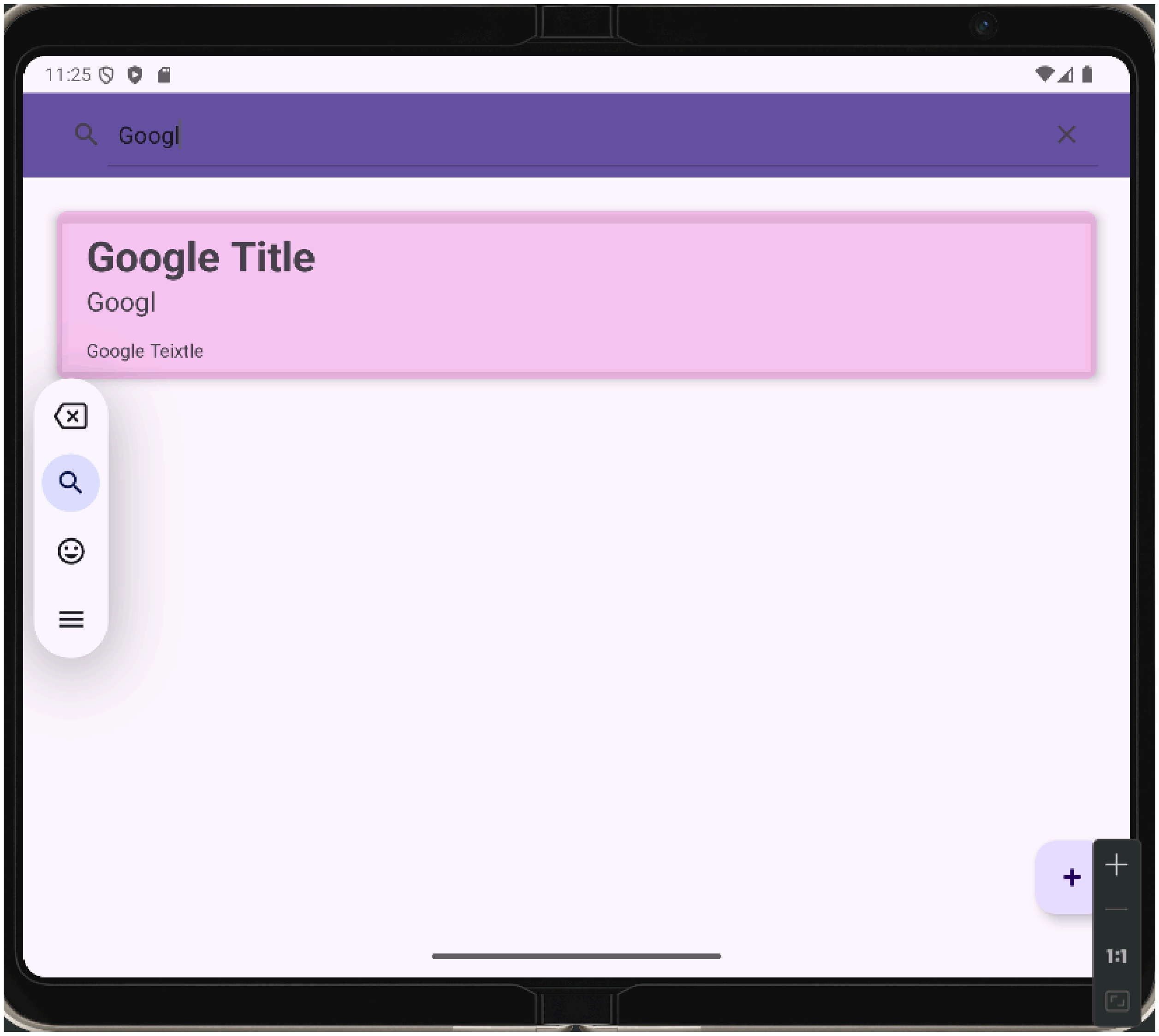
OntarioTech
UNIVERSITY

Johvonne Keane, 100784273
Ahmed Darwish, 100754743
Okiki Ojo, 100790236
Amasowomwan Daniel, 100787640
Chinomso Nosiri, 100846639

Lab #2

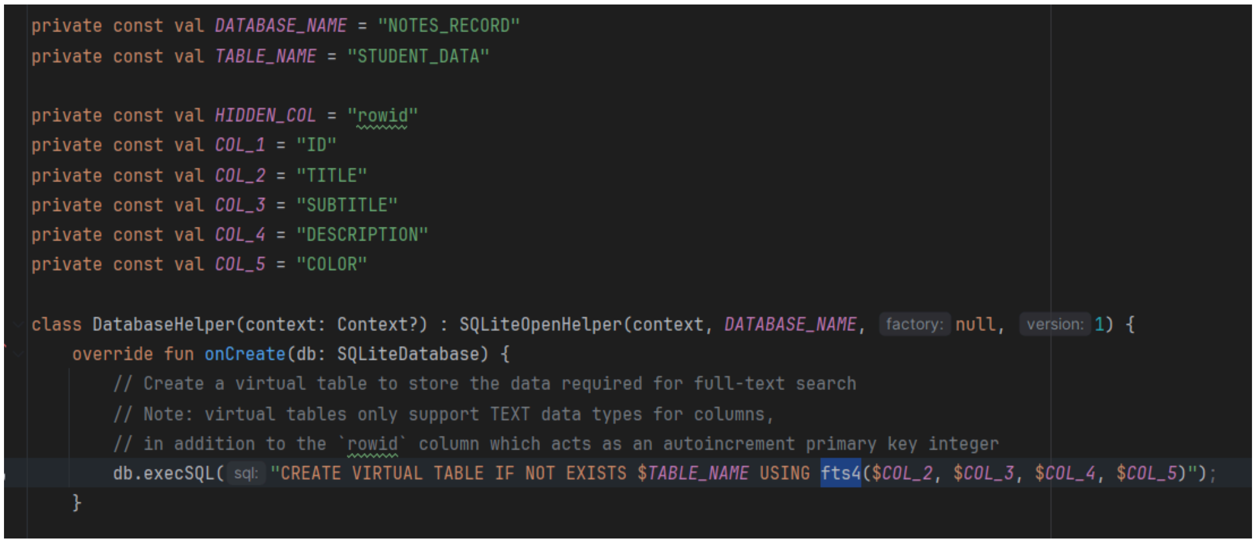
App:

Searching using Note Titles



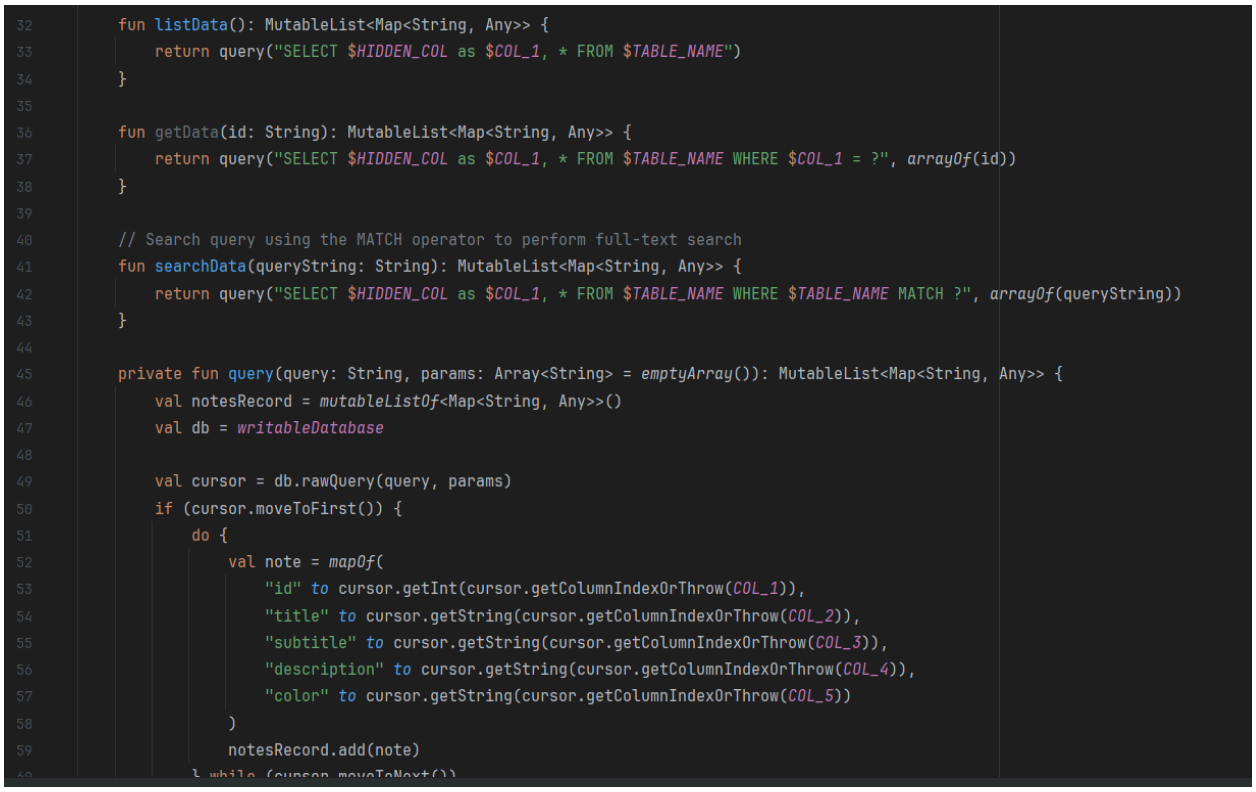
Here, we used the search bar to look up “Google Title note” as you can see, by typing “Googl” we were to automatically find the pink note of google title.

For getting the search to work we choose to use a SQLite virtual table to take advantage of the Fuzzy Search fts4 module built into SQLite, it required us to only store text and forced us to remove the ID column (as virtual tables already have a `rowid` hidden field).



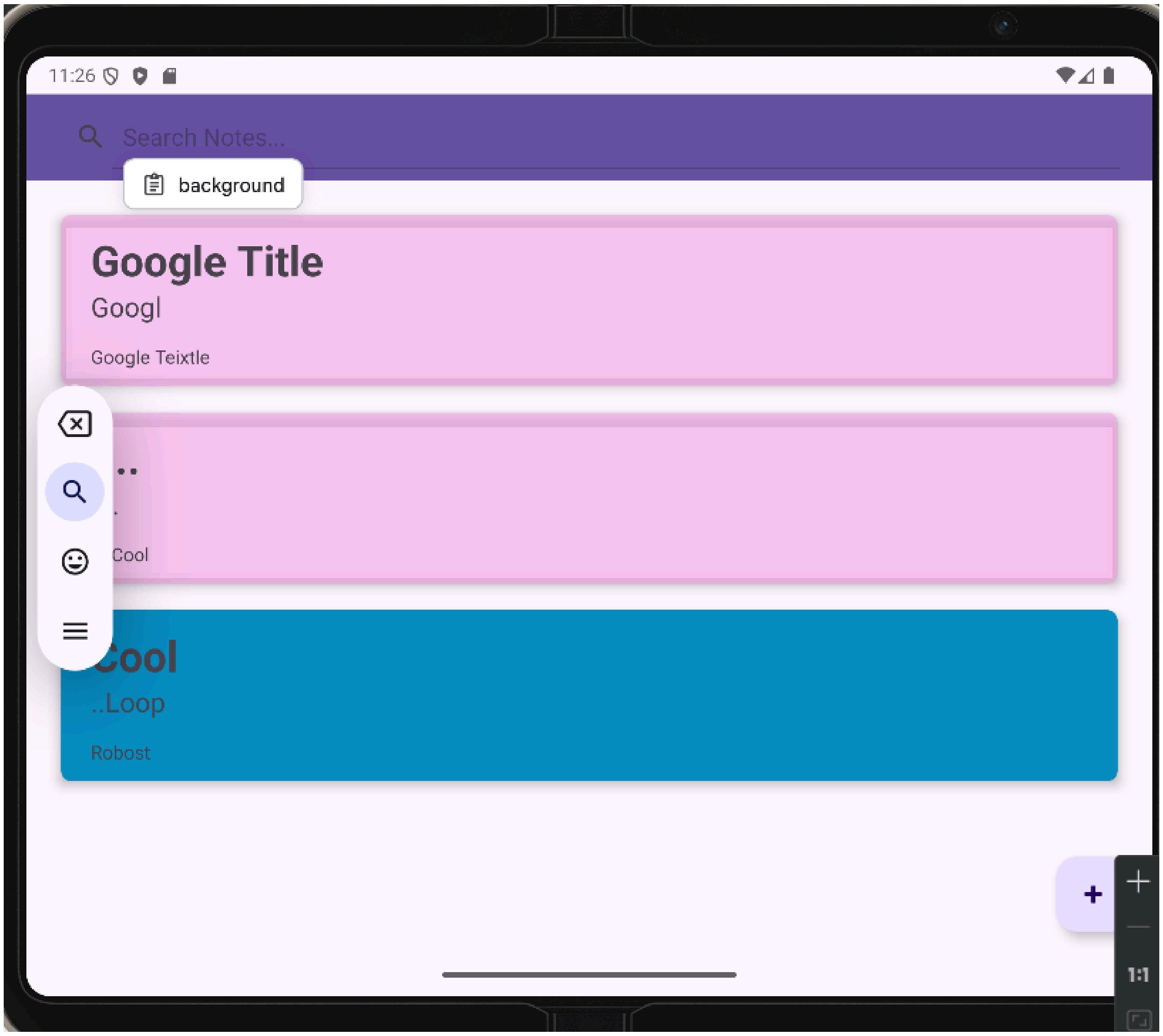
```kotlin
private const val DATABASE_NAME = "NOTES_RECORD"
private const val TABLE_NAME = "STUDENT_DATA"

private const val HIDDEN_COL = "rowid"
private const val COL_1 = "ID"
private const val COL_2 = "TITLE"
private const val COL_3 = "SUBTITLE"
private const val COL_4 = "DESCRIPTION"
private const val COL_5 = "COLOR"

class DatabaseHelper(context: Context?) : SQLiteOpenHelper(context, DATABASE_NAME, factory: null, version: 1) {
    override fun onCreate(db: SQLiteDatabase) {
        // Create a virtual table to store the data required for full-text search
        // Note: virtual tables only support TEXT data types for columns,
        // in addition to the `rowid` column which acts as an autoincrement primary key integer
        db.execSQL( sql: "CREATE VIRTUAL TABLE IF NOT EXISTS $TABLE_NAME USING fts4($COL_2, $COL_3, $COL_4, $COL_5)");
    }
```

It also forced us to modify the SQL queries to expose the hidden `rowid` field as `ID`.



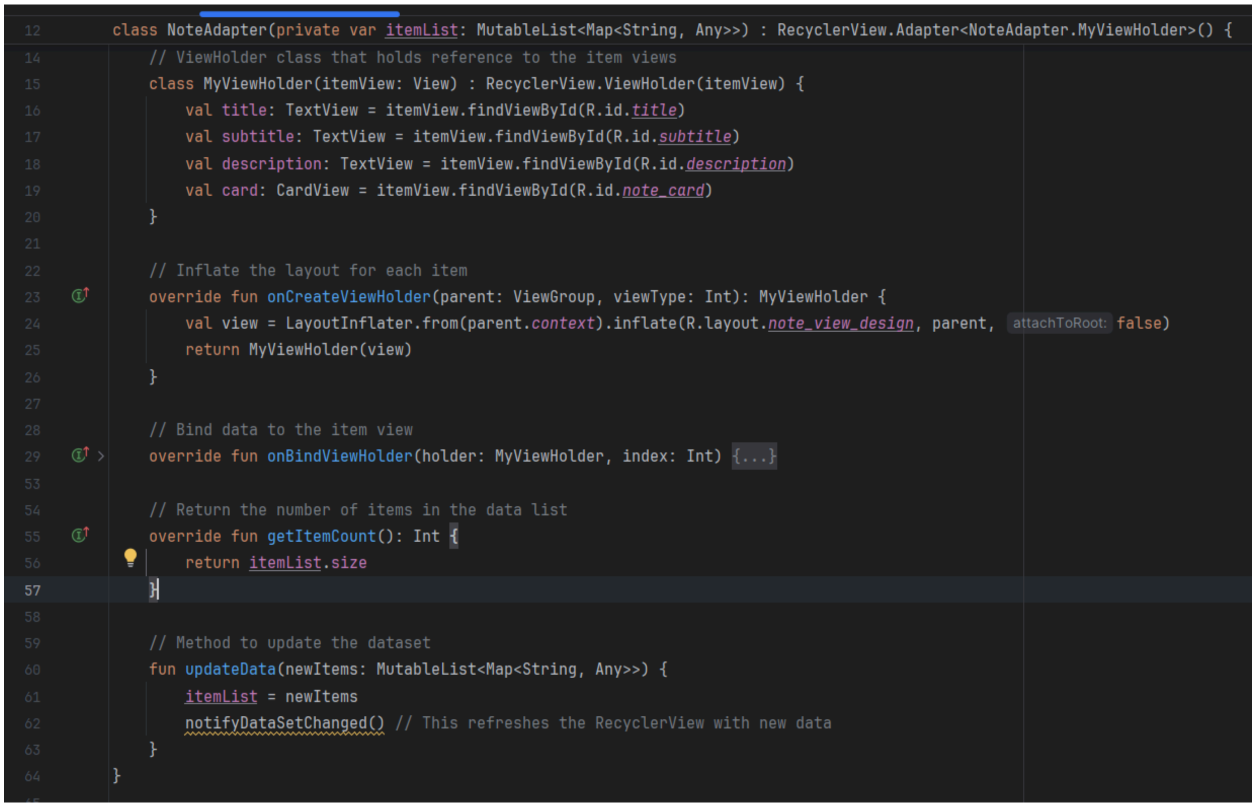
```kotlin
    fun listData(): MutableList<Map<String, Any>> {
        return query("SELECT $HIDDEN_COL as $COL_1, * FROM $TABLE_NAME")
    }

    fun getData(id: String): MutableList<Map<String, Any>> {
        return query("SELECT $HIDDEN_COL as $COL_1, * FROM $TABLE_NAME WHERE $COL_1 = ?", arrayOf(id))
    }

    // Search query using the MATCH operator to perform full-text search
    fun searchData(queryString: String): MutableList<Map<String, Any>> {
        return query("SELECT $HIDDEN_COL as $COL_1, * FROM $TABLE_NAME WHERE $TABLE_NAME MATCH ?", arrayOf(queryString))
    }

    private fun query(query: String, params: Array<String> = emptyArray()): MutableList<Map<String, Any>> {
        val notesRecord = mutableListOf<Map<String, Any>>()
        val db = writableDatabase

        val cursor = db.rawQuery(query, params)
        if (cursor.moveToFirst()) {
            do {
                val note = mapOf(
                    "id" to cursor.getInt(cursor.getColumnIndexOrThrow(COL_1)),
                    "title" to cursor.getString(cursor.getColumnIndexOrThrow(COL_2)),
                    "subtitle" to cursor.getString(cursor.getColumnIndexOrThrow(COL_3)),
                    "description" to cursor.getString(cursor.getColumnIndexOrThrow(COL_4)),
                    "color" to cursor.getString(cursor.getColumnIndexOrThrow(COL_5))
                )
                notesRecord.add(note)
            } while (cursor.moveToNext())
```

Default view showing all Notes

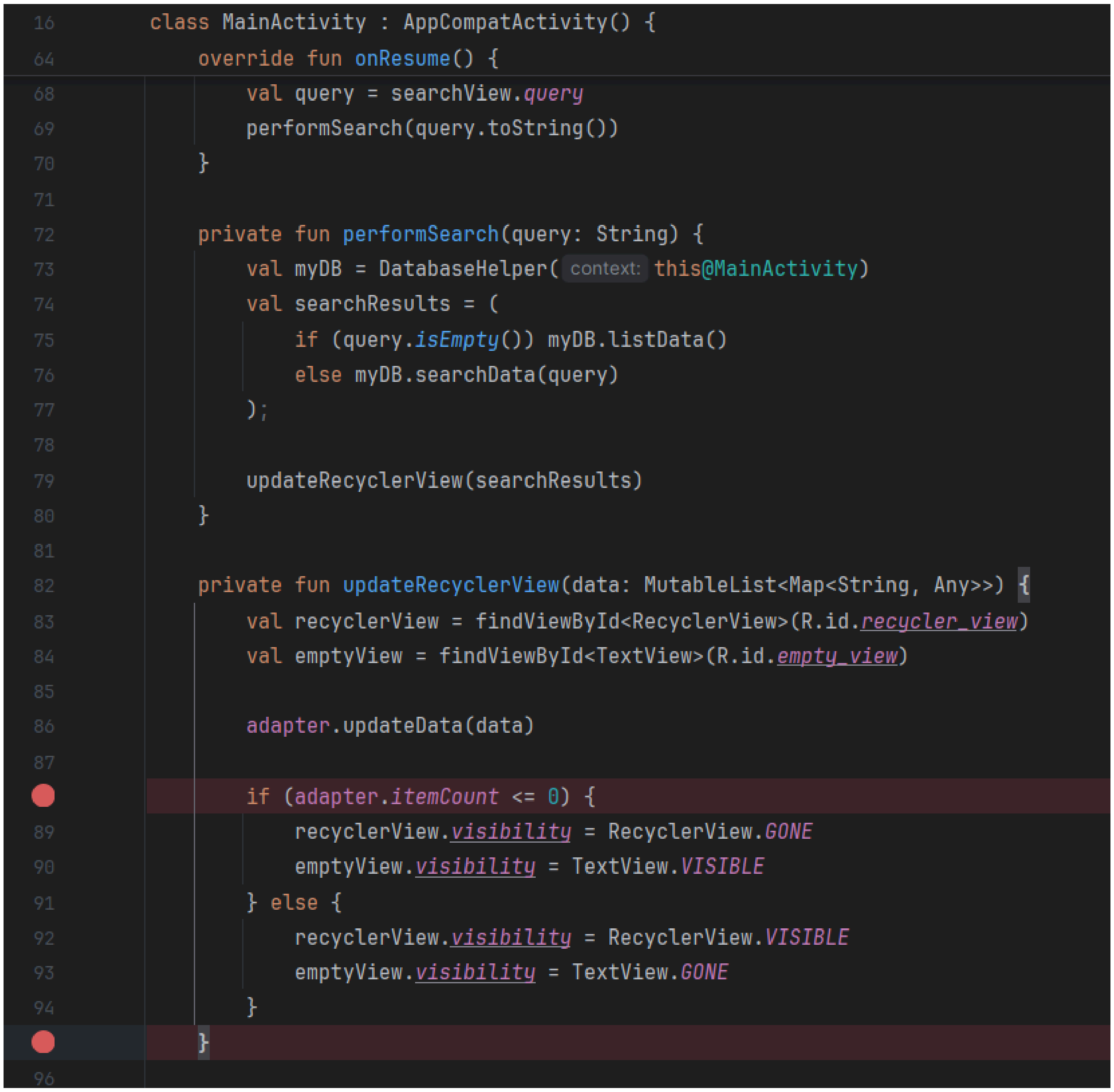


Here you can see all of the notes in the default view. The notes are displayed in different colors and with different titles.

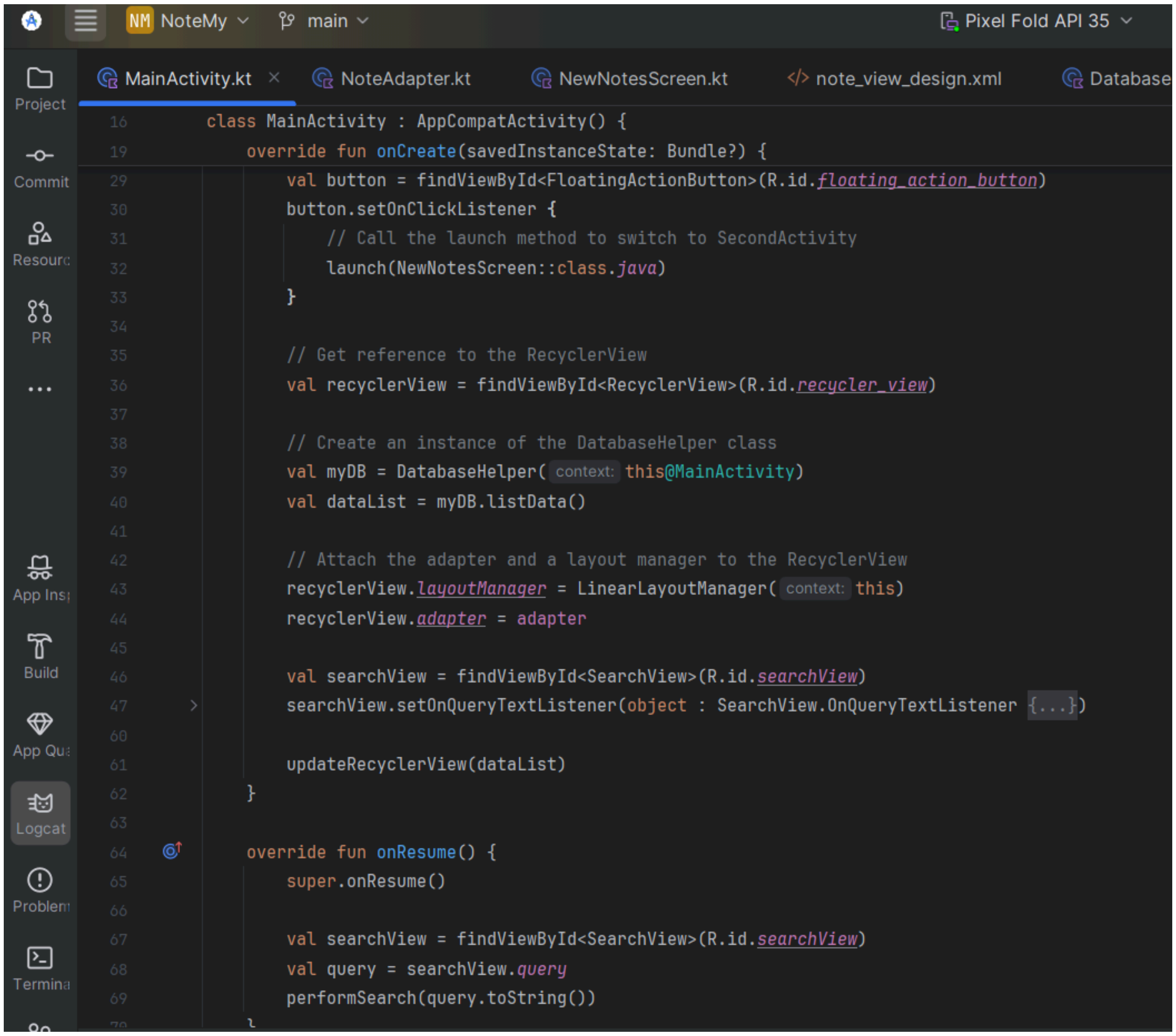
We really struggled to get the RecyclerView to work, it was very finicky and it is really difficult to debug adapters, we later determined that the error was caused by
1. The NestedScrollView layout_height was set to 0dp
2. You shouldn't update the RecyclerView directly, you should save the adapter used for the RecyclerView then update the adapters itemsList manually by adding an updateData method to the adapter.



```kotlin
class NoteAdapter(private var itemList: MutableList<Map<String, Any>>) : RecyclerView.Adapter<NoteAdapter.MyViewHolder>() {
    // ViewHolder class that holds reference to the item views
    class MyViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
        val title: TextView = itemView.findViewById(R.id.title)
        val subtitle: TextView = itemView.findViewById(R.id.subtitle)
        val description: TextView = itemView.findViewById(R.id.description)
        val card: CardView = itemView.findViewById(R.id.note_card)
    }

    // Inflate the layout for each item
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): MyViewHolder {
        val view = LayoutInflater.from(parent.context).inflate(R.layout.note_view_design, parent,  attachToRoot: false)
        return MyViewHolder(view)
    }

    // Bind data to the item view
    override fun onBindViewHolder(holder: MyViewHolder, index: Int) {...}

    // Return the number of items in the data list
    override fun getItemCount(): Int {
        return itemList.size
    }

    // Method to update the dataset
    fun updateData(newItems: MutableList<Map<String, Any>>) {
        itemList = newItems
        notifyDataSetChanged() // This refreshes the RecyclerView with new data
    }
}
```

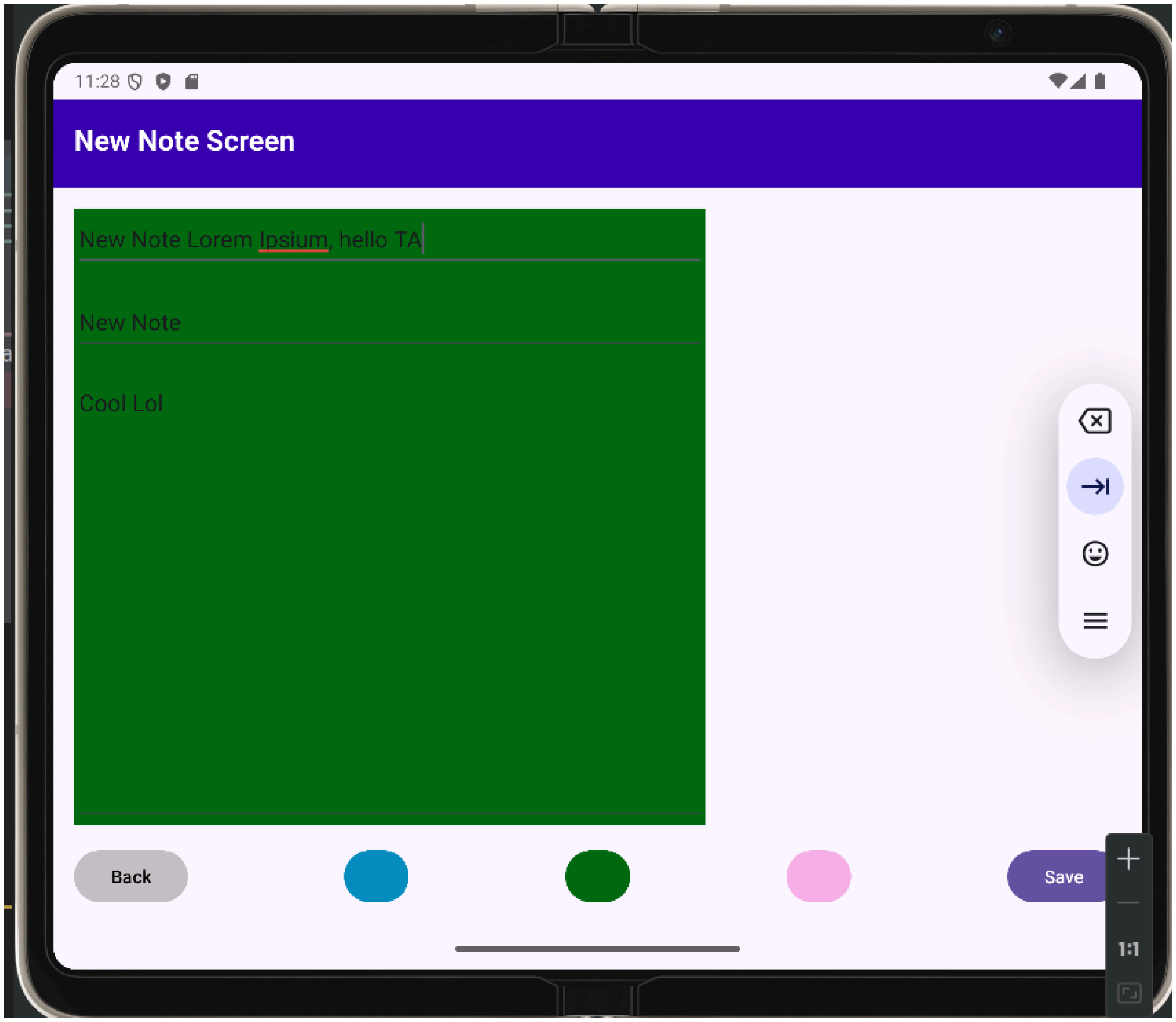


```kotlin
class MainActivity : AppCompatActivity() {
    override fun onResume() {
        val query = searchView.query
        performSearch(query.toString())
    }

    private fun performSearch(query: String) {
        val myDB = DatabaseHelper( context: this@MainActivity)
        val searchResults = (
            if (query.isEmpty()) myDB.listData()
            else myDB.searchData(query)
        );

        updateRecyclerView(searchResults)
    }

    private fun updateRecyclerView(data: MutableList<Map<String, Any>>) {
        val recyclerView = findViewById<RecyclerView>(R.id.recycler_view)
        val emptyView = findViewById<TextView>(R.id.empty_view)

        adapter.updateData(data)

        if (adapter.itemCount <= 0) {
            recyclerView.visibility = RecyclerView.GONE
            emptyView.visibility = TextView.VISIBLE
        } else {
            recyclerView.visibility = RecyclerView.VISIBLE
            emptyView.visibility = TextView.GONE
        }
    }
```



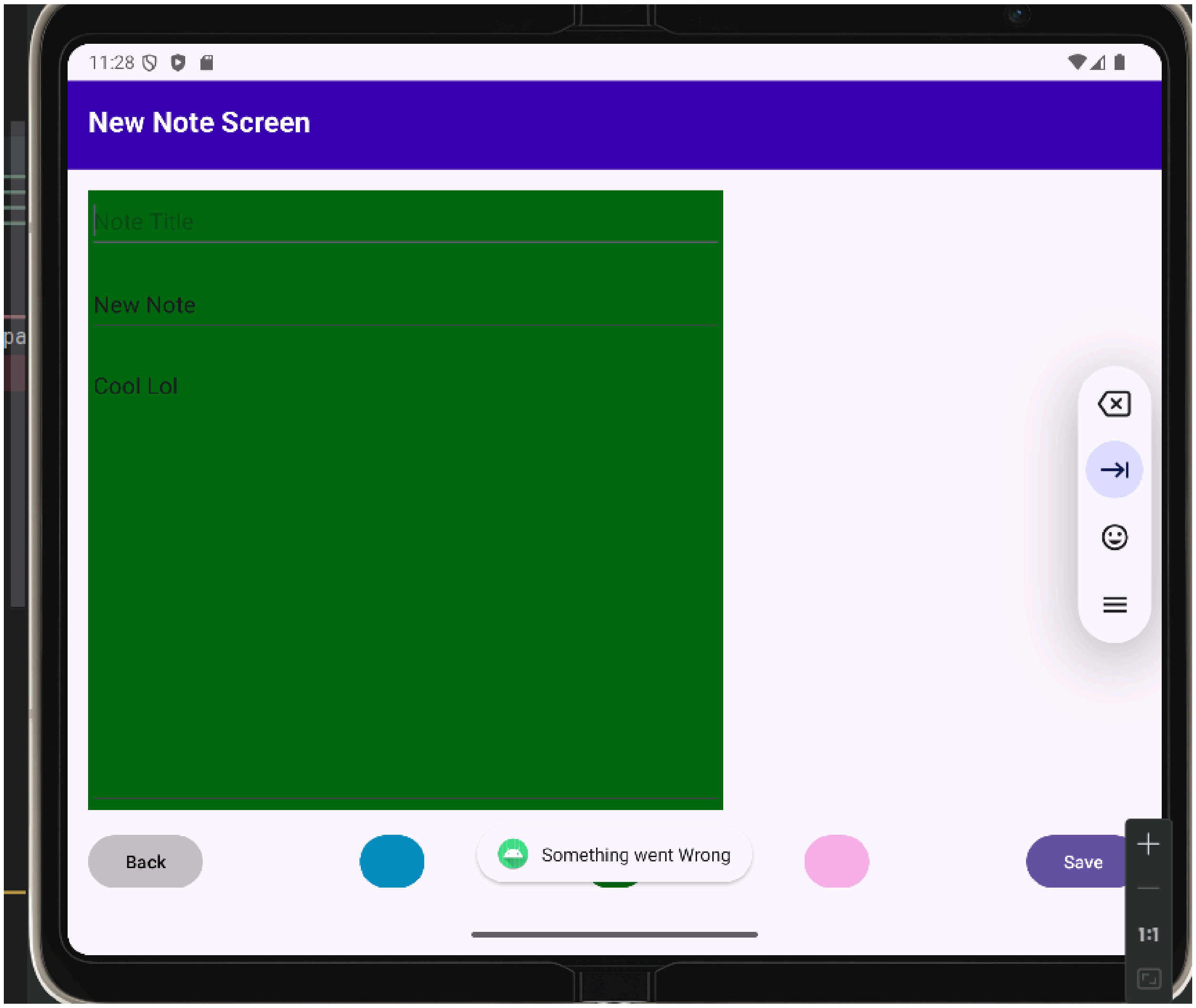
```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        val button = findViewById<FloatingActionButton>(R.id.floating_action_button)
        button.setOnClickListener {
            // Call the launch method to switch to SecondActivity
            launch(NewNotesScreen::class.java)
        }

        // Get reference to the RecyclerView
        val recyclerView = findViewById<RecyclerView>(R.id.recycler_view)

        // Create an instance of the DatabaseHelper class
        val myDB = DatabaseHelper( context: this@MainActivity)
        val dataList = myDB.listData()

        // Attach the adapter and a layout manager to the RecyclerView
        recyclerView.layoutManager = LinearLayoutManager( context: this)
        recyclerView.adapter = adapter

        val searchView = findViewById<SearchView>(R.id.searchView)
        searchView.setOnQueryTextListener(object : SearchView.OnQueryTextListener {...})

        updateRecyclerView(dataList)
    }

    override fun onResume() {
        super.onResume()

        val searchView = findViewById<SearchView>(R.id.searchView)
        val query = searchView.query
        performSearch(query.toString())
```

New Notes Screen

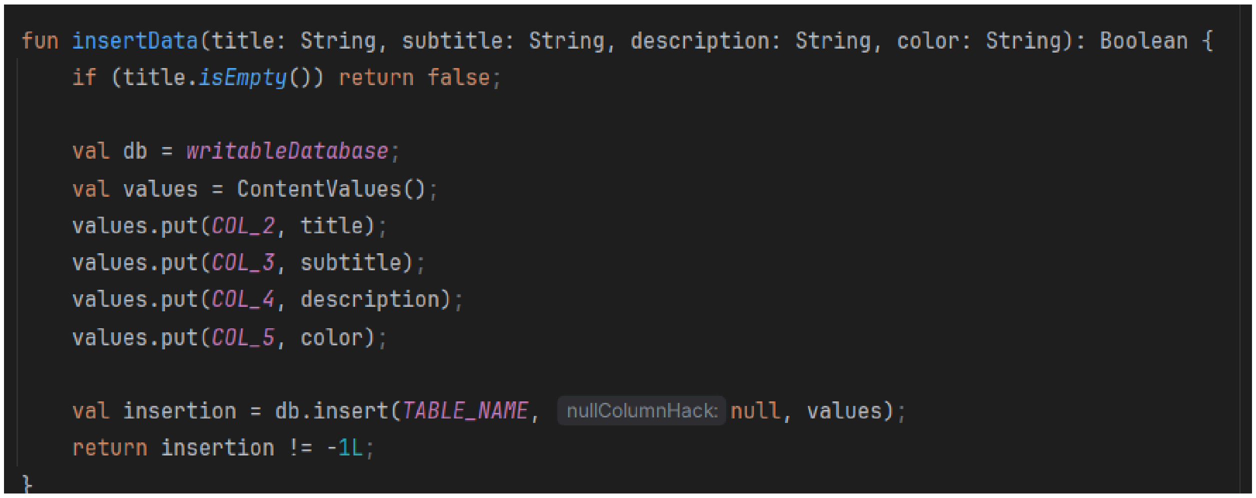


This is where you can edit the note and pick the color of it. You can choose the title, the subtitle and the context of the note.

Empty New Note Title Error



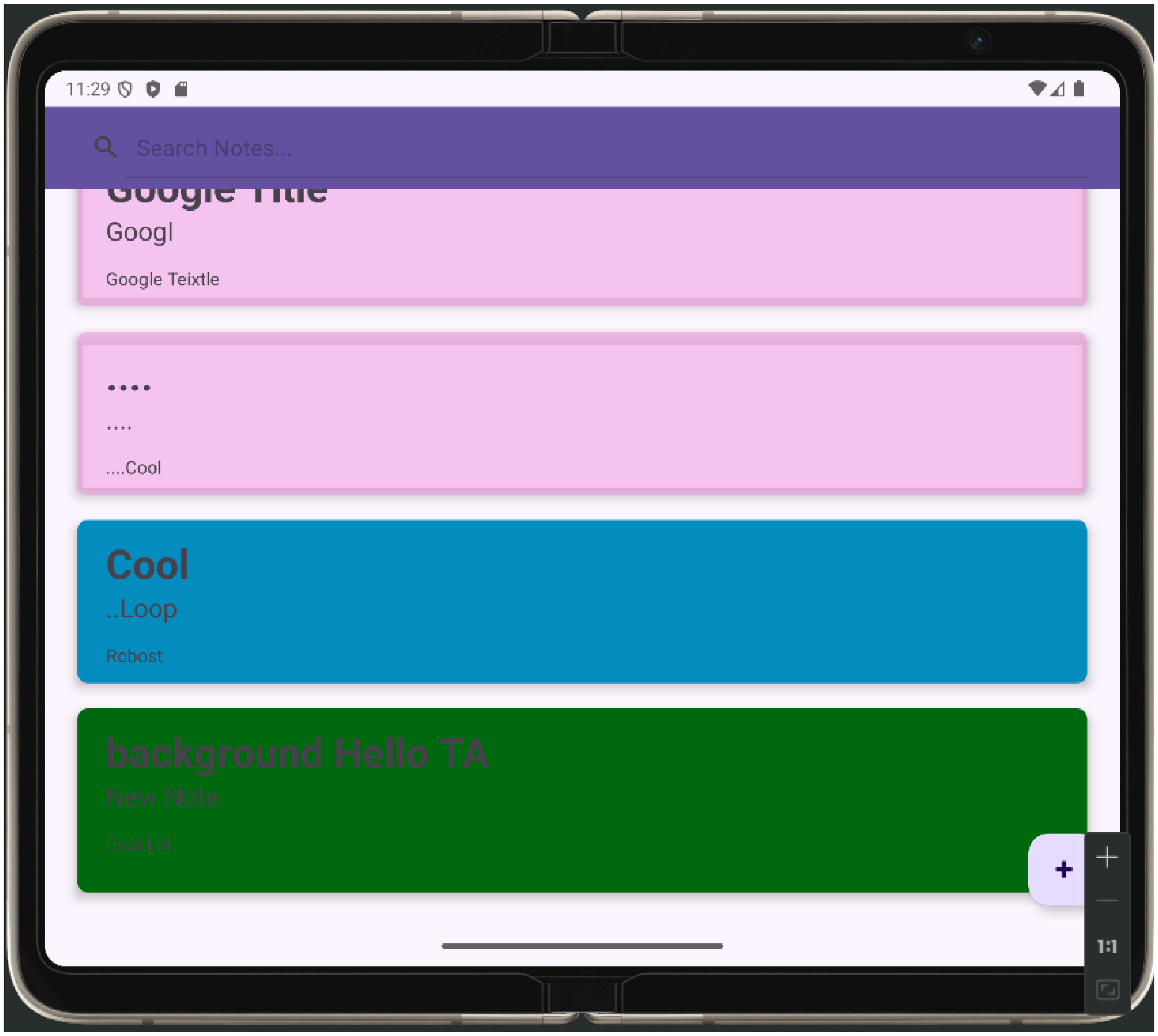
This screenshot displays an error message if the title of the note is not written/is empty. The note must have a title.



```
fun insertData(title: String, subtitle: String, description: String, color: String): Boolean {
    if (title.isEmpty()) return false;

    val db = writableDatabase;
    val values = ContentValues();
    values.put(COL_2, title);
    values.put(COL_3, subtitle);
    values.put(COL_4, description);
    values.put(COL_5, color);

    val insertion = db.insert(TABLE_NAME,  nullColumnHack: null, values);
    return insertion != -1L;
}
```

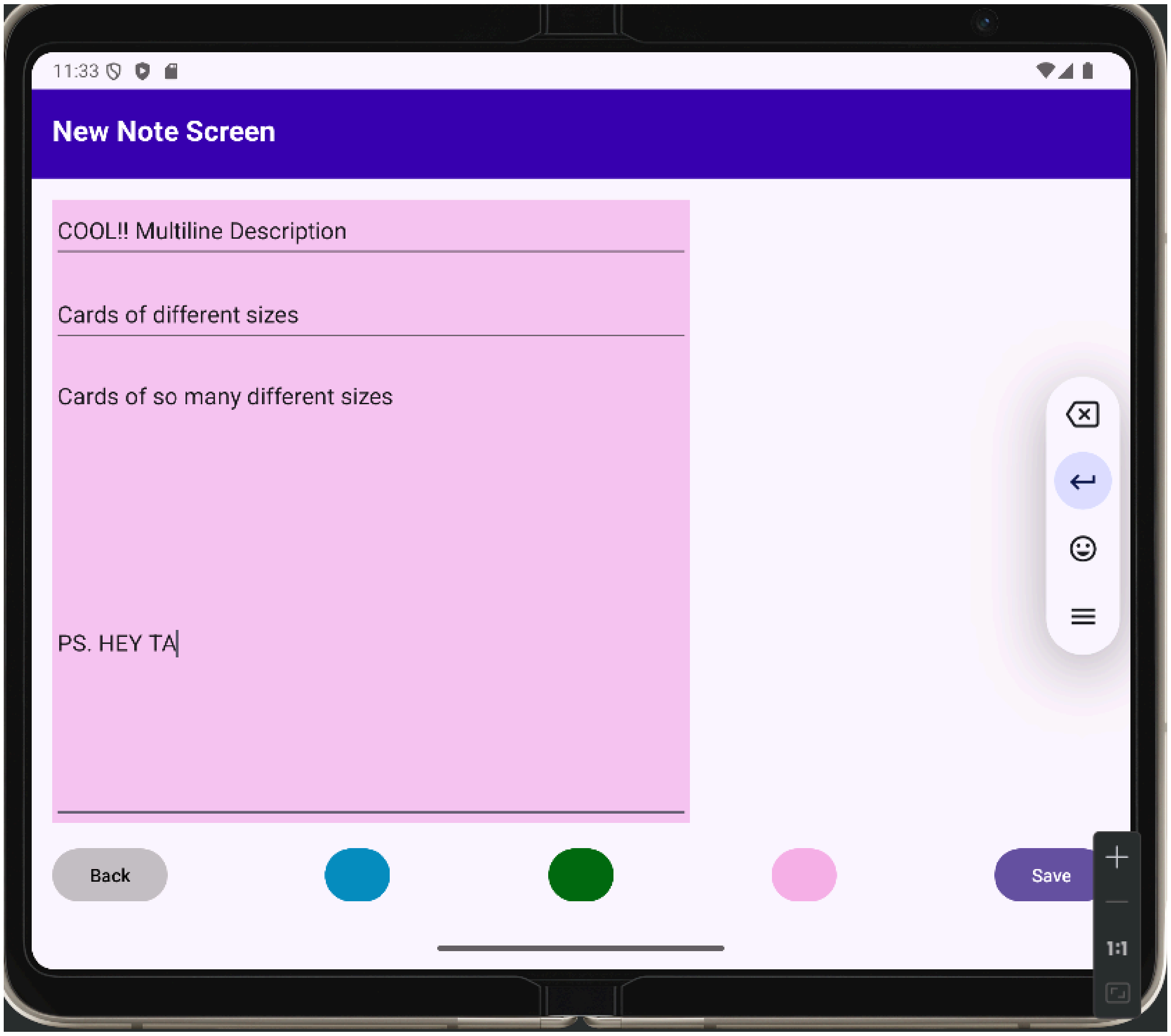
We basically just check if the title is empty, if it is we return false, which represents that the error didn't go through properly.

List View when a new Note is Added



This shows all of the notes that are added and with different colors. This is what the user would see if he added notes and wanted to see them. The button on the right is the add button.

Multiple Line Card Descriptions



COOL!! Multiline Description

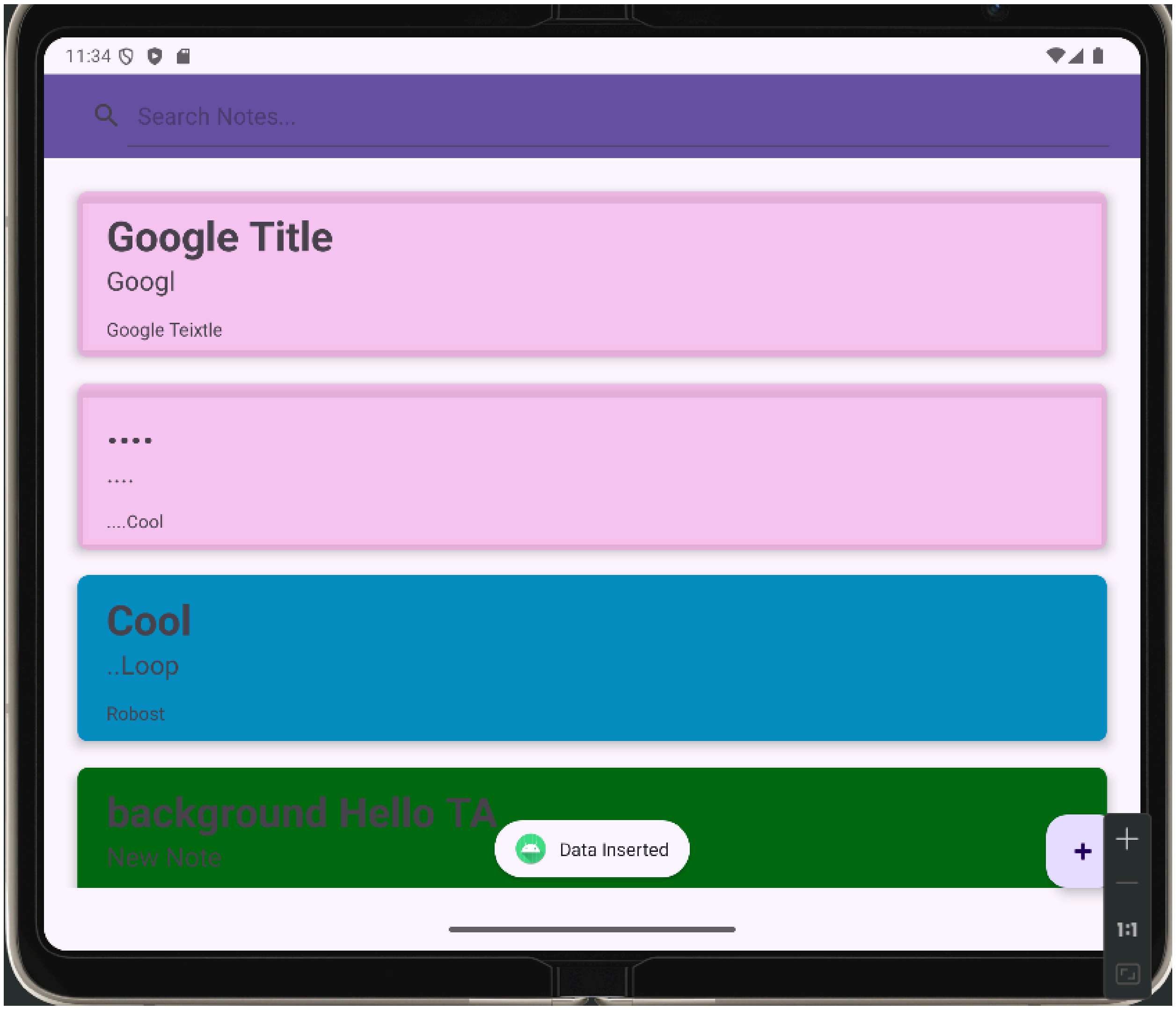
Cards of different sizes

Cards of so many different sizes

PS. HEY TA

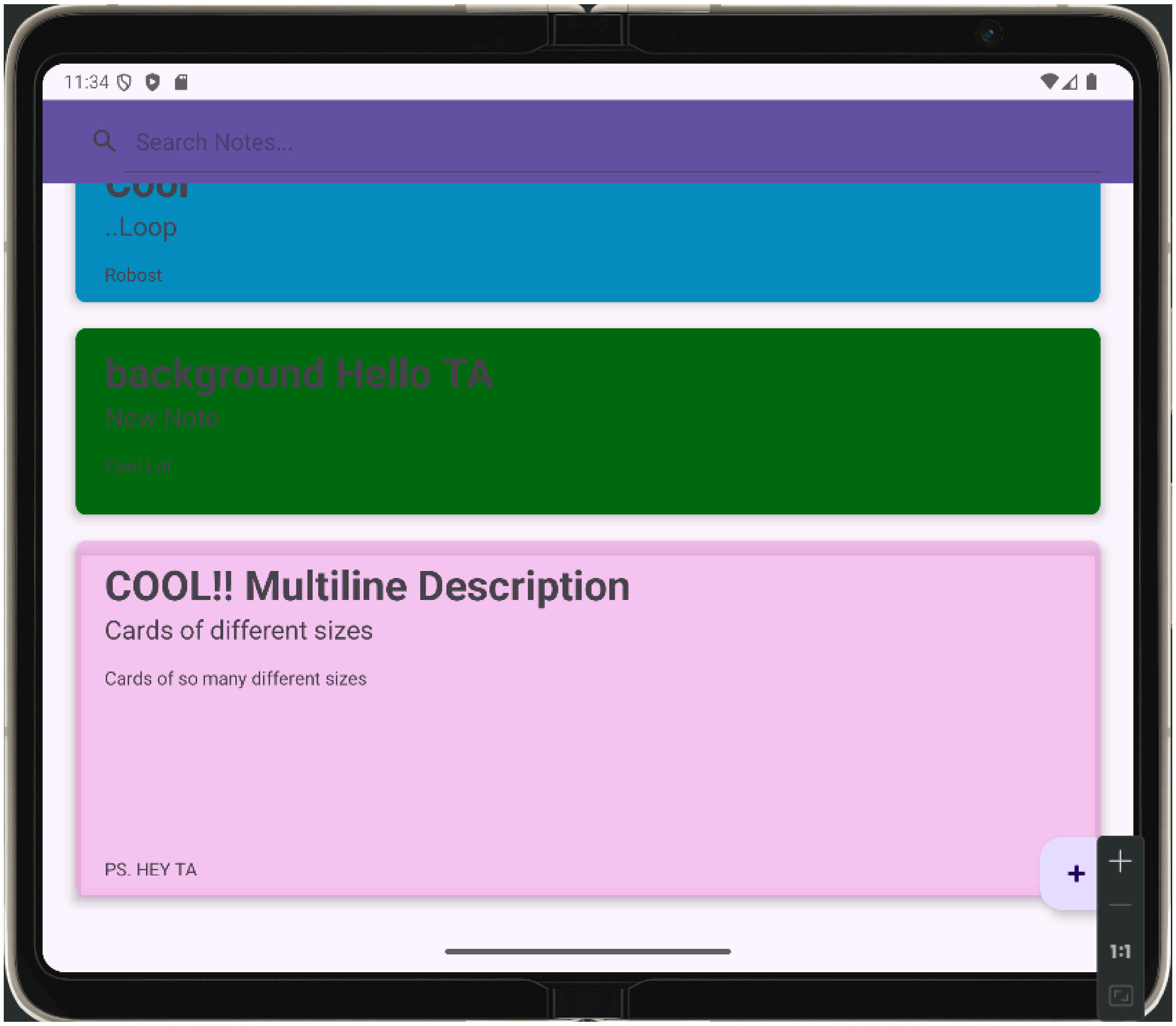
This is a multiple like card description. The functionality is simple and is working

New Note Inserted Toast Message



A toast message(a pop up) is shown when a new note/data is inserted

Different Sized Notes


11:34

Search Notes...

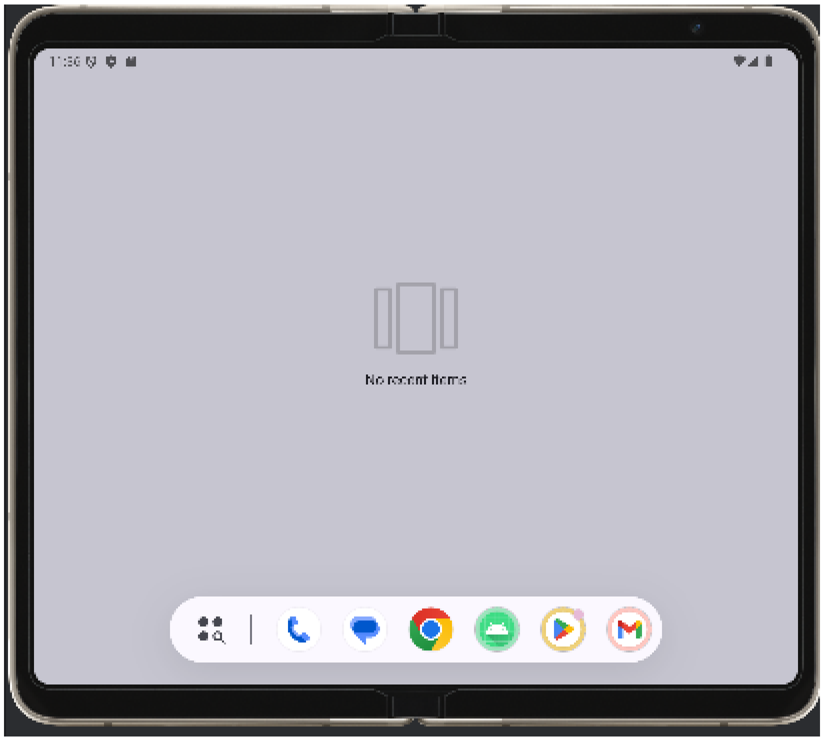
Cool
..Loop
Robost

background Hello TA
New Note
Cool Lol

COOL!! Multiline Description
Cards of different sizes
Cards of so many different sizes
PS. HEY TA

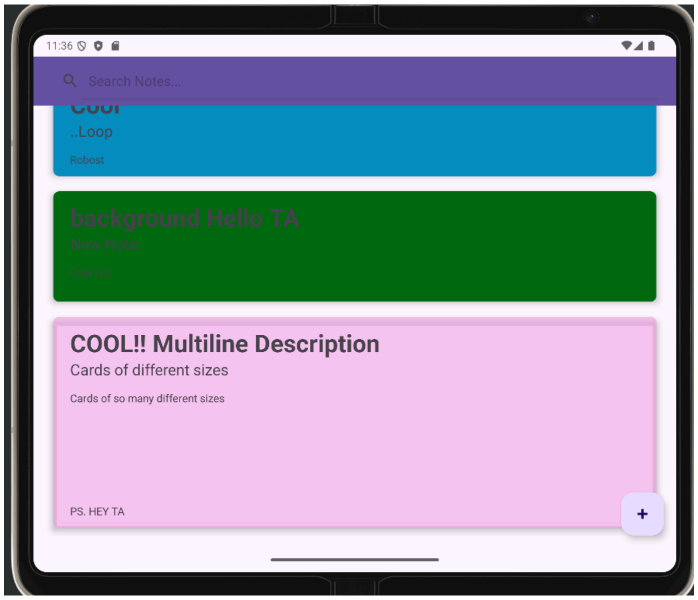
Here you can see different sizes of notes due to multi line notes shown previously. This is a functionality that was requested in the lab.

Closed App



This is the closed app screen.

Re-opened App, data still remains



..Loop

Robost

background Hello TA

New Note

Cool Lol

COOL!! Multiline Description

Cards of different sizes

Cards of so many different sizes

PS. HEY TA

This is the view that you see when you open the app. You will immediately see the notes that you have written previously