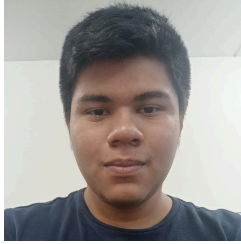


<b>PICTURE HERE</b> 	<b>Nombre: Juan Sebastián Pinto Polania</b> <b>Código: 20231212983</b>
<b>WEEK 12?</b> <b>GRADE:</b>	

Buenas tardes profesora, por este medio le comparto el desarrollo de las actividades solicitadas por usted en el parcial de hoy.

## Actividad 1

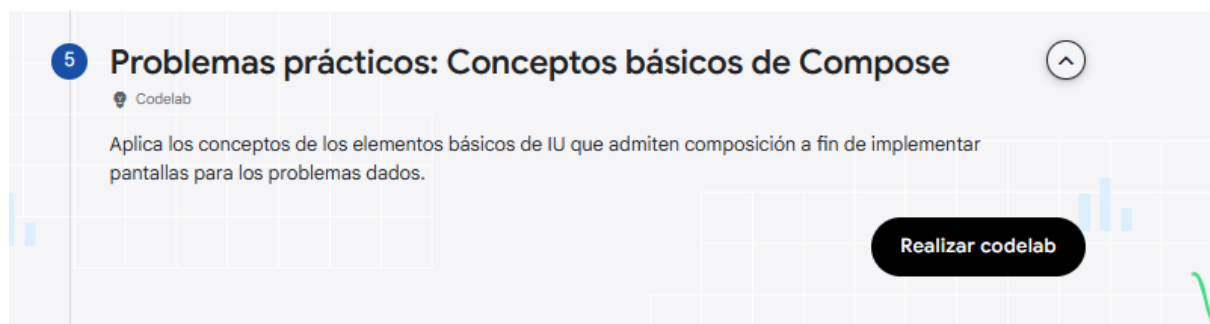
En este ejercicio, compilaré una pantalla para la app que mostrará un instructivo para Jetpack Compose. Para resolver este problema, utilizaré los recursos de imágenes y strings que se encuentran en la sección Recursos.

### 1.1 Antes de trabajar

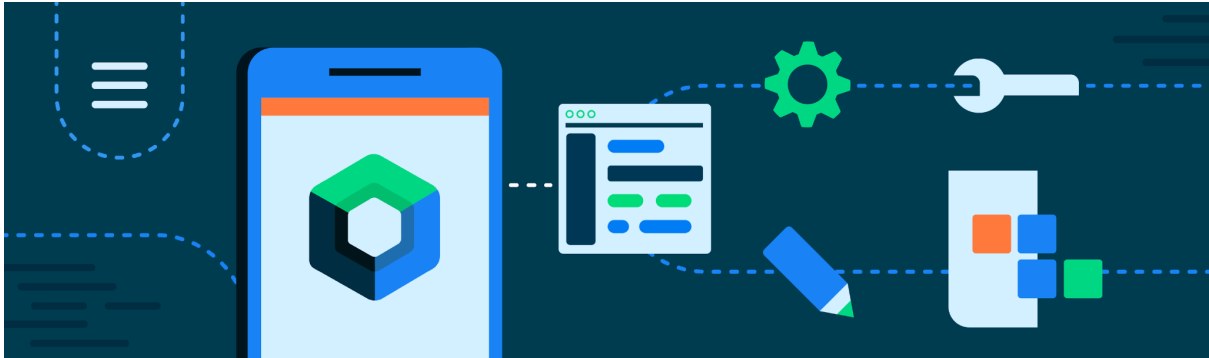
Antes que nada es recomendable en primer lugar abrir el link proporcionado por la profesora en el documento del parcial:

<https://developer.android.com/courses/pathways/android-basics-compose-unit-1-pathway-3?hl=es-419>

Que nos redirigirá a una página de actividades de Jetpack Compose, después de ello iremos al apartado "5) Problemas prácticos: Conceptos básicos de Compose" e ingresamos a la actividad del codelab:



Dentro del Codelab vamos a la segunda actividad donde dice “Artículo de Compose” y seguido a ello seguimos las instrucciones que nos dan, además de descargar los recursos que necesitamos, que en este caso sería la [imagen](#) que nos proporcionan más abajo en el documento:

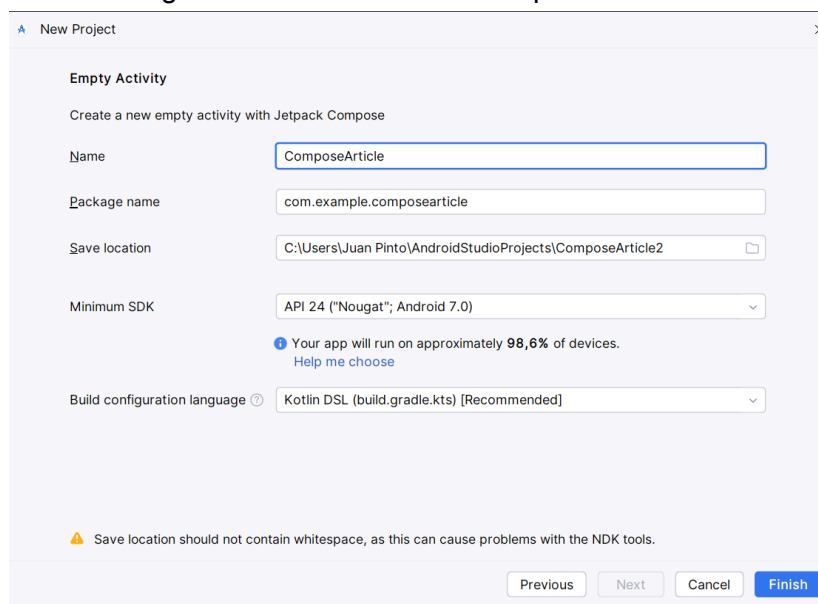


Además de los recursos para los strings:

- [Jetpack Compose tutorial](#)
- [Jetpack Compose is a modern toolkit for building native Android UI. Compose simplifies and accelerates UI development on Android with less code, powerful tools, and intuitive Kotlin APIs.](#)
- [In this tutorial, you build a simple UI component with declarative functions. You call Compose functions to say what elements you want and the Compose compiler does the rest. Compose is built around Composable functions. These functions let you define your app's UI programmatically because they let you describe how it should look and provide data dependencies, rather than focus on the process of the UI's construction, such as initializing an element and then attaching it to a parent. To create a Composable function, you add the @Composable annotation to the function name.](#)

## 1.2) Front

Una vez tengamos en cuenta lo anterior procedemos a crear el proyecto en Android Studio:



y después procedemos a crear el proyecto teniendo en cuenta los requerimientos del codelab, que quedaría más o menos de esta forma:

```

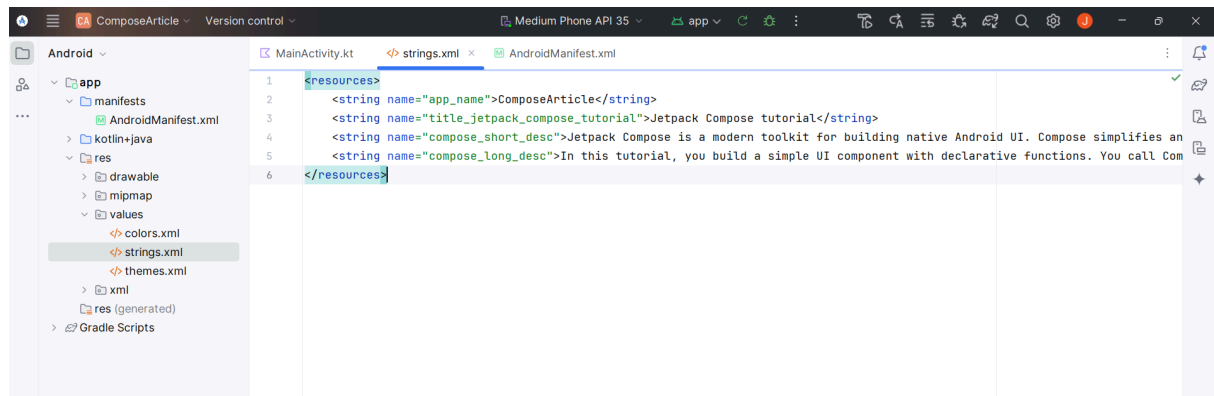
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            TaskCompletedTheme {
                // A surface container using the 'background' color
                from the theme
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colorScheme.background
                ) {
                    TaskCompletedScreen()
                }
            }
        }
    }
}

@Composable
fun TaskCompletedScreen() {
    Column(
        modifier = Modifier
            .fillMaxWidth()
            .fillMaxHeight(),
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        val image = painterResource(R.drawable.ic_task_completed)
        Image(painter = image, contentDescription = null)
        Text(
            text = stringResource(R.string.all_task_completed),
            modifier = Modifier.padding(top = 24.dp, bottom = 8.dp),
            fontWeight = FontWeight.Bold
        )
        Text(
            text = stringResource(R.string.nice_work),
            fontSize = 16.sp
        )
    }
}

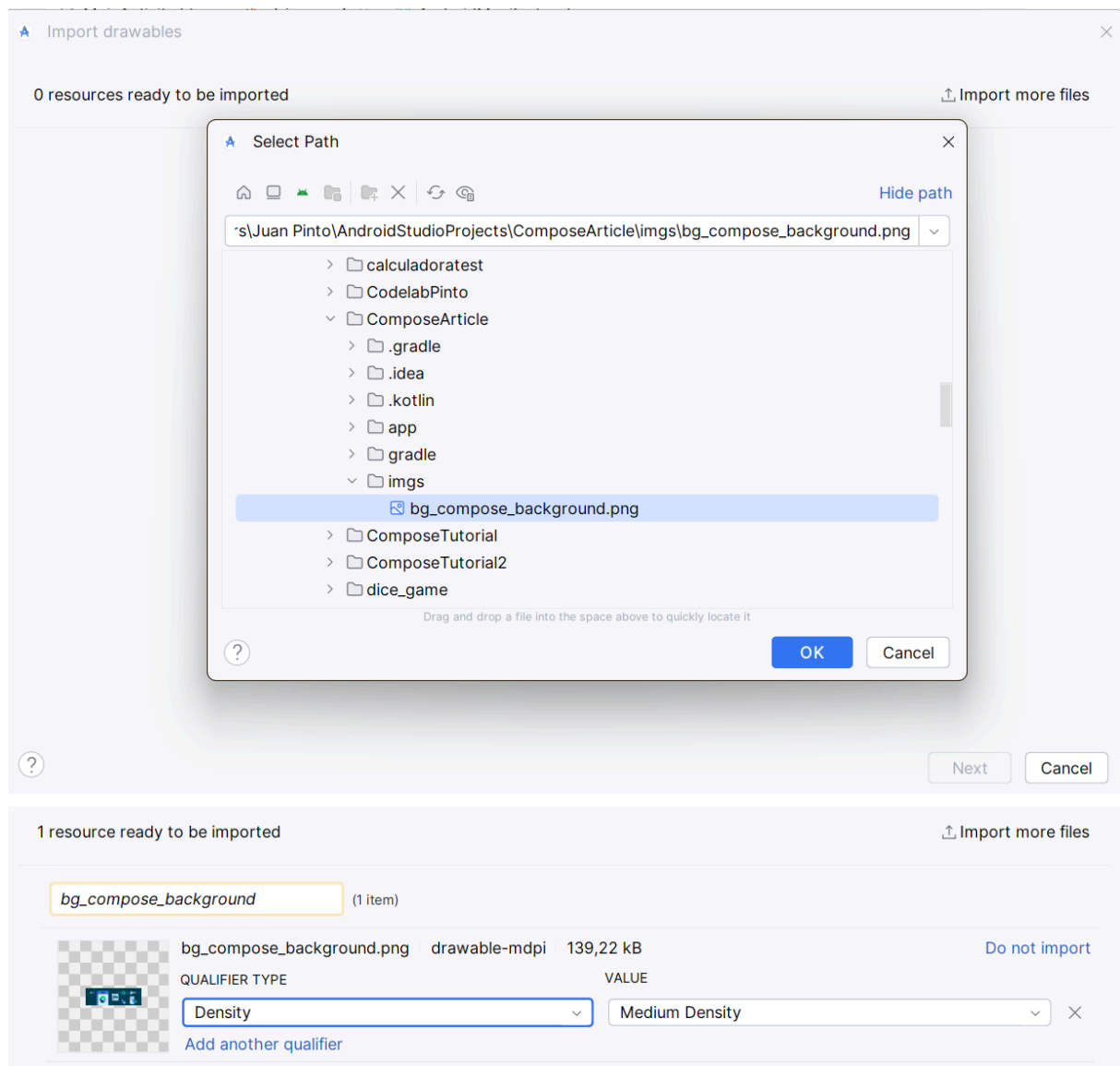
@Preview(showBackground = true)
@Composable
fun TaskCompletedPreview() {
    TaskCompletedTheme {
        TaskCompletedScreen()
    }
}

```

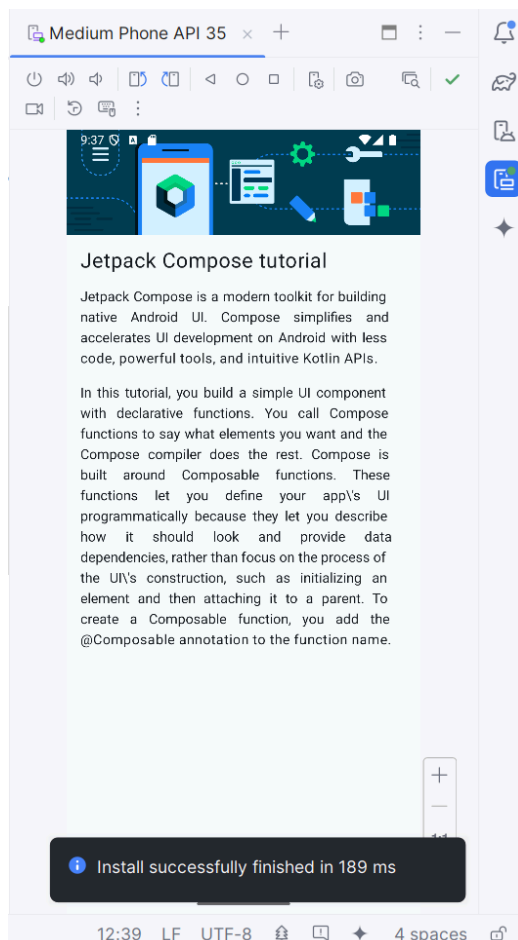
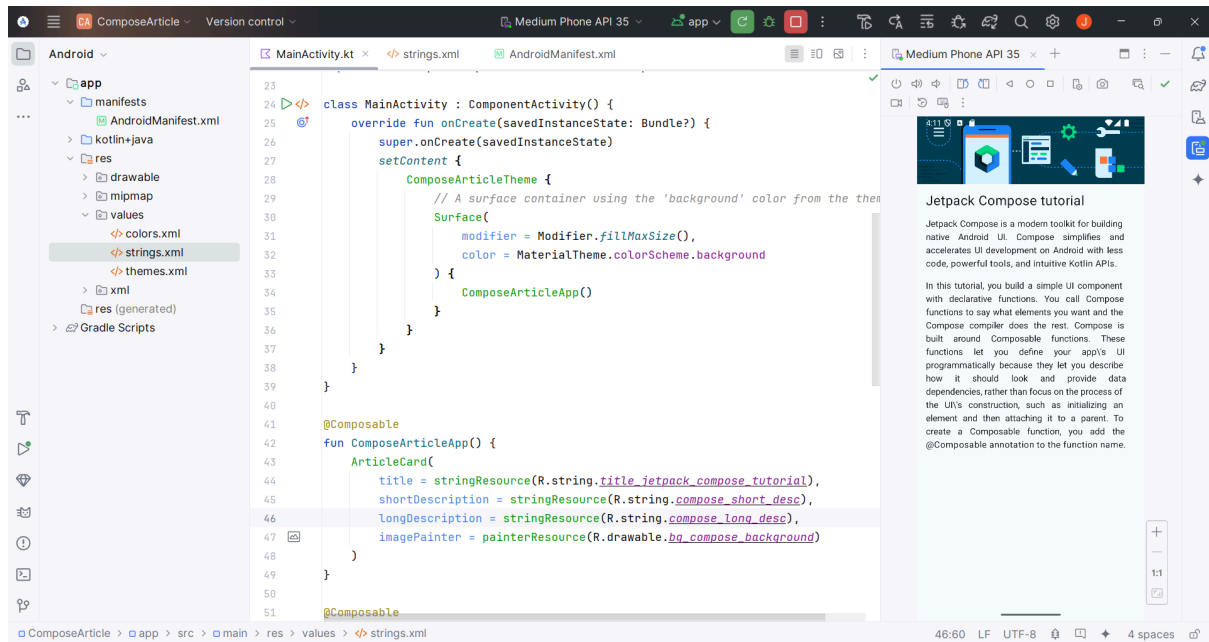
Después procedemos a ajustar los strings para que contengan los siguientes datos:



y por último importamos el drawable de la imagen que descargamos desde el GitHub:

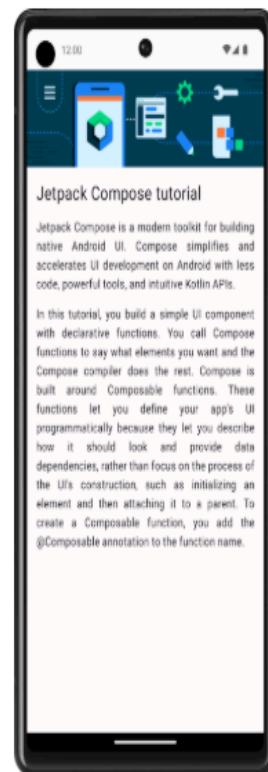


Una vez ya hayamos hecho esto nuestra actividad está lista para ser ejecutada y testada, el resultado que obtenemos es satisfactorio y de acuerdo con lo solicitado en el codelab.



### Captura de pantalla final

Cuando termines la implementación, tu diseño debería coincidir con esta captura de pantalla:



## Actividad 2

Me ahorraré el apartado de “antes de trabajar” ya que es prácticamente el mismo, con la leve diferencia que al entrar al codelab debemos posicionarnos en la actividad “3. Administrador de tareas” y descargar la [imagen](#) correspondiente para esta actividad.

Una vez aclarado esto, procedemos a crear el front.

### 2.1) Front

El proceso de este no es muy diferente al primero, primero procedemos a crear la estructura básica del front, que nos quedaría algo así:

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            TaskCompletedTheme {
                // A surface container using the 'background' color
                from the theme
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colorScheme.background
                ) {
                    TaskCompletedScreen()
                }
            }
        }
    }
}

@Composable
fun TaskCompletedScreen() {
    Column(
        modifier = Modifier
            .fillMaxWidth()
            .fillMaxHeight(),
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        val image = painterResource(R.drawable.ic_task_completed)
        Image(painter = image, contentDescription = null)
        Text(
            text = stringResource(R.string.all_task_completed),
            modifier = Modifier.padding(top = 24.dp, bottom = 8.dp),
            fontWeight = FontWeight.Bold
        )
        Text(
            text = stringResource(R.string.nice_work),

```

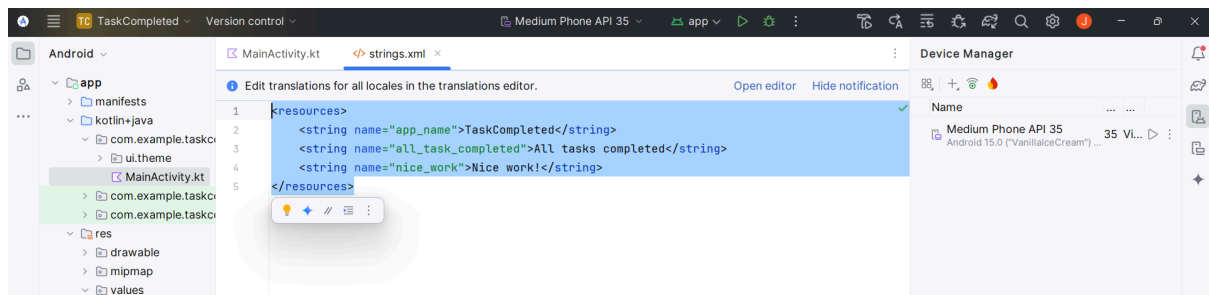
```

        fontSize = 16.sp
    )
}

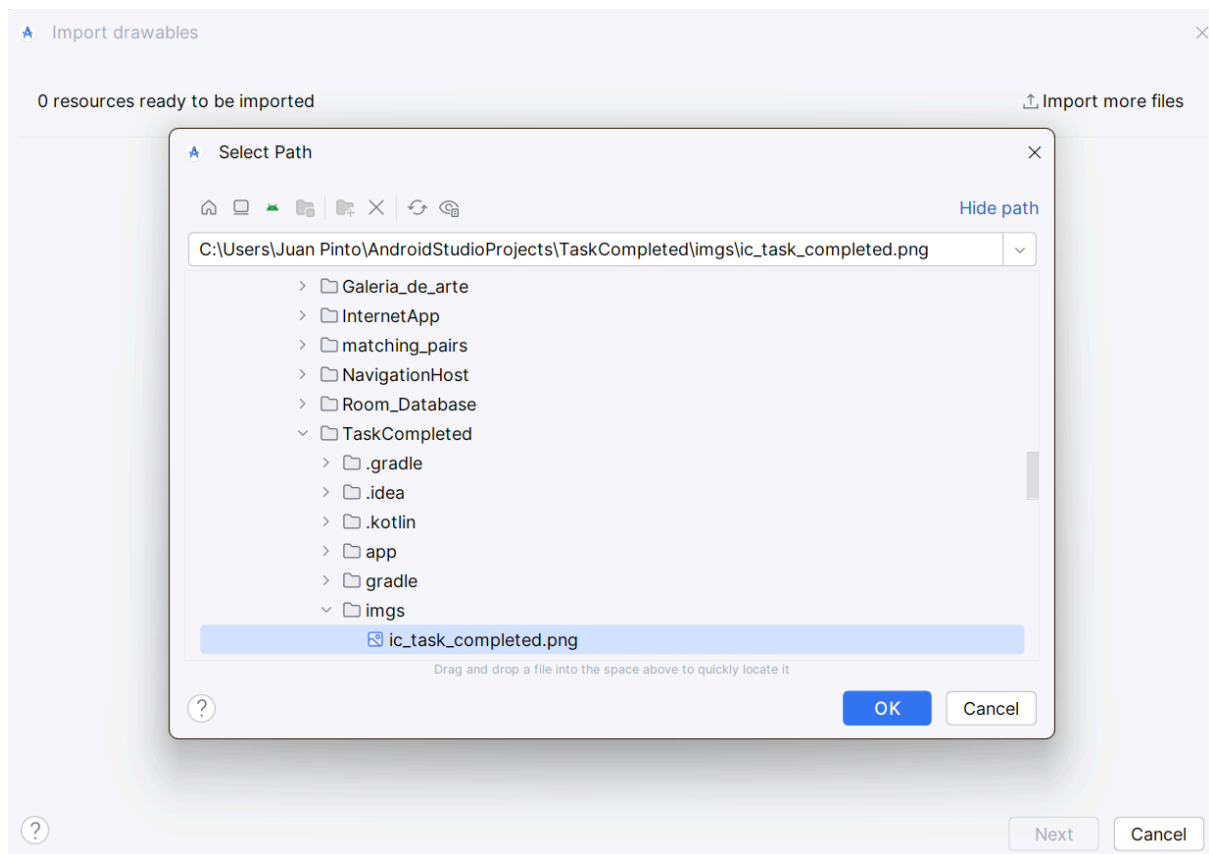
@Preview(showBackground = true)
@Composable
fun TaskCompletedPreview() {
    TaskCompletedTheme {
        TaskCompletedScreen()
    }
}

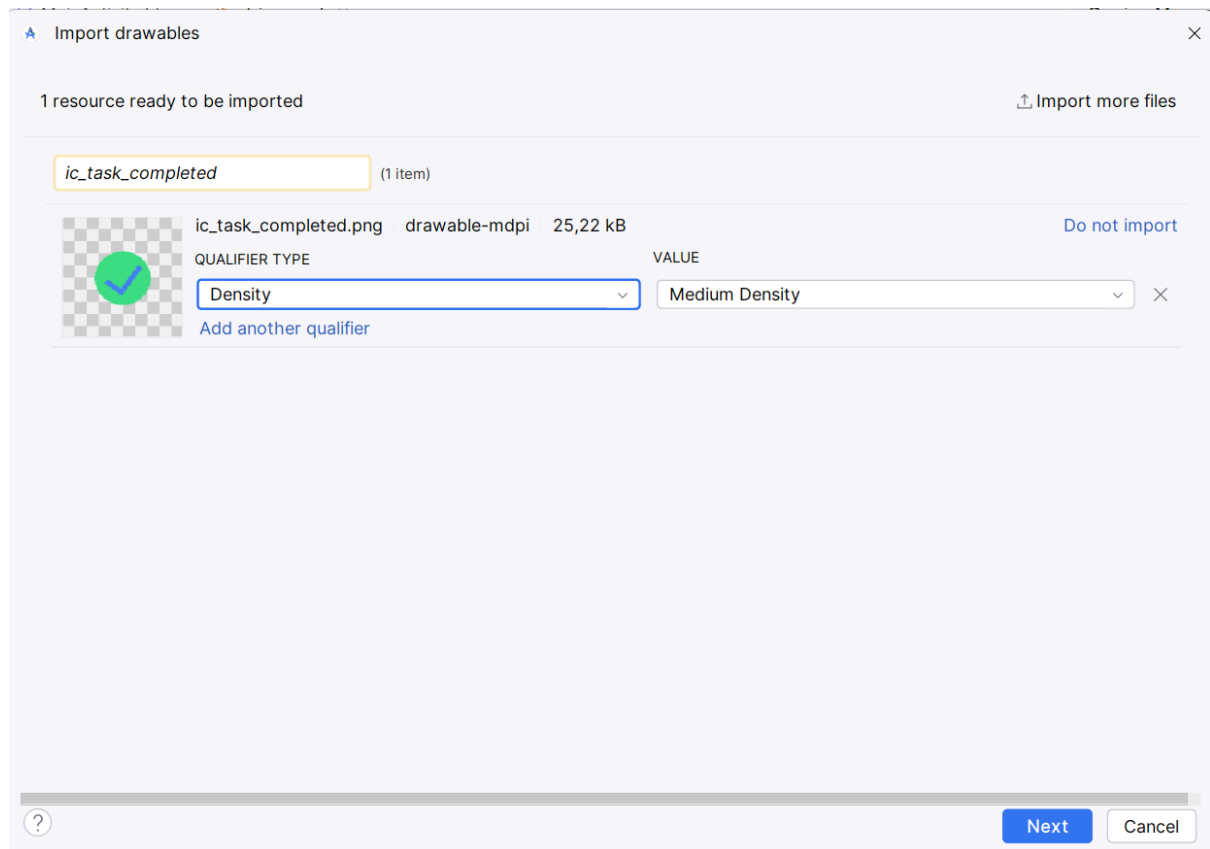
```

Después procedemos a declarar el texto de los strings:

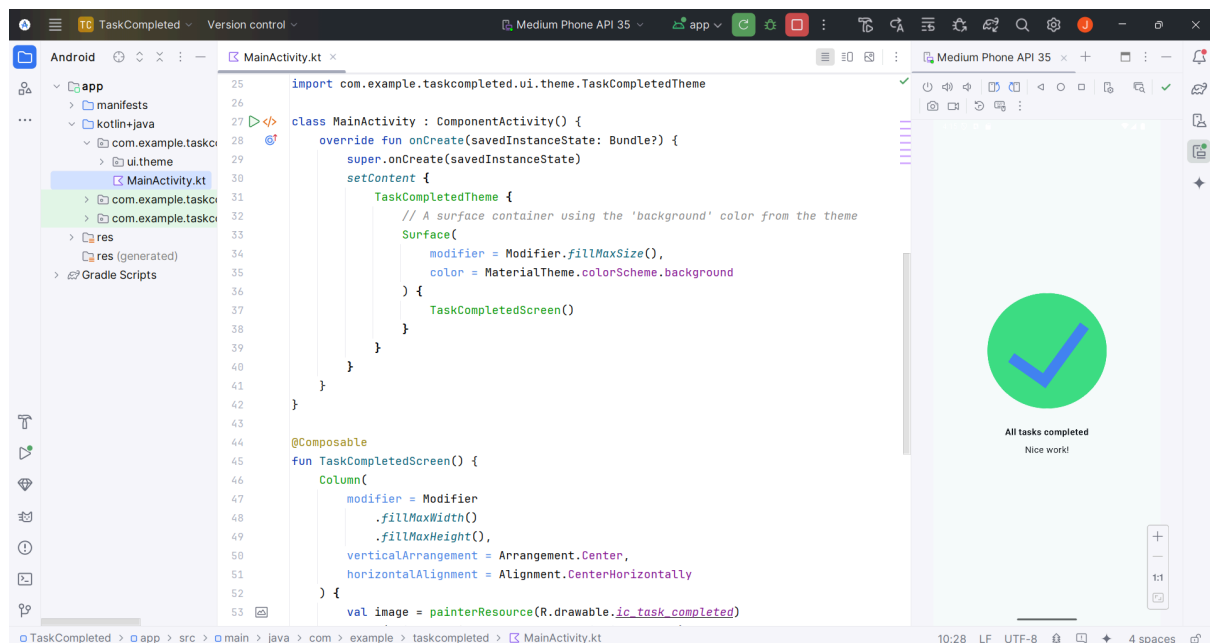


Y por último procedemos a importar el Drawable desde el resource manager:

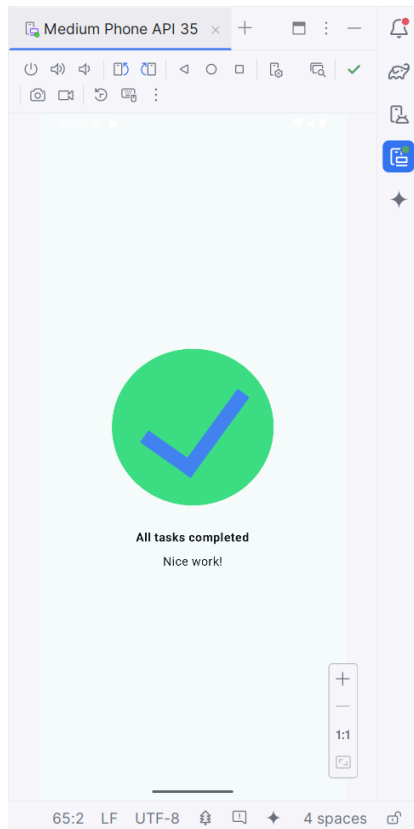




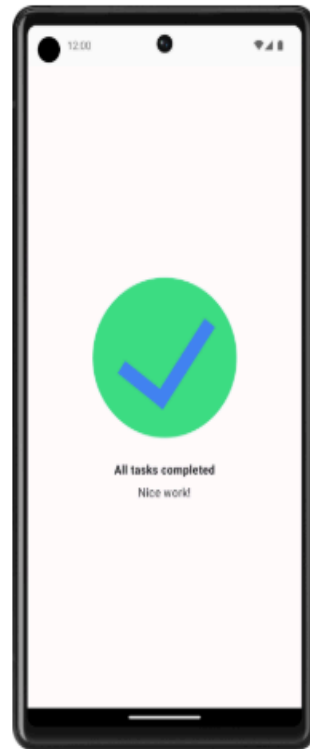
Y ya por último nos quedaría testear que la app ejecute correctamente y que tenga su respectivo parecido con lo solicitado en el codelab:







Cuando termines la implementación, tu diseño debería coincidir con esta captura de pantalla:



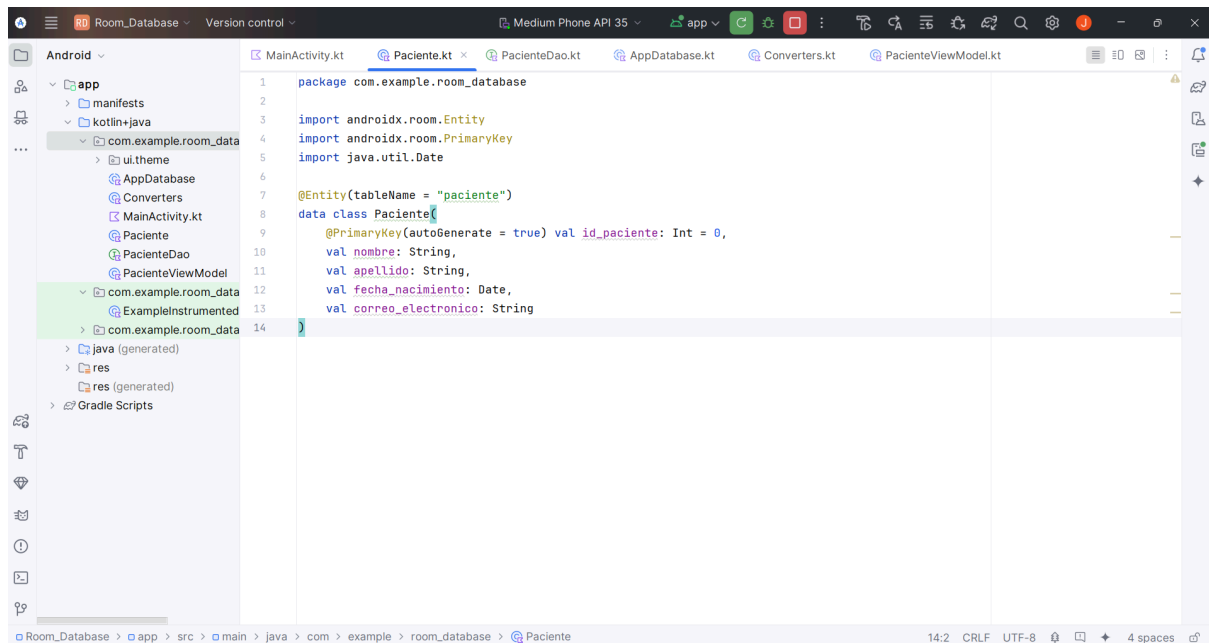
### Actividad 3

Para la actividad tres procederé a usar el Taller 6. SQL DATABASES como plantilla, por ende los pasos como crear el repositorio, dependencias y demás cosas nos la saltamos y procedemos a editar los archivos que tenemos para poder realizar la actividad.

En primer lugar procedemos a eliminar los archivos de la anterior actividad ya que estos pueden generarnos problemas al momento de testear, así que los archivos que tengan que ver directamente con “User” como puede ser UserDao, User.kt, etc, los eliminamos de forma segura, después procedemos a crear y/o editar los archivos:

## Crear la Entidad (Paciente)

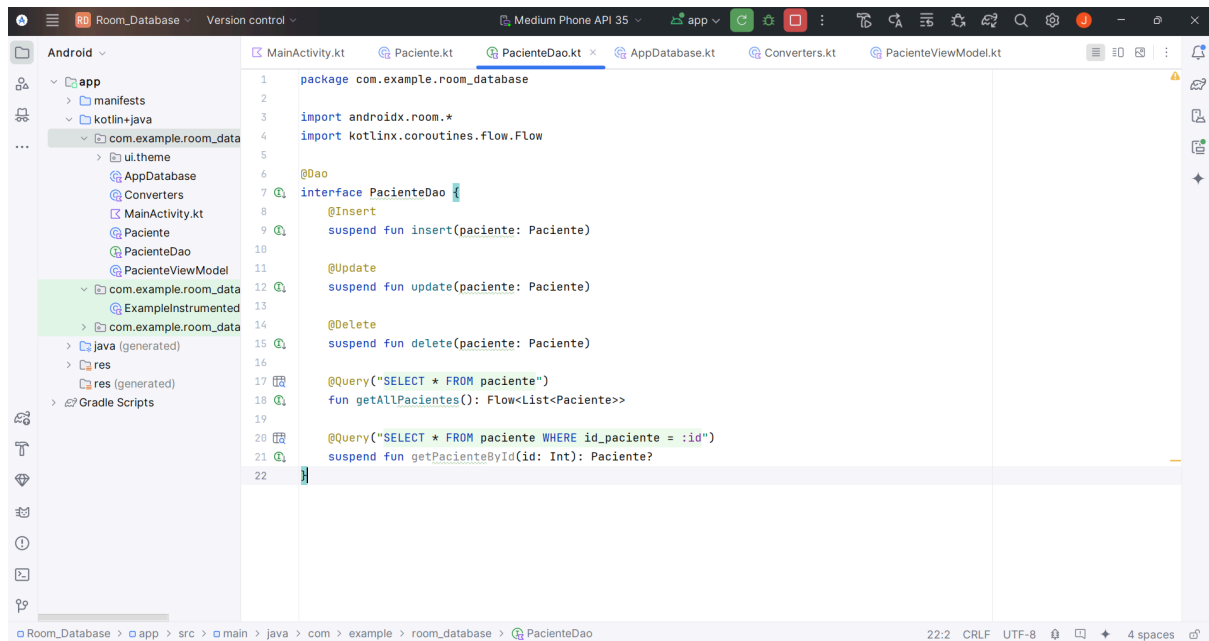
Procedemos a crear el archivo Paciente.kt (tipo Data Class) en com.example.room\_database.



```
1 package com.example.room_database
2
3 import androidx.room.Entity
4 import androidx.room.PrimaryKey
5 import java.util.Date
6
7 @Entity(tableName = "paciente")
8 data class Paciente(
9     @PrimaryKey(autoGenerate = true) val id_paciente: Int = 0,
10     val nombre: String,
11     val apellido: String,
12     val fecha_nacimiento: Date,
13     val correo_electronico: String
14 )
```

## Definir el DAO (PacienteDao)

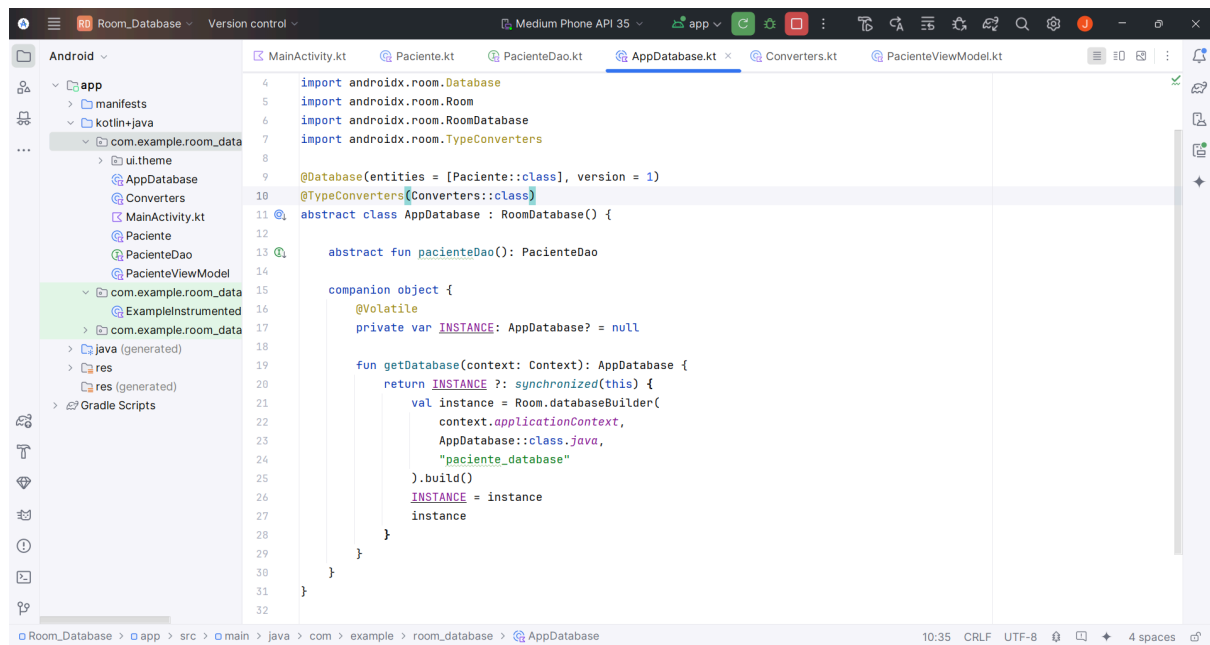
El DAO proporciona métodos para manipular la base de datos, procedemos a crear el archivo PacienteDao.kt (tipo Interface) en com.example.room\_database.



```
1 package com.example.room_database
2
3 import androidx.room.*
4 import kotlinx.coroutines.flow.Flow
5
6 @Dao
7 interface PacienteDao {
8     @Insert
9     suspend fun insert(paciente: Paciente)
10
11     @Update
12     suspend fun update(paciente: Paciente)
13
14     @Delete
15     suspend fun delete(paciente: Paciente)
16
17     @Query("SELECT * FROM paciente")
18     fun getAllPacientes(): Flow<List<Paciente>>
19
20     @Query("SELECT * FROM paciente WHERE id_paciente = :id")
21     suspend fun getPacienteById(id: Int): Paciente?
22 }
```

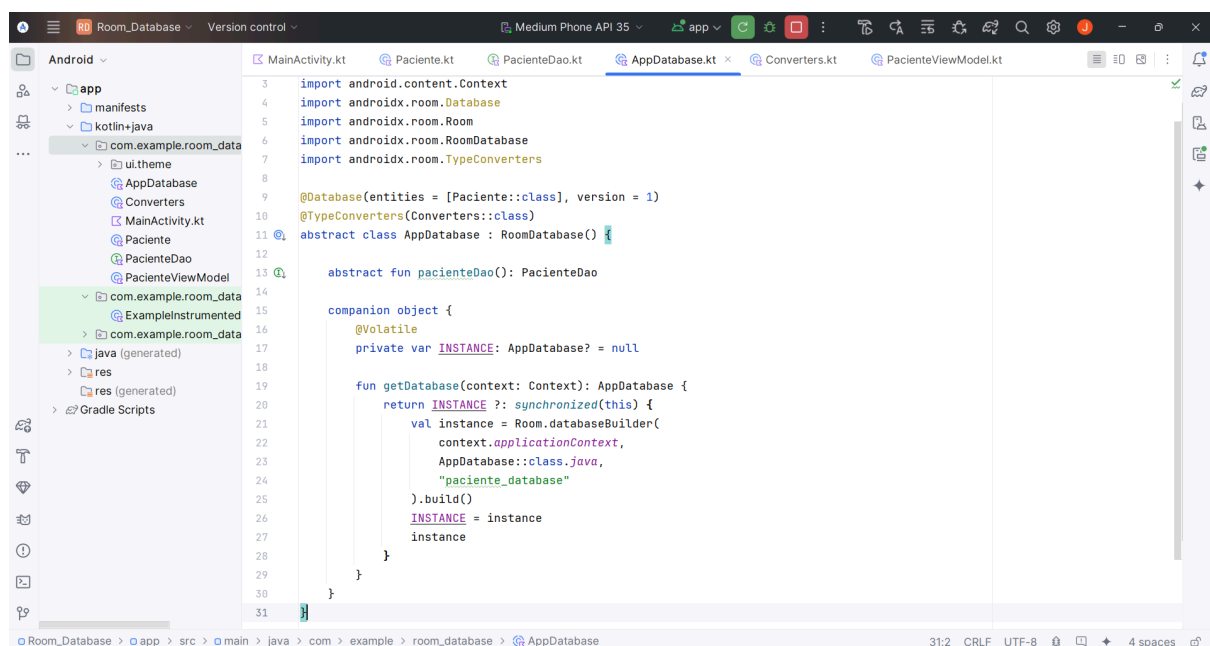
## Crear la base de datos

Procedemos a crear la base de datos siguiendo las instrucciones y el código proporcionado en el documento. Procedemos a crear el archivo AppDatabase.kt en com.example.room\_database.



## Crear el ViewModel.

Crear el ViewModel (PacienteViewModel). El ViewModel maneja la lógica de los datos:  
Filename: PacienteViewModel.kt.



```

package com.example.room_database

import android.content.Context
import androidx.room.Database
import androidx.room.Room
import androidx.room.RoomDatabase
import androidx.room.TypeConverters

@Database(entities = [Paciente::class], version = 1)
@TypeConverters(Converters::class)
abstract class AppDatabase : RoomDatabase() {

    abstract fun pacienteDao(): PacienteDao

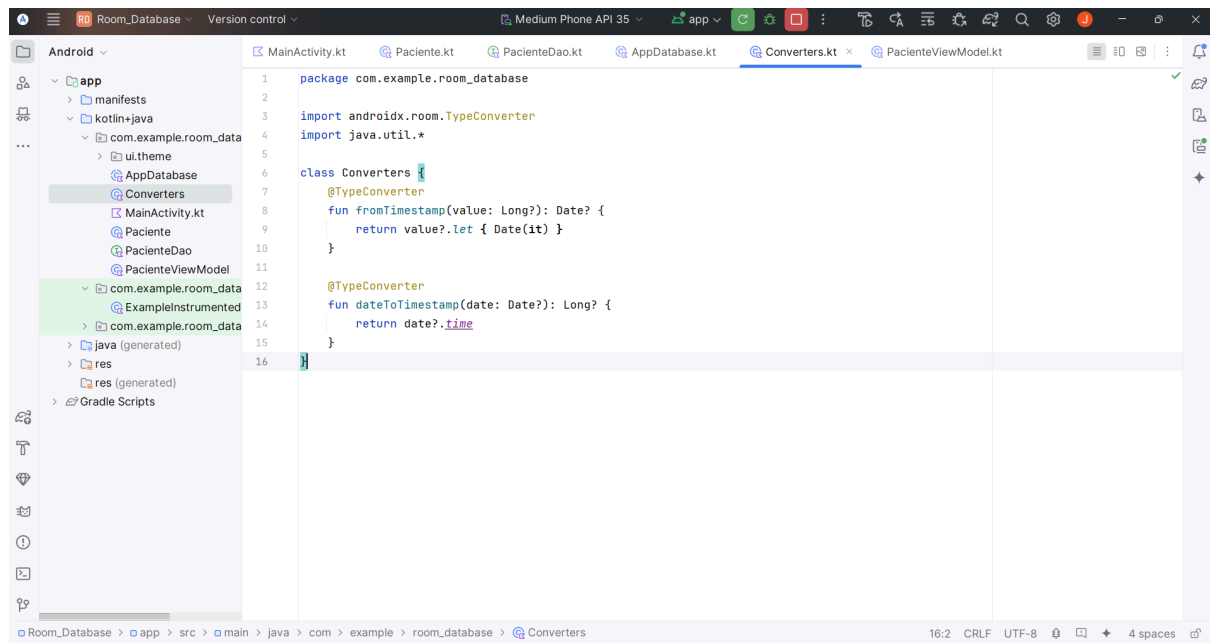
    companion object {
        @Volatile
        private var INSTANCE: AppDatabase? = null

        fun getDatabase(context: Context): AppDatabase {
            return INSTANCE ?: synchronized(this) {
                val instance = Room.databaseBuilder(
                    context.applicationContext,
                    AppDatabase::class.java,
                    "paciente_database"
                ).build()
                INSTANCE = instance
                instance
            }
        }
    }
}

```

## Crear Converters.kt.

En mi proyecto utilicé el archivo `Converters.kt` para que Room pudiera manejar correctamente las fechas. Como Room no puede guardar directamente objetos de tipo `Date`, creé funciones que convirtieran esos objetos a milisegundos (`Long`) antes de guardarlos, y que luego los volvieran a convertir a `Date` al recuperarlos. Estas funciones las marqué con la anotación `@TypeConverter` y las registré en la base de datos usando `@TypeConverters`. Gracias a esto, logré que los datos de tipo fecha se almacenaran sin errores y se mostraran correctamente en la aplicación.



## Crear la interfaz

Crear la UI con Jetpack Compose. Aquí diseñamos la interfaz de usuario en Jetpack Compose. Se Implementa la UI en MainActivity.kt. En esta actividad, agregamos botones para insertar, actualizar, eliminar y mostrar usuarios.

```
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.activity.enableEdgeToEdge
import androidx.activity.viewModels
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.foundation.text.KeyboardOptions
import androidx.compose.material3.*
import androidx.compose.runtime.*
import androidx.compose.ui.Modifier
import androidx.compose.ui.text.input.KeyboardType
import androidx.compose.ui.unit.dp
import androidx.compose.runtime.livedata.observeAsState
import com.example.room_database.ui.theme.Room_DatabaseTheme
import java.text.SimpleDateFormat
import java.util.Locale

class MainActivity : ComponentActivity() {
    private val pacienteViewModel: PacienteViewModel by viewModels()

    override fun onCreate(savedInstanceState: Bundle?) {
```

```

        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            Room_DatabaseTheme {
                PacienteApp(pacienteViewModel)
            }
        }
    }
}

@Composable
fun PacienteApp(pacienteViewModel: PacienteViewModel) {
    var id by remember { mutableStateOf("") }
    var nombre by remember { mutableStateOf("") }
    var apellido by remember { mutableStateOf("") }
    var fechaNacimiento by remember { mutableStateOf("") }
    var correo by remember { mutableStateOf("") }

    val pacientes by
pacienteViewModel.pacientes.observeAsState(emptyList())
    var showDialog by remember { mutableStateOf(false) }
    var pacienteToDelete by remember {
mutableStateOf<Paciente?>(null) }

    Scaffold { padding ->
        Column(modifier = Modifier.padding(padding).padding(16.dp))
    {

        TextField(
            value = id,
            onChange = { id = it },
            label = { Text("ID (para actualizar)") },
            modifier = Modifier.fillMaxWidth(),
            keyboardOptions =
KeyboardOptions.Default.copy(keyboardType = KeyboardType.Number)
        )

        TextField(
            value = nombre,
            onChange = { nombre = it },
            label = { Text("Nombre") },
            modifier = Modifier.fillMaxWidth()
        )

        TextField(
            value = apellido,
            onChange = { apellido = it },
            label = { Text("Apellido") },
            modifier = Modifier.fillMaxWidth()
        )
    }
}

```

```

    )

    TextField(
        value = fechaNacimiento,
        onChange = { fechaNacimiento = it },
        label = { Text("Fecha nacimiento (yyyy-MM-dd)") },
        modifier = Modifier.fillMaxWidth()
    )

    TextField(
        value = correo,
        onChange = { correo = it },
        label = { Text("Correo electrónico") },
        modifier = Modifier.fillMaxWidth()
    )

    Spacer(modifier = Modifier.height(8.dp))

    Row(
        modifier = Modifier.fillMaxWidth(),
        horizontalArrangement = Arrangement.SpaceEvenly
    ) {
        Button(onClick = {
            try {
                val fecha = SimpleDateFormat("yyyy-MM-dd",
                    Locale.getDefault()).parse(fechaNacimiento)
                if (nombre.isNotBlank() &&
                    apellido.isNotBlank() && correo.isNotBlank() && fecha != null) {
                    pacienteViewModel.insertPaciente(nombre, apellido, fecha, correo)
                    nombre = ""; apellido = "";
                    fechaNacimiento = ""; correo = ""
                }
            } catch (e: Exception) {

            }
        }) {
            Text("Insertar")
        }

        Button(onClick = {
            val idInt = id.toIntOrNull()
            try {
                val fecha = SimpleDateFormat("yyyy-MM-dd",
                    Locale.getDefault()).parse(fechaNacimiento)
                if (idInt != null && nombre.isNotBlank() &&
                    apellido.isNotBlank() && correo.isNotBlank() && fecha != null) {

```

```

        pacienteViewModel.updatePaciente(idInt,
nombre, apellido, fecha, correo)
        id = ""; nombre = ""; apellido = "";
fechaNacimiento = ""; correo = ""
    }
    } catch (e: Exception) {

    }
}) {
    Text("Actualizar")
}
}

Spacer(modifier = Modifier.height(16.dp))

LazyColumn {
    items(pacientes) { paciente ->
        val fechaFormateada =
SimpleDateFormat("yyyy-MM-dd",
Locale.getDefault()).format(paciente.fecha_nacimiento)

        Row(
            modifier = Modifier
                .fillMaxWidth()
                .padding(vertical = 4.dp)
        ) {
            Column {
                Text("ID: ${paciente.id_paciente}")
                Text("Nombre: ${paciente.nombre}
${paciente.apellido}")
                Text("Nacimiento: $fechaFormateada")
            }
            Spacer(modifier = Modifier.weight(1f))
            Button(onClick = {
                pacienteToDelete = paciente
                showDialog = true
            }) {
                Text("Eliminar")
            }
        }
    }
}

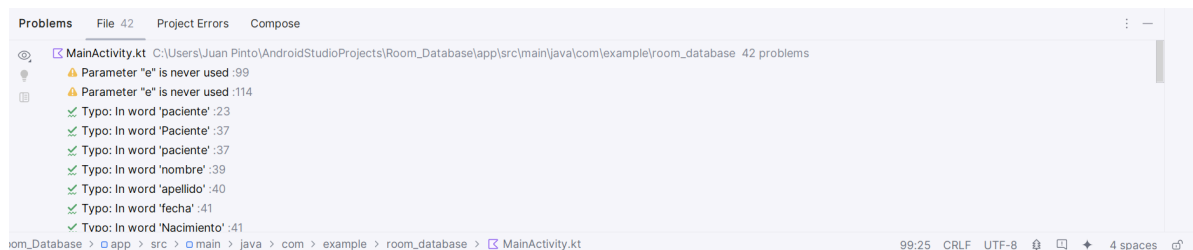
if (showDialog && pacienteToDelete != null) {
    AlertDialog(
        onDismissRequest = { showDialog = false },
        confirmButton = {
            Button(onClick = {

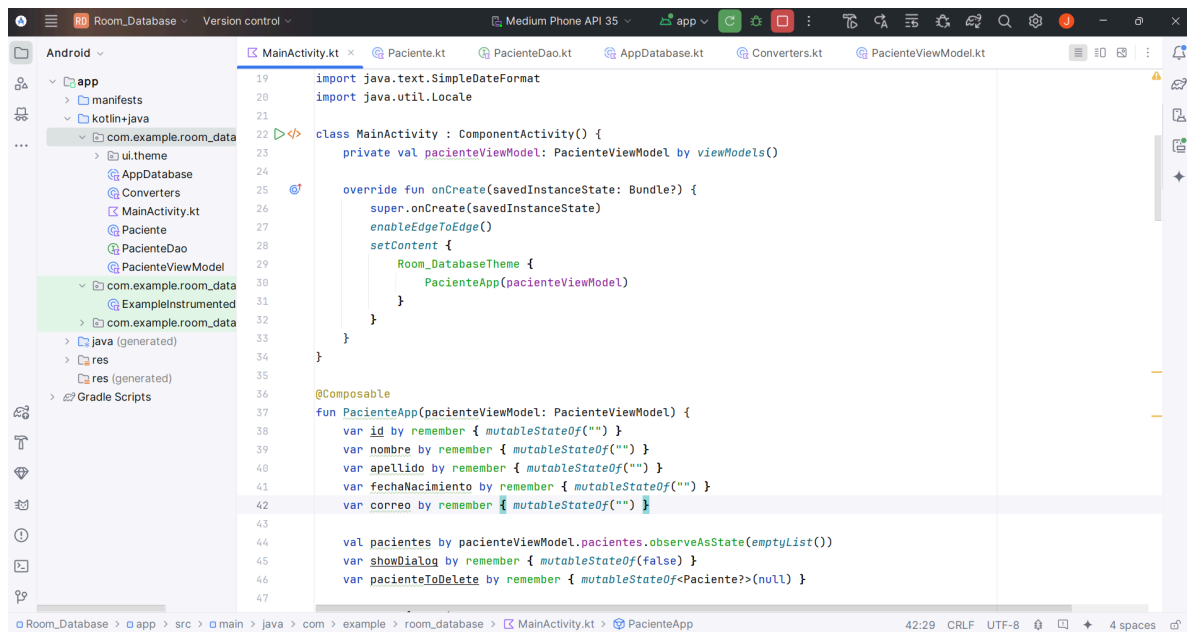
```



```
pacienteViewModel.deletePaciente(pacienteToDelete!!)
        showDialog = false
        pacienteToDelete = null
    }) {
        Text("Confirmar")
    }
},
dismissButton = {
    Button(onClick = { showDialog = false }) {
        Text("Cancelar")
    }
},
title = { Text("Eliminar paciente") },
text = { Text("¿Eliminar a
${pacienteToDelete?.nombre} ${pacienteToDelete?.apellido}?") }
    )
}
}
}
```

Por cierto se llegan a apreciar warnings, esto es debido al parámetro “e” que en teoría era para mostrar algún mensaje de error pero no me dió el tiempo para cuadrarlo bien, por eso aparece que no se usa, de todos modos lo reporto por si las moscas.

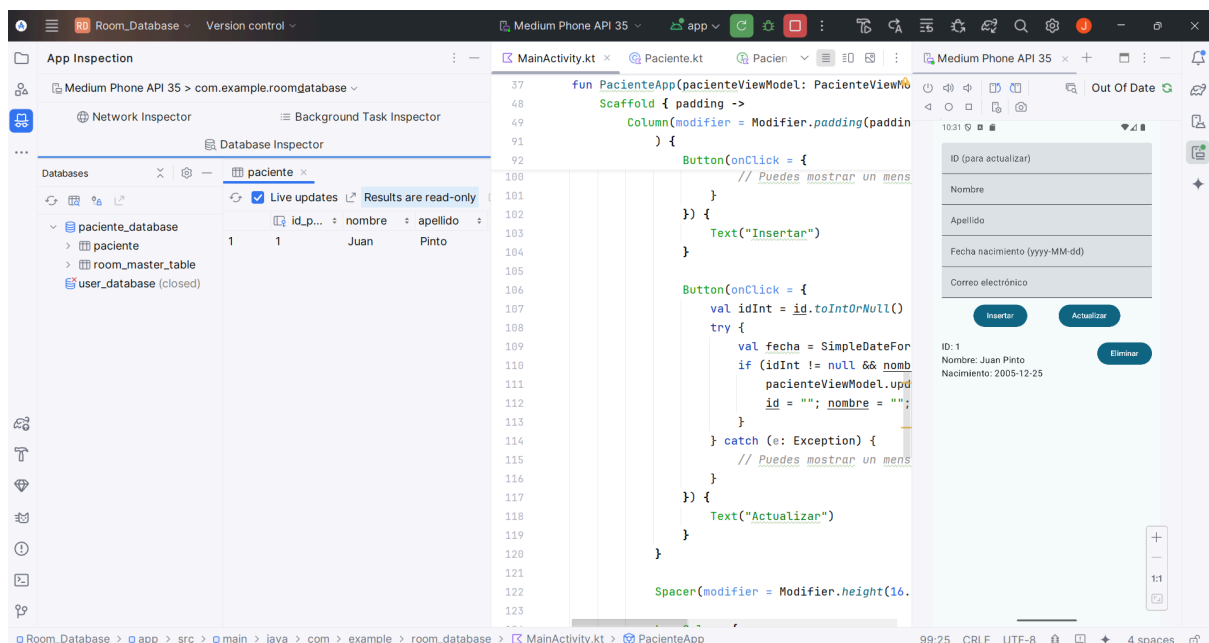




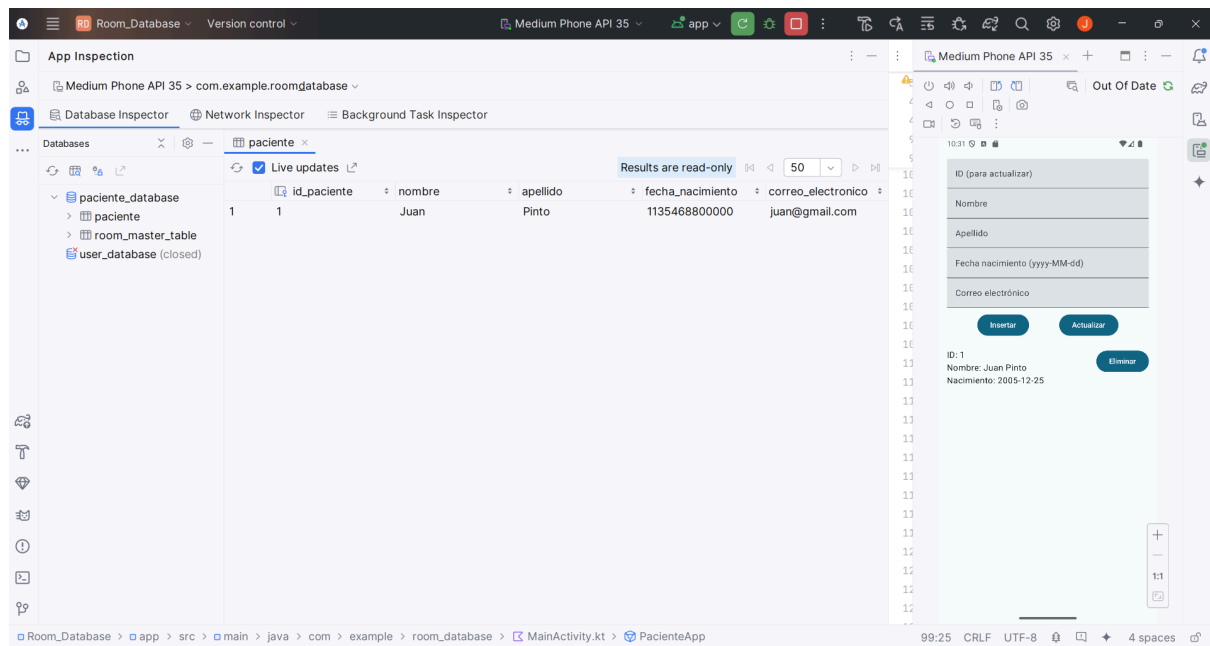
## Testeo de la aplicación y confirmación de su correcto funcionamiento con la base de datos.

Una vez comprobado que no hay errores o warnings en los anteriores puntos procedemos a probar la aplicación ejecutándose en el emulador.

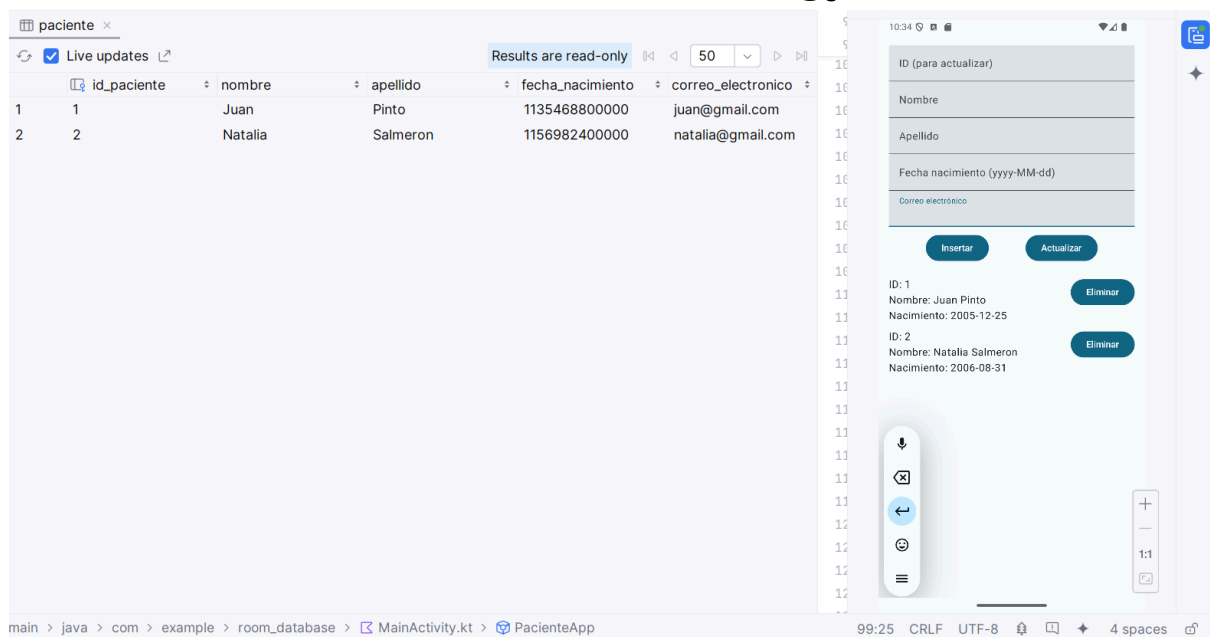
Primero, comenzamos confirmando que no hay errores al abrir la aplicación por primera vez ni tampoco hay errores en el código de la interfaz.



Ahora bien, en la app tenemos la siguiente pantalla al momento de iniciarla.



Crearemos por ejemplo el nombre de “Natalia” y apellido “Salmeron” con fecha de nacimiento en el 2006-08-31 con correo electrónico “natalia@gmail.com”:



Confirmamos que efectivamente en la base de datos se insertó correctamente el dato y también se muestra en la pantalla principal de la aplicación, procedamos a crear otro usuario, esta vez a “Sergio” y apellido “Saavedra” con fecha de nacimiento en el 2005-11-14 con correo electrónico “sergio.saavedra23@gmail.com”:

App Inspection

Medium Phone API 35 > com.example.roomdatabase

Database Inspector Network Inspector Background Task Inspector

Databases

- paciente\_database
  - paciente
    - room\_master\_table
    - user\_database (closed)

Live updates

Results are read-only

id_paciente	nombre	apellido	fecha_nacimiento	correo_electronico
1	Juan	Pinto	1135488800000	juan@gmail.com
2	Natalia	Salmeron	1156982400000	natalia@gmail.com
3	Sergio	Saavedra	1131926400000	sergio.saavedra23@gmail.com

Insertar Actualizar

ID: 1  
Nombre: Juan Pinto  
Nacimiento: 2005-12-25

ID: 2  
Nombre: Natalia Salmeron  
Nacimiento: 2006-08-31

ID: 3  
Nombre: Sergio Saavedra  
Nacimiento: 2005-11-14

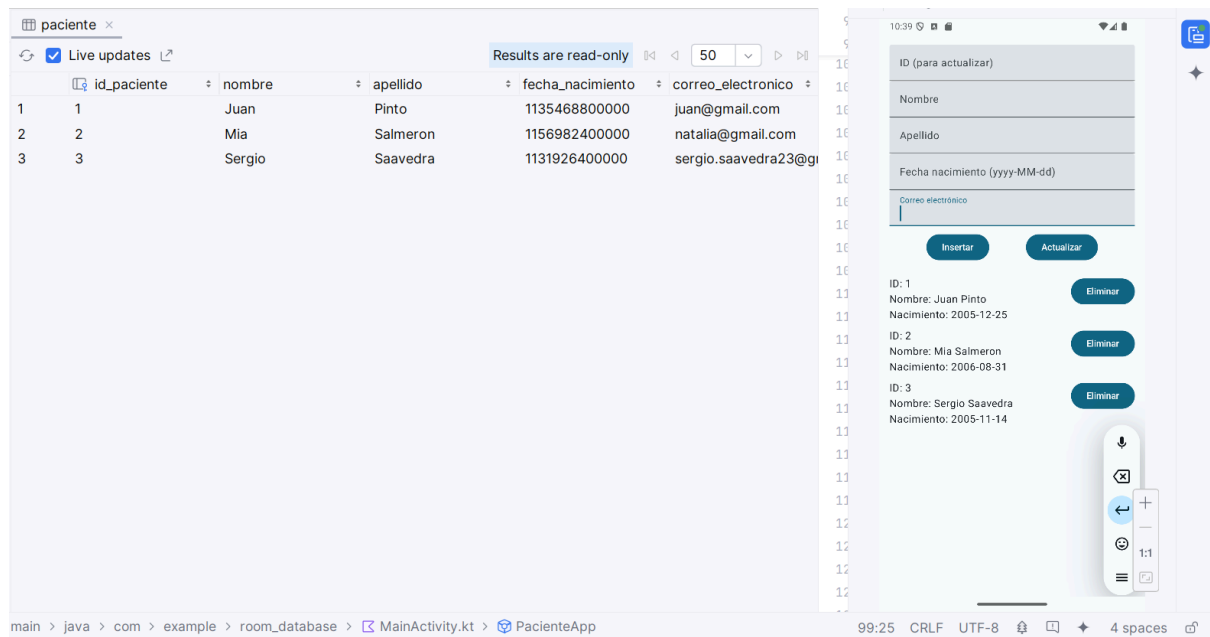
Eliminar

Eliminar

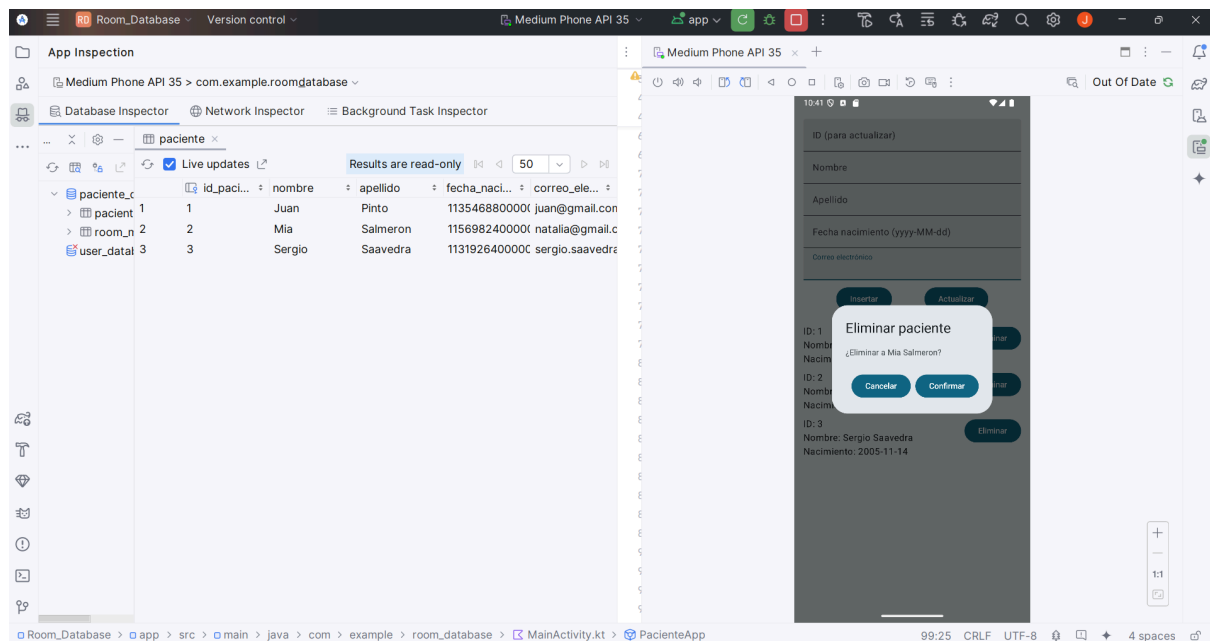
Eliminar

99:25 CRLF UTF-8 4 spaces

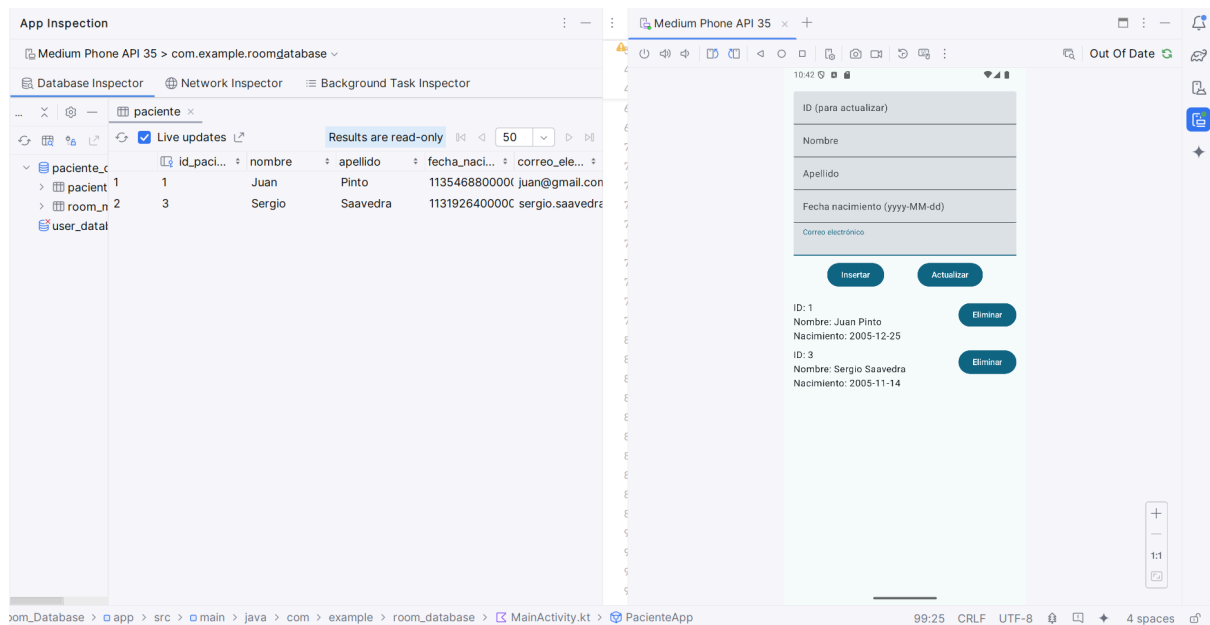
Y podemos confirmar que efectivamente el dato se agregó sin problema alguno, ahora vamos a editar la información, digamos que ahora “Natalia” identificada con ID “2” ya no se llamará de ese modo sino “Mia” conservando los mismos datos, quedaría de esta forma:



Por último, procedemos a eliminar un dato, en este caso eliminemos a “Mia”. Podemos confirmar que antes de eliminar el dato se muestra un mensaje de advertencia, pidiendo confirmación de si continuar o no.



y una vez confirmamos podemos ver que efectivamente el dato se eliminó por completo de la bd.



NOTA: En el **App Inspector** de Android Studio, cuando reviso la base de datos, noto que la fecha de nacimiento no aparece en formato legible como "2005-12-25", sino como un número largo (por ejemplo, 1135468800000). Esto es completamente normal porque Room, gracias al **Converters.kt**, guarda internamente las fechas como milisegundos desde el 1 de enero de 1970 (el llamado *epoch time*). Aunque en el inspector se ve como un número largo, en la aplicación se convierte correctamente de nuevo a un objeto **Date** y se muestra en el formato adecuado para el usuario.

## Actividad 4