Demo 11: BaseDAtos_GPS_SintesisRecVoz-ETC for Jetpack Compose

Mobile Programming

Antonio Isai López Leal

Polytechnic University of Victoria

September - December 2024

Abstract

- **Add Tutor:** Inserts a new tutor into the SQLite database, using a name provided by the user.
- **Delete Tutor:** Removes a tutor by ID. If the ID field is empty, it displays an error message indicating Field is 'empty'.
- **Update Tutor:** Updates the name of a tutor by ID, modifying the tutor's name if both fields are filled.
- **View Tutors:** Retrieves a list of all tutors in the format Tutor X: 'Name (ID: Y)' and displays it in a vertical list.
- **Add Tutored Student:** Associates a 'tutorado' (student) with a specified tutor ID, given both fields are provided.
- **View Tutored Students by Tutor:** Lists all students associated with a specific tutor ID in a format similar to Tutored Student X: 'Name'.

Original XML Code - activity_main.xml (1)

```
ScrollView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    >

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity" >
    <TextView
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"
        android:id="@+id/TV4"
        android:text="NOMBRE DEL TUTOR" />
    <EditText
        android:id="@+id/ET2"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="" />
    <TextView
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"
        android:id="@+id/TV5"
        android:text="ID TUTOR ACTUALIZAR/MODIFICAR" />
```

- The layout starts with a `ScrollView` to enable vertical scrolling, containing a vertically oriented `LinearLayout`.
- At the top, there is a `TextView` labeled *NOMBRE DEL TUTOR* followed by an `EditText` (ID: `ET2`) for entering the tutor's name.
- Another `TextView` labeled *ID TUTOR ACTUALIZAR/MODIFICAR* indicates the input field for tutor ID.

Original XML Code - activity_main.xml (2)

```
<EditText
    android:id="@+id/ET1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="" />
<TextView
    android:layout_height="wrap_content"
    android:layout_width="fill_parent"
    android:id="@+id/TV6"
    android:text="Nombre del Tutorado" />
 <EditText
    android:id="@+id/ET3"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="0" />
            <Button
                android:layout_width="fill_parent"
                android:layout_height="fill_parent"
                android:text="Insertar Tutor"
                android:id="@+id/BT01"   />
            <Button
                android:layout_width="fill_parent"
                android:layout_height="fill_parent"
                android:text="Borrar Tutor"
                android:id="@+id/BT02"   />
```

- An `EditText` (ID: `ET1`) is placed next to accept the tutor's ID, followed by a `TextView` labeled *Nombre del Tutorado*.
- Another `EditText` (ID: `ET3`) is provided below to enter the tutored person's name.
- This section is followed by a series of buttons for different actions, starting with *Insertar Tutor* (ID: `BT01`) and *Borrar Tutor* (ID: `BT02`).

Original XML Code - activity_main.xml (3)

```
        <Button
                       android:layout_width="fill_parent"
                       android:layout_height="fill_parent"
                       android:text="Consultar Tutores"
                       android:id="@+id/BT03"   />
                   <Button
                       android:layout_width="fill_parent"
                       android:layout_height="fill_parent"
                       android:text="Actualizar Tutor"
                       android:id="@+id/BT04"   />
                   <Button
                       android:layout_width="fill_parent"
                       android:layout_height="fill_parent"
                       android:text="Agregar Tutorado"
                       android:id="@+id/BT05"   />
                   <Button
                       android:layout_width="fill_parent"
                       android:layout_height="fill_parent"
                       android:text="Consultar Tutorados por Tutor"
                       android:id="@+id/BT06"   />
           <Button
                   android:layout_width="fill_parent"
                   android:layout_height="fill_parent"
                   android:text="Otra ACtividad"
                   android:id="@+id/BT07"   />
           <TextView
           android:layout_height="wrap_content"
           android:layout_width="fill_parent"
           android:id="@+id/TV2"
           android:text="CONTENIDO DEL TEXTIVIEW" />
     </LinearLayout>
     </ScrollView>
```

- Additional buttons allow for further actions like *Consultar Tutores*, *Actualizar Tutor*, and more, each with a unique ID (e.g., `BT03`, `BT04`).
- At the bottom, a `TextView` (ID: `TV2`) displays the content or results of operations.

Original XML Code - activity_main2.xml

```
<ScrollView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    >

<LinearLayout

    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity" >

                <Button
                    android:layout_width="fill_parent"
                    android:layout_height="fill_parent"
                    android:text="Insertar Tutor"
                    android:id="@+id/BT01_AC2"   />
        <TextView
                android:id="@+id/TV_MARCIANO1"
                android:layout_width="match_parent"
                android:layout_height="match_parent" />
</LinearLayout>
</ScrollView>
```

- The layout is structured to make interactions intuitive, with each button handling a specific operation.

Composable Functions and Compose Modifiers in Android

Composable Functions:

- A Composable function is a modern way to build UI in Android, written directly in Kotlin.
- It is annotated with `@Composable`, which signals to the Compose compiler that the function is for UI construction.
- Unlike traditional XML layouts, Composable functions allow for a more flexible, programmatic approach to UI development.
- In our project, `TutorScreen`, `ActionButton`, and various UI elements like `TextField` are implemented as Composable functions.

Compose Modifiers:

- Compose modifiers allow you to decorate or enhance a Composable, controlling its size, layout, and behavior.
- For example, in our project, we use `Modifier.fillMaxWidth()` and `Modifier.padding()` to adjust the width and padding of UI elements.
- Modifiers can also handle interactions, making elements clickable, scrollable, or draggable.
- They allow us to enrich the UI with functionalities like accessibility labels and processing user input.

Main Activity of a Kotlin App using Composables

```
class MainActivity : ComponentActivity() {
    private lateinit var databaseHelper: TutorDatabaseHelper

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        databaseHelper = TutorDatabaseHelper(this)
        setContent {
            TutorScreen(databaseHelper)
        }
    }
}
```

MainScreen Composable (1) - MainActivity.kt

```kotlin
package com.z_iti_271311_u1_ae_lopez_leal_antonio_isai

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.material3.Button
import androidx.compose.material3.Text
import androidx.compose.material3.TextField
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.text.input.TextFieldValue
import androidx.compose.ui.unit.dp
import com.z_iti_271311_u1_ae_lopez_leal_antonio_isai.data.TutorDatabaseHelper
```

- Create a new project in Android Studio using "Empty Compose Activity" as the template.
- In the `kotlin+java` directory, create a new package named `data` to organize database-related files.
- Inside the `data` package, add a file named `TutorDatabaseHelper.kt` to define database operations for managing tutors and students.
- Open `MainActivity.kt` and replace its content with the main UI and logic setup to enable CRUD operations and display results.

MainScreen Composable (2) - MainActivity.kt

```kotlin
@Composable
fun TutorScreen(databaseHelper: TutorDatabaseHelper) {
    val tutorName = remember { mutableStateOf(TextFieldValue()) }
    val tutorId = remember { mutableStateOf(TextFieldValue()) }
    val tutoredName = remember { mutableStateOf(TextFieldValue("0")) }

    var resultList by remember { mutableStateOf(listOf<String>()) }

    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(16.dp),
        horizontalAlignment = Alignment.CenterHorizontally
    ) {

        TextField(
            value = tutorName.value,
            onValueChange = { tutorName.value = it },
            label = { Text("Nombre del Tutor") },
            modifier = Modifier.fillMaxWidth()
        )
        Spacer(modifier = Modifier.height(8.dp))

        TextField(
            value = tutorId.value,
            onValueChange = { tutorId.value = it },
            label = { Text("ID Tutor Actualizar/Modificar") },
            modifier = Modifier.fillMaxWidth()
        )
        Spacer(modifier = Modifier.height(8.dp))
```

- **@Composable fun TutorScreen(databaseHelper: TutorDatabaseHelper)** - A composable function that displays a screen for managing tutors and students (tutorados) with a provided `databaseHelper` object for database operations. It includes input fields and a list to display results from database actions.
- **State Variables:**
  - `tutorName`: A mutable state holding the name of the tutor entered by the user. Initialized as an empty `TextFieldValue`.
  - `tutorId`: A mutable state for the tutor's ID, used for updating or deleting records. Also initialized as an empty `TextFieldValue`.
  - `tutoredName`: Holds the name of the student (tutorado) associated with a tutor, initialized with the default value of `"0"`.
  - `resultList`: A mutable state containing the list of strings used to display the result of database operations (e.g., successful insertion or deletion).

MainScreen Composable (3) - MainActivity.kt

```
TextField(
        value = tutoredName.value,
        onValueChange = { tutoredName.value = it },
        label = { Text("Nombre del Tutorado") },
        modifier = Modifier.fillMaxWidth()
    )
    Spacer(modifier = Modifier.height(16.dp))

    ActionButton("INSERTAR TUTOR") {
        if (tutorName.value.text.isEmpty()) {
            resultList = listOf("CAMPO VACÍO:
            Por favor ingresa el nombre del tutor.")
        } else {
            databaseHelper.insertTutor(tutorName.value.text)
            resultList = listOf("Tutor
            '${tutorName.value.text}' insertado.")
        }
    }
    ActionButton("BORRAR TUTOR") {
        if (tutorId.value.text.isEmpty()) {
            resultList = listOf("CAMPO VACÍO: Por favor ingresa el ID del tutor.")
        } else {
            databaseHelper.deleteTutor(tutorId.value.text.toLong())
            resultList = listOf("Tutor con ID ${tutorId.value.text} eliminado.")
        }
    }
```

UI Structure with Column:

- Uses `Column` to organize composables vertically, filling the screen's size with `Modifier.fillMaxSize()` and adding padding.
- Aligns all child composables horizontally at the center with `horizontalAlignment = Alignment.CenterHorizontally`.

MainScreen Composable (4) - MainActivity.kt

```
ActionButton("CONSULTAR TUTORES") {
            val tutors = databaseHelper.getTutors()
            if (tutors.isEmpty()) {
                resultList = listOf("No hay tutores para mostrar.")
            } else {
                resultList = tutors.mapIndexed { index, (id, name) -> "Tutor ${index + 1}: $name (ID: $id)" }
            }
        }
        ActionButton("ACTUALIZAR TUTOR") {
            if (tutorId.value.text.isEmpty()
            || tutorName.value.text.isEmpty()) {
                resultList = listOf("CAMPO VACÍO:
                Por favor ingresa el ID y
                el nombre del tutor.")
            } else {
                databaseHelper.updateTutor
                (tutorId.value.text.toLong(),tutorName.value.text)
                resultList = listOf("Tutor con ID ${tutorId.value.text}
                actualizado a '${tutorName.value.text}'.")
            }
        }
        ActionButton("AGREGAR TUTORADO") {
            if (tutoredName.value.text.isEmpty() || tutorId.value.text.isEmpty()) {
                resultList = listOf("CAMPO VACÍO: Por favor ingresa el nombre del tutorado y el ID del tutor.")
            } else {
                databaseHelper.addTutorado(tutoredName.value.text, tutorId.value.text.toLong())
                resultList = listOf("Tutorado '${tutoredName.value.text}' agregado al tutor con ID ${tutorId.value.text}.")
            }
        }
```

Input Fields:

- `TextField` for `tutorName`: An input field allowing the user to enter the tutor's name, with a label "Nombre del Tutor" and full width.
- `TextField` for `tutorId`: Another input field where the user enters the tutor's ID for modification or deletion purposes, labeled "ID Tutor Actualizar/Modificar".
- Adds `Spacer` elements between fields to add vertical spacing.

MainScreen Composable (5) - MainActivity.kt

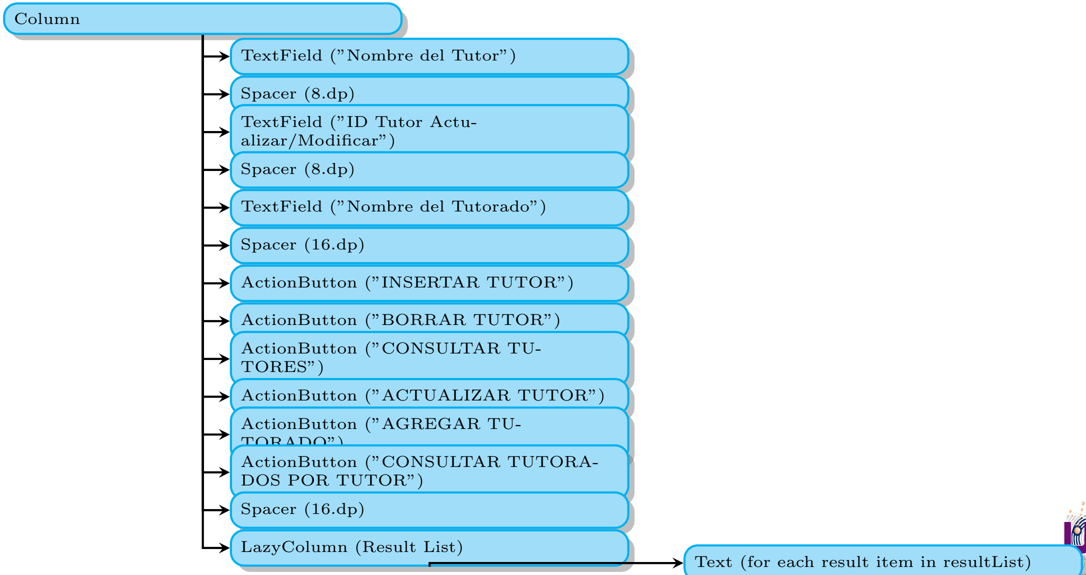
```
ActionButton("CONSULTAR TUTORADOS POR TUTOR") {
            if (tutorId.value.text.isEmpty()) {
                resultList = listOf("CAMPO VACÍO: Por favor ingresa el ID del tutor.")
            } else {
                val tutorados = databaseHelper.getTutoradosByTutor(tutorId.value.text.toLong())
                if (tutorados.isEmpty()) {
                    resultList = listOf("No hay tutorados para el tutor con ID ${tutorId.value.text}.")
                } else {
                    resultList = tutorados.mapIndexed { index, tutorado -> "Tutorado ${index + 1}: $tutorado" }
                }
            }
        }
        Spacer(modifier = Modifier.height(16.dp))

        LazyColumn(
            modifier = Modifier
                .fillMaxWidth()
                .padding(16.dp)
        ) {
            items(resultList) { item ->
                Text(text = item, modifier = Modifier.padding(vertical = 4.dp))
            }
        }
    }
}
```

MainScreen Composable (6) - MainActivity.kt

```kotlin
@Composable
fun ActionButton(text: String, onClick: () -> Unit) {
    Button(
        onClick = onClick,
        modifier = Modifier
            .fillMaxWidth()
            .padding(vertical = 4.dp)
    ) {
        Text(text)
    }
}
```

- **@Composable fun ActionButton(text: String, onClick: () -¿ Unit)** - A reusable composable function that creates a button with customizable text and action. The button is styled to occupy the full width of its container with vertical padding for spacing.
- **Parameters:**
  - `text`: A `String` parameter that defines the label displayed on the button.
  - `onClick`: A lambda function (`() -> Unit`) that specifies the action to perform when the button is clicked.
- **Button Component:**
  - `Button`: The main clickable component. It triggers the `onClick` action passed as a parameter.
  - `modifier = Modifier.fillMaxWidth()`: Ensures the button spans the entire width of its parent container.
  - `modifier.padding(vertical = 4.dp)`: Adds vertical padding (4.dp) above and below the button for spacing.
- **Text Component:**
  - Displays the `text` parameter as the button's label.

Component Hierarchy


Column
TextField ("Nombre del Tutor")
Spacer (8.dp)
TextField ("ID Tutor Actualizar/Modificar")
Spacer (8.dp)
TextField ("Nombre del Tutorado")
Spacer (16.dp)
ActionButton ("INSERTAR TUTOR")
ActionButton ("BORRAR TUTOR")
ActionButton ("CONSULTAR TUTORES")
ActionButton ("ACTUALIZAR TUTOR")
ActionButton ("AGREGAR TUTORADO")
ActionButton ("CONSULTAR TUTORADOS POR TUTOR")
Spacer (16.dp)
LazyColumn (Result List)
Text (for each result item in resultList)

Composable Functions in Android

Composable Function:

- A Composable function is a special type of function in Android used to build UI in a declarative way. It is a Kotlin function annotated with `@Composable`.
- The `@Composable` annotation informs the Compose compiler that the function is meant for UI construction.
- Unlike traditional XML layouts in Android, Composable functions allow us to create UI using Kotlin code, which is more intuitive and flexible.
- By annotating a function with `@Composable`, it becomes a composable function that can be called within other composable functions to build a hierarchy of UI elements.

Compose Modifiers:

- Compose Modifiers allow you to decorate or configure composable functions to control their size, layout, behavior, and appearance.
- Examples of using modifiers include:
    - Setting a composable's size or padding.
    - Adding interactivity, such as `clickable`, `scrollable`, or `draggable`.
    - Providing accessibility information, like setting content descriptions.
- Modifiers are applied in a chain, where each modifier can build upon the previous one, allowing for powerful and flexible customization of composable functions.

TutorDatabaseHelper.kt

```kotlin
import android.content.ContentValues
import android.content.Context
import android.database.sqlite.SQLiteDatabase
import android.database.sqlite.SQLiteOpenHelper

class TutorDatabaseHelper(context: Context) : SQLiteOpenHelper(context, DATABASE_NAME, null, DATABASE_VERSION) {

    override fun onCreate(db: SQLiteDatabase) {

        db.execSQL("CREATE TABLE $TABLE_TUTOR (ID INTEGER PRIMARY KEY AUTOINCREMENT, NAME TEXT NOT NULL)")

        db.execSQL("CREATE TABLE $TABLE_TUTORADO (ID INTEGER PRIMARY KEY AUTOINCREMENT, " +
                "NAME TEXT NOT NULL, TUTOR_ID INTEGER, FOREIGN KEY (TUTOR_ID) REFERENCES $TABLE_TUTOR(ID))")
    }

    override fun onUpgrade(db: SQLiteDatabase, oldVersion: Int, newVersion: Int) {
        db.execSQL("DROP TABLE IF EXISTS $TABLE_TUTOR")
        db.execSQL("DROP TABLE IF EXISTS $TABLE_TUTORADO")
        onCreate(db)
    }
```

TutorDatabaseHelper.kt

```kotlin
fun insertTutor(name: String): Long {
    val db = this.writableDatabase
    val values = ContentValues().apply {
        put("NAME", name)
    }
    return db.insert(TABLE_TUTOR, null, values)
}

fun deleteTutor(id: Long) {
    val db = this.writableDatabase
    db.delete(TABLE_TUTOR, "ID=?", arrayOf(id.toString()))
}

fun updateTutor(id: Long, name: String) {
    val db = this.writableDatabase
    val values = ContentValues().apply {
        put("NAME", name)
    }
    db.update(TABLE_TUTOR, values, "ID=?", arrayOf(id.toString()))
}
```

- **insertTutor(name: String): Long** - Inserts a new tutor into the database with the specified name. It creates a `ContentValues` object containing the tutor's name and inserts it into the `TABLE_TUTOR`. Returns the ID of the newly inserted row as a `Long`.
- **deleteTutor(id: Long)** - Deletes a tutor from the database based on the provided ID. It removes the row in `TABLE_TUTOR` where the `ID` matches the specified `id` parameter.
- **updateTutor(id: Long, name: String)** - Updates the name of a tutor in the database based on the provided ID. It creates a `ContentValues` object with the new name and updates the row in `TABLE_TUTOR` where the `ID` matches the specified `id` parameter.

TutorDatabaseHelper.kt

```kotlin
fun getTutors(): List<Pair<Long, String>> {
    val db = this.readableDatabase
    val cursor = db.query(TABLE_TUTOR, arrayOf("ID", "NAME"),
    null, null, null, null, null)
    val tutors = mutableListOf<Pair<Long, String>>()
    while (cursor.moveToNext()) {
        val id = cursor.getLong(cursor.getColumnIndexOrThrow("ID"))
        val name = cursor.getString(cursor.getColumnIndexOrThrow("NAME"))
        tutors.add(Pair(id, name))
    }
    cursor.close()
    return tutors
}

fun addTutorado(name: String, tutorId: Long): Long {
    val db = this.writableDatabase
    val values = ContentValues().apply {
        put("NAME", name)
        put("TUTOR_ID", tutorId)
    }
    return db.insert(TABLE_TUTORADO, null, values)
}
```

- **getTutors(): List¡Pair¡Long, String¿¿** - Retrieves a list of all tutors in the database. This function queries `TABLE_TUTOR` for the `ID` and `NAME` columns and iterates through each row, adding each tutor as a `Pair` of `ID` and `Name` to a list. The list of tutors is then returned.
- **addTutorado(name: String, tutorId: Long): Long** - Adds a new student (tutorado) associated with a specific tutor to the database. It creates a `ContentValues` object containing the student's `name` and the `tutorId` of the associated tutor, then inserts this data into `TABLE_TUTORADO`. Returns the ID of the newly inserted row as a `Long`.

TutorDatabaseHelper.kt

```
    fun getTutoradosByTutor(tutorId: Long): List<String> {
        val db = this.readableDatabase
        val cursor = db.query(TABLE_TUTORADO, arrayOf("ID", "NAME")
        , "TUTOR_ID=?", arrayOf(tutorId.toString()),
        null, null, null)
        val tutorados = mutableListOf<String>()
        while (cursor.moveToNext()) {
            val name = cursor.getString(cursor.getColumnIndexOrThrow("NAME"))
            tutorados.add(name)
        }
        cursor.close()
        return tutorados
    }

    companion object {
        private const val DATABASE_VERSION = 1
        private const val DATABASE_NAME = "tutorDB"
        const val TABLE_TUTOR = "Tutor"
        const val TABLE_TUTORADO = "Tutorado"
    }
}
```

- **getTutoradosByTutor(tutorId: Long): List¡String¿** - Retrieves a list of students (tutorados) associated with a specific tutor based on the provided `tutorId`. This function queries `TABLE_TUTORADO` for entries where `TUTOR_ID` matches the specified `tutorId` and retrieves the `NAME` of each student. It adds each name to a list, which is returned as the result.
- **companion object** - Defines constants and

  database configuration for `TutorDatabaseHelper`:
  - `DATABASE_VERSION`: The version of the database, set to 1.
  - `DATABASE_NAME`: The name of the database, defined as `"tutorDB"`.
  - `TABLE_TUTOR`: The name of the table storing tutor information, set as `"Tutor"`.
  - `TABLE_TUTORADO`: The name of the table storing student information, defined as `"Tutorado"`.

TutorAppTheme.kt

```kotlin
package com.z_iti_271311_u1_ae_lopez_leal_antonio_isai.ui.theme

import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.darkColorScheme
import androidx.compose.material3.lightColorScheme
import androidx.compose.runtime.Composable

private val LightColors = lightColorScheme(
    primary = androidx.compose.ui.graphics.Color(0xFF6200EE),
    onPrimary = androidx.compose.ui.graphics.Color.White,
    secondary = androidx.compose.ui.graphics.Color(0xFF03DAC6),
    onSecondary = androidx.compose.ui.graphics.Color.Black
)

private val DarkColors = darkColorScheme(
    primary = androidx.compose.ui.graphics.Color(0xFFBB86FC),
    onPrimary = androidx.compose.ui.graphics.Color.Black,
    secondary = androidx.compose.ui.graphics.Color(0xFF03DAC6),
    onSecondary = androidx.compose.ui.graphics.Color.Black
)
```

- Create a file named `TutorAppTheme.kt` in the `ui.theme` package.
- Copy and paste the code provided into `TutorAppTheme.kt`.
- This code defines two color schemes:
  - `LightColors`: A color scheme for light mode, with primary and secondary colors.
  - `DarkColors`: A color scheme for dark mode, with adjusted primary and secondary colors.
- These color schemes can be used within `MaterialTheme` to toggle between light and dark modes.

TutorAppTheme.kt

```kotlin
@Composable
fun TutorAppTheme(
    darkTheme: Boolean = false, // Puedes habilitar el tema oscuro si quieres
    content: @Composable () -> Unit
) {
    val colors = if (darkTheme) DarkColors else LightColors

    MaterialTheme(
        colorScheme = colors,
        typography = Typography,
        shapes = Shapes,
        content = content
    )
}
```

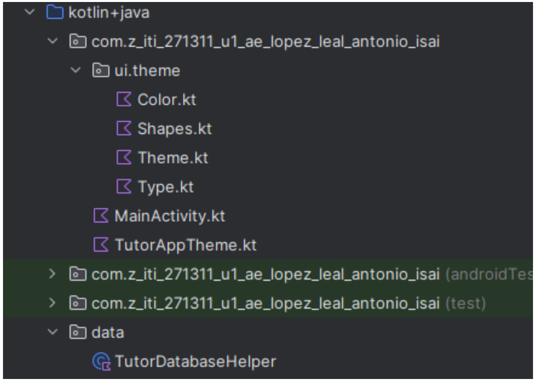
- `darkTheme: Boolean = false` - A Boolean parameter that controls whether the dark theme is enabled. By default, it is set to `false`, meaning the light theme is applied unless specified otherwise.
- `colors` - A variable that selects the appropriate color scheme based on the `darkTheme` setting. If `darkTheme` is `true`, `DarkColors` is used; otherwise, `LightColors` is applied.
- `MaterialTheme` - The core theme composable in Jetpack Compose. It applies the selected `colorScheme`, `typography`, and `shapes` to the app's UI components within the provided `content`.

Shapes.kt / Type.kt

```
// Shapes.kt
package com.z_iti_271311_u1_ae_lopez_leal_antonio_isai.ui.theme

import androidx.compose.material3.Shapes

val Shapes = Shapes()

// Type.kt
package com.z_iti_271311_u1_ae_lopez_leal_antonio_isai.ui.theme

import androidx.compose.material3.Typography
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.text.font.FontFamily
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.sp

val Typography = Typography(
    bodyLarge = TextStyle(
        fontFamily = FontFamily.Default,
        fontWeight = FontWeight.Normal,
        fontSize = 16.sp,
        lineHeight = 24.sp,
        letterSpacing = 0.5.sp
    )
)
```

- **Shapes.kt** - Defines custom shapes for the app's UI components using Jetpack Compose's `Shapes` class.
  - `Shapes`: A single instance of the `Shapes` class that can be customized for different corner shapes. Here, it's initialized with default settings, which can later be customized within `MaterialTheme`.
- **Type.kt** - Defines typography styles for the app's UI using the `Typography` class from Jetpack Compose.
  - `Typography`: An instance of the `Typography` class customized to define the default font styles.
  - `bodyLarge`: A `TextStyle` object defining the font style for large body text.

Project Directory Structure



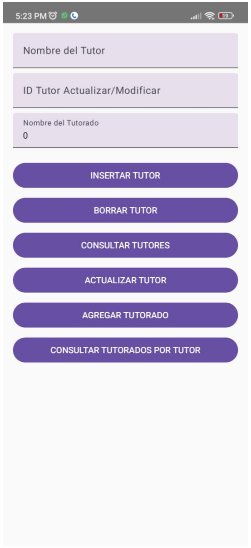
```
kotlin+java
  com.z_iti_271311_u1_ae_lopez_leal_antonio_isai
    ui.theme
      Color.kt
      Shapes.kt
      Theme.kt
      Type.kt
    MainActivity.kt
    TutorAppTheme.kt
  com.z_iti_271311_u1_ae_lopez_leal_antonio_isai (androidTes
  com.z_iti_271311_u1_ae_lopez_leal_antonio_isai (test)
  data
    TutorDatabaseHelper
```

- `data` - Contains `TutorDatabaseHelper`, a helper class for managing the SQLite database operations.
- `ui.theme` - Contains files for customizing the UI theme with Jetpack Compose:
  - `Color.kt` - Defines the color schemes for light and dark themes.
  - `Shapes.kt` - Sets default shape styling for UI components.
  - `Type.kt` - Defines the typography styles for text elements.
  - `Theme.kt` - The main theme setup file that combines colors, shapes, and typography.
- `MainActivity.kt` - The main entry point for the application, containing the composable UI layout.
- `TutorAppTheme.kt` - Configures the app theme, allowing switching between light and dark mode.

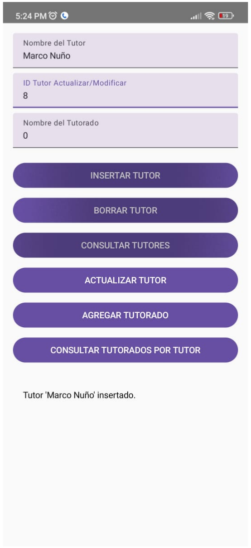
Explanation of the Conversion

- The `ConstraintLayout` in XML is converted to a `Box` in Jetpack Compose.
- The `fillMaxSize()` function makes the `Box` occupy the entire available space, similar to `match_parent`.
- The `contentAlignment = Alignment.Center` centers the text inside the `Box`, mimicking the `ConstraintLayout` constraints.
- The `TextView` is converted to `BasicText` with the text "Hello World!".

Result

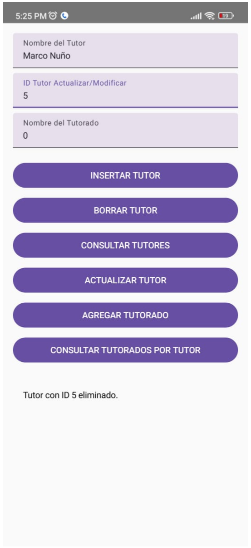

5:23 PM

Nombre del Tutor
ID Tutor Actualizar/Modificar
Nombre del Tutorado: 0


5:24 PM

Nombre del Tutor: Marco Nuño
ID Tutor Actualizar/Modificar: 8
Nombre del Tutorado: 0

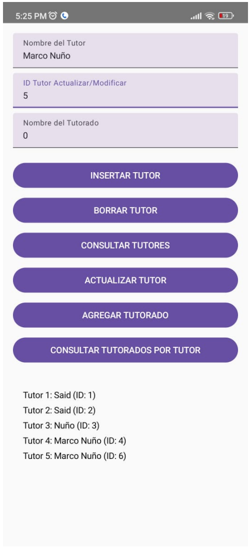
```
Tutor 'Marco Nuño' insertado.
```


5:25 PM

Nombre del Tutor: Marco Nuño
ID Tutor Actualizar/Modificar: 5
Nombre del Tutorado: 0

```
Tutor con ID 5 eliminado.
```

Result

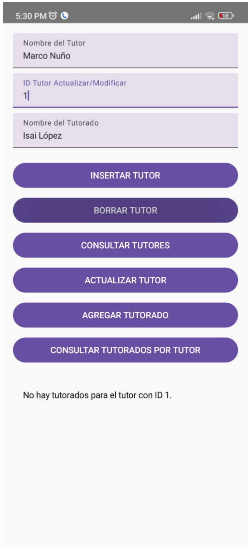

5:25 PM

Nombre del Tutor: Marco Nuño

ID Tutor Actualizar/Modificar: 5

Nombre del Tutorado: 0

```
Tutor 1: Said (ID: 1)
Tutor 2: Said (ID: 2)
Tutor 3: Nuño (ID: 3)
Tutor 4: Marco Nuño (ID: 4)
Tutor 5: Marco Nuño (ID: 6)
```

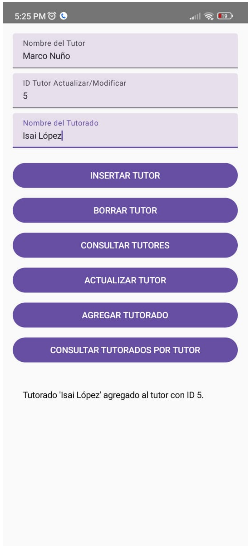

5:30 PM

Nombre del Tutor: Marco Nuño

ID Tutor Actualizar/Modificar: 1

Nombre del Tutorado: Isai López

```
No hay tutorados para el tutor con ID 1.
```


5:25 PM

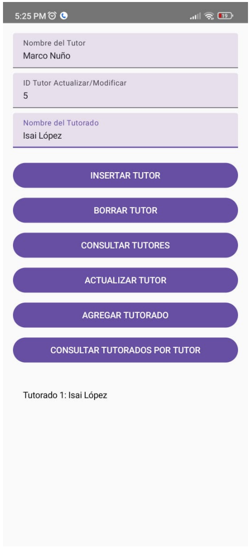
Nombre del Tutor: Marco Nuño

ID Tutor Actualizar/Modificar: 5

Nombre del Tutorado: Isai López

```
Tutorado 'Isai López' agregado al tutor con ID 5.
```

Result



Nombre del Tutor
Marco Nuño

ID Tutor Actualizar/Modificar
5

Nombre del Tutorado
Isai López

Tutorado 1: Isai López