERT) {
            dbHelper.insert(edtAddPlaylist.text.toString().trim())
            readDatabase()
        }
        dialog.dismiss()
    }

    btnClosePlaylist.setOnClickListener { dialog.dismiss() }

    dialog.show()
}
```

Playlist Activity - IItemClick

- Implement the onItemClick function
- Popup menu - inflate the playlist menu layout
- This a menu that will allow you to update & delete playlist items

```
override fun onItemClick(playlist: Playlist, imgBtn: ImageButton) {
    val popup = PopupMenu( context: this, imgBtn)
    popup.menuInflater.inflate(R.menu.menu_playlist, popup.menu)
    popup.setOnMenuItemClickListener { item ->
        when (item.itemId) {
            R.id.update -> addNewPlaylistDialog(DatabaseStatus.UPDATE, playlist.id, playlist.name!!)
            R.id.delete -> {
                dbHelper.delete(playlist.id.toLong())
                readDataBase()
            }
        }
        true ^setOnMenuItemClickListener
    }
    popup.show()
}
```

Practical

- Series of tasks covering today's lecture
- Worth 1% of your final mark for the Design and Development of Applications for Mobile Devices course
- Deadline: Friday, 12 June at 5pm

Exam 03

- Series of tasks covering lectures 09-12
- Worth 6% of your final mark for the Design and Development of Applications for Mobile Devices course
- Deadline: Friday, 24 April at 5pm