Mobile Assignment 2

Okiki Ojo (100790236)
CRN: 44434

GitHub: https://github.com/okikio-school/mobile-app-assignments/tree/main/Assignment2

For the pre-filled locations a friend generated a good list of pre-existing locations within the GTA, I decided to take advantage of that list, it is listed in the repo within the **res/raw/** directory, it is called the **gta_locations.csv**.

This assignment is a clone of the NoteMy Labs, so I will focus on the core changes I made to ensure the assignment goals were met.

First, to ensure the database was seeded correctly, I created a new table called Flags, it stores all the the settings regarding whether the database has been already seeded with data from the CSV, If not we read the seed data from the CSV and seeds the locations table with new gta locations.

```kotlin
/**
 * Reads and parses a CSV file from the res/raw folder.
 *
 * @param context The Android context to access resources.
 * @param fileName The resource ID of the CSV file (e.g., R.raw.data).
 * @return A list of rows, where each row is a list of strings representing columns.
 */
private fun readCsvFromRaw(context: Context, fileName: Int): List<List<String>> {
    val result = mutableListOf<List<String>>()
    // Open the resource as an InputStream
    val inputStream = context.resources.openRawResource(fileName)
    BufferedReader(InputStreamReader(inputStream)).use { reader ->
        // Read each line from the CSV
        reader.forEachLine { line ->
            // Split by commas and trim spaces
            val row = line.split(",").map { it.trim() }
            result.add(row)
        }
    }
    return result
}
```

The basic logic is as such, create a virtual table for the locations table to makes it easier to index for Fuzzy searching later on down the line, we then also create the flags table where we

store whether the database was seeded or not.

```
class DatabaseHelper(private val 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 $LOCATIONS_TABLE USING fts4($COL_ADDR, $COL_LONGITUDE, $COL_LATITUDE)");

        // Create a flags table to track if the database has been seeded
        db.execSQL( sql: "CREATE TABLE IF NOT EXISTS $FLAGS_TABLE ($FLAG_KEY_COLUMN TEXT PRIMARY KEY, $FLAG_VALUE_COLUMN TEXT)");

        // Check if the database is already seeded
        if (!isDatabaseSeeded(db)) {
            // Seed the database from the CSV file
            seedDatabaseFromCsv(db, R.raw.gta_locations)

            // Mark database as seeded
            flagDatabaseAsSeeded(db)
        }
    }
```

The flags table is a set of flags (pun intended) to indicate whether the locations table has been seeded. We create a flag key called seeded which would store the flag value of “true” when the database is seeded. We can then check if the seeded flag key exists and/or if it is true to determine if we should seed the database.

```
/**
 * Checks if the database has already been seeded.
 *
 * @param db The writable database instance.
 * @return True if the database is already seeded, false otherwise.
 */
private fun isDatabaseSeeded(db: SQLiteDatabase): Boolean {
    val cursor = db.rawQuery( sql: "SELECT $FLAG_VALUE_COLUMN FROM $FLAGS_TABLE WHERE $FLAG_KEY_COLUMN = 'seeded'",  selectionArgs: null)
    val isSeeded = cursor.moveToFirst() && cursor.getString( columnIndex: 0) == "true"
    cursor.close()
    return isSeeded
}

/**
 * Marks the database as seeded by inserting a flag into the metadata table.
 *
 * @param db The writable database instance.
 */
private fun flagDatabaseAsSeeded(db: SQLiteDatabase) {
    val values = ContentValues().apply {
        put(FLAG_KEY_COLUMN, "seeded")
        put(FLAG_VALUE_COLUMN, "true")
    }
    db.insert(FLAGS_TABLE,  nullColumnHack: null, values)
}
```

During Seeding from the CSV we actually have to skip the first row as the first row contains the headers of the CSV.

```kotlin
/**
* Seeds the database with data from a CSV file in the res/raw directory.
*
* @param db The writable database instance to insert data into.
* @param csvResourceId The resource ID of the CSV file (e.g., R.raw.gta_locations).
*/
private fun seedDatabaseFromCsv(db: SQLiteDatabase, csvResourceId: Int) {
    val csvData = readCsvFromRaw(context, csvResourceId)
    // Skip the header row
    for (i in 1 ≤ until < csvData.size) {
        val row = csvData[i]
        // Ensure the row has enough data fields (address, longitude, latitude)
        if (row.size >= 3) {
            val contentValues = ContentValues().apply {
                put(COL_ADDR, expandAddrsAbbriviations(row[0])) // Expand address abbreviations
                put(COL_LONGITUDE, row[1])
                put(COL_LATITUDE, row[2])
            }
            db.insert(LOCATIONS_TABLE, nullColumnHack: null, contentValues)
        }
    }
}
```

To ensure Fuzzy search works correctly we have to expand the common location abbreviations used in maps, e.g. N = North, Blvd = Boulevard, St = Street, etc… I created a list of some of the most commonly used abbreviations in Canada, I then split the street addresses by their spaces and checked to see if the abbreviation is used, if so replace the usage with the expanded form.

```
// Used to identify if the database has been seeded
private const val FLAGS_TABLE = "FLAGS"
private const val FLAG_KEY_COLUMN = "FLAG"
private const val FLAG_VALUE_COLUMN = "VALUE"
private val ADDRS_ABBRIVIATIONS = mapOf(
    // Directional Abbreviations
    "n" to "North",
    "s" to "South",
    "e" to "East",
    "w" to "West",
    "ne" to "Northeast",
    "nw" to "Northwest",
    "se" to "Southeast",
    "sw" to "Southwest",

    // Street Type Abbreviations
    "rd" to "Road",
    "st" to "Street",
    "ave" to "Avenue",
    "blvd" to "Boulevard",
    "dr" to "Drive",
    "pl" to "Place",
    "ct" to "Court",
    "ln" to "Lane",
```

```
/**
 * Expands abbreviations in the address based on the ADDRS_ABBRIVIATIONS map.
 *
 * @param address The raw address string with abbreviations.
 * @return The expanded address string.
 */
fun expandAddrsAbbriviations(address: String): String {
    // Split the address into individual words
    val words = address.split( ...delimiters: " ")
    val expandedWords = mutableListOf<String>()

    // Loop through each word to check for abbreviations
    for (word in words) {
        // Convert the word to lowercase for case-insensitive matching
        val lower = word.lowercase()

        // If the word is in ADDRS_ABBRIVIATIONS, replace it; otherwise, keep it as is
        val expandedWord = if (ADDRS_ABBRIVIATIONS.containsKey(lower)) {
            ADDRS_ABBRIVIATIONS[lower]!!
        } else {
            word
        }
        expandedWords.add(expandedWord)
    }

    // Join all expanded words back into a single string with spaces
    val resultAddress = expandedWords.joinToString( separator: " ")
    return resultAddress
}
```

We then make modifications to the onUpgrade method to ensure the flags table gets deleted when the locations table gets deleted.

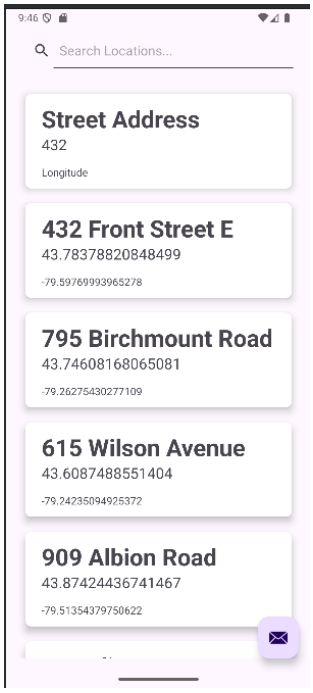
```
override fun onUpgrade(db: SQLiteDatabase, old: Int, new: Int) {
    db.execSQL( sql: "DROP TABLE IF EXISTS $LOCATIONS_TABLE");
    db.execSQL( sql: "DROP TABLE IF EXISTS $FLAGS_TABLE");
    onCreate(db);
}
```

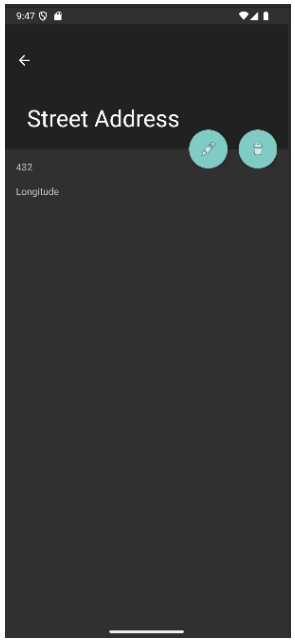
For layouts I used constraint layout to group items together almost like a container for a set of items, I then used padding to create space between all the items.

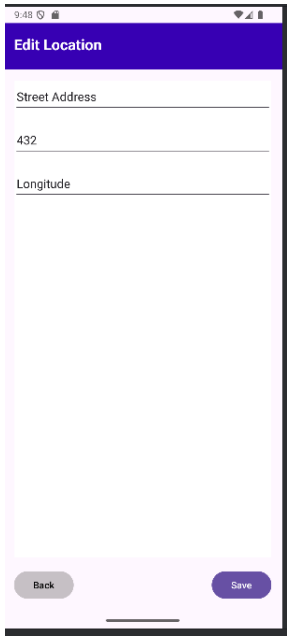
For intent, I modified the floating action button to open the new location page using an intent.

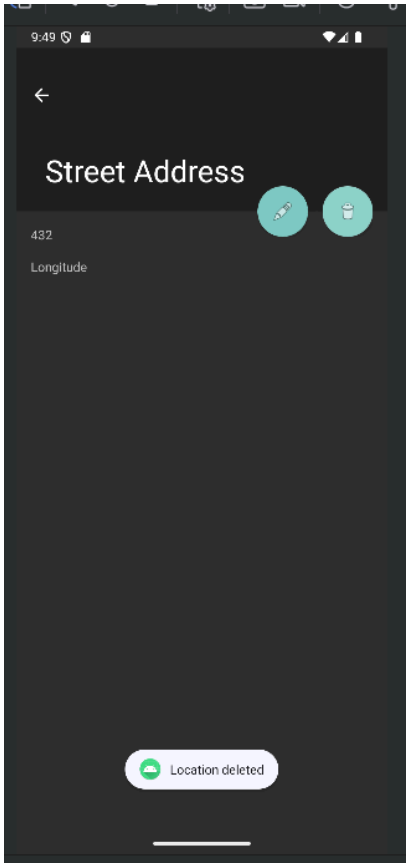
I added support for the long press to ensure that users can edit/delete locations directly,

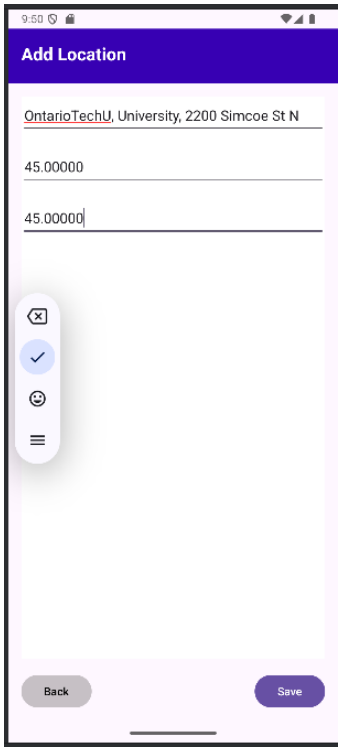
Fuzzy search to identify locations using abbreviations

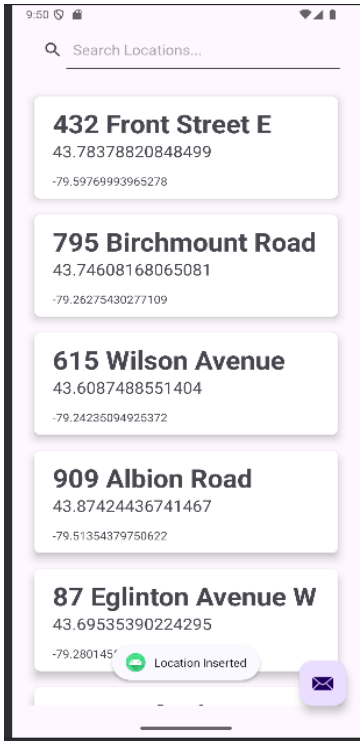

Search Locations...
Street Address
432
Longitude
432 Front Street E
43.78378820848499
-79.59769993965278
795 Birchmount Road
43.74608168065081
-79.26275430277109
615 Wilson Avenue
43.6087488551404
-79.24235094925372
909 Albion Road
43.87424436741467
-79.51354379750622

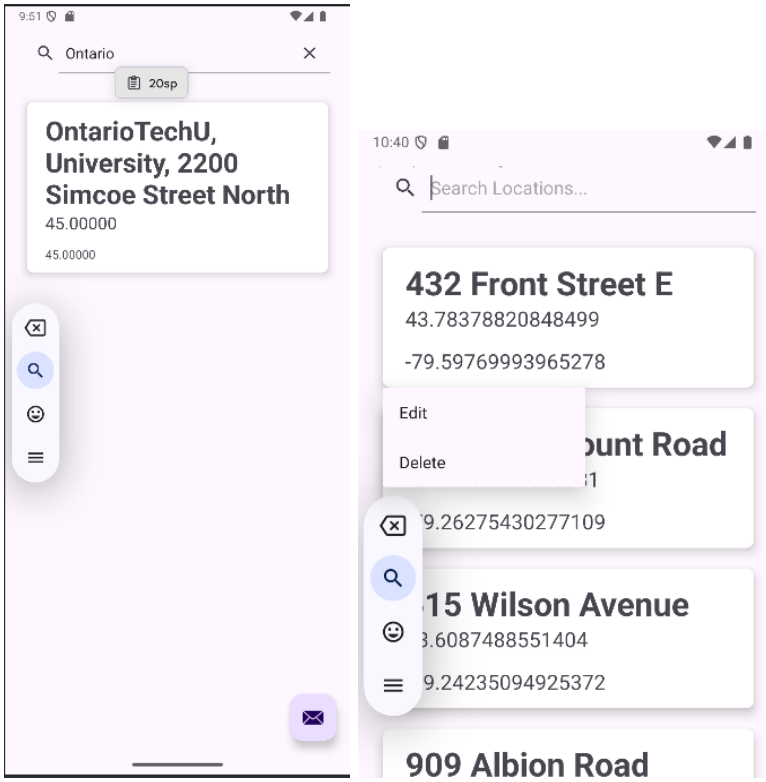

Street Address
432
Longitude


Edit Location
Street Address
432
Longitude
Back
Save


Street Address
432
Longitude
Location deleted


Add Location
OntarioTechU, University, 2200 Simcoe St N
45.00000
45.00000
Back
Save


Search Locations...
432 Front Street E
43.78378820848499
-79.59769993965278
795 Birchmount Road
43.74608168065081
-79.26275430277109
615 Wilson Avenue
43.6087488551404
-79.24235094925372
909 Albion Road
43.87424436741467
-79.51354379750622
87 Eglinton Avenue W
43.69535390224295
Location Inserted



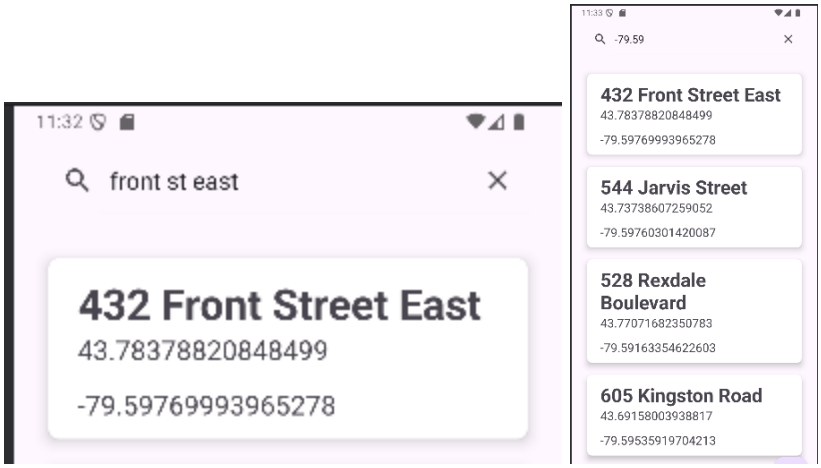

For the most part due to the basis of the assignment being the lab, the assignment was pretty easy to do. I did notice a couple areas of improvement in the implementation of search such that fuzzy search was more forgiving.

Specifically, I replaced all whitespace with the wildcard character to allow for better partial matches. I also added a subquery option to match unabbreviated searches, including longitude and latitude.

```
// Search query using the MATCH operator to perform full-text search
fun searchData(queryString: String): MutableList<Map<String, Any>> {
    val unabbriviatedQuery = expandAddrsAbbriviations(queryString);

    // Expand the search query to include wildcard characters this should allow for better partial matching
    val expansiveSearch = "%" + queryString.replace(Regex( pattern: "\\s+"), replacement: "%") + "%"
    val unabbriviatedSearch = "%" + unabbriviatedQuery.replace(Regex( pattern: "\\s+"), replacement: "%") + "%"
    println("Search query: $queryString -> $expansiveSearch -> $unabbriviatedQuery -> $unabbriviatedSearch")

    return query("SELECT $HIDDEN_COL as $COL_ID, * FROM $LOCATIONS_TABLE WHERE $COL_ADDR LIKE ? OR $COL_ADDR LIKE ? OR $COL_LONGITUDE LIKE ? OR $COL_LATITUDE LIKE ?",
        arrayOf(expansiveSearch, unabbriviatedSearch, expansiveSearch, expansiveSearch))
}
```


11:32
front st east
432 Front Street East
43.78378820848499
-79.59769993965278
11:33
-79.59
432 Front Street East
43.78378820848499
-79.59769993965278
544 Jarvis Street
43.73738607259052
-79.59760301420087
528 Rexdale Boulevard
43.77071682350783
-79.59163354622603
605 Kingston Road
43.69158003938817
-79.59535919704213