Se inicia un nuevo proyecto en blanco, acompañado de la inclusión de la carpeta "backend" dentro del directorio principal de la aplicación. En esta carpeta, se generan dos ficheros, a saber, ElementoListaModel.kt y Database.kt.

El primer archivo engloba la entidad correspondiente a los ítems presentes en la lista de compras, así como su correspondiente DAO. Dicho DAO incluye funciones para insertar, actualizar, eliminar y obtener elementos.

El segundo archivo contiene la configuración esencial de la base de datos, así como la creación de la tabla de elementos, la cual se concreta mediante el archivo previamente mencionado. Asimismo, en este archivo se crea un Singleton destinado a instanciar la base de datos dentro del contexto de la aplicación.

También es fundamental realizar la instalación y la inclusión de las bibliotecas requeridas por Android Studio, como el caso de KTX y KSP, siguiendo las indicaciones necesarias.

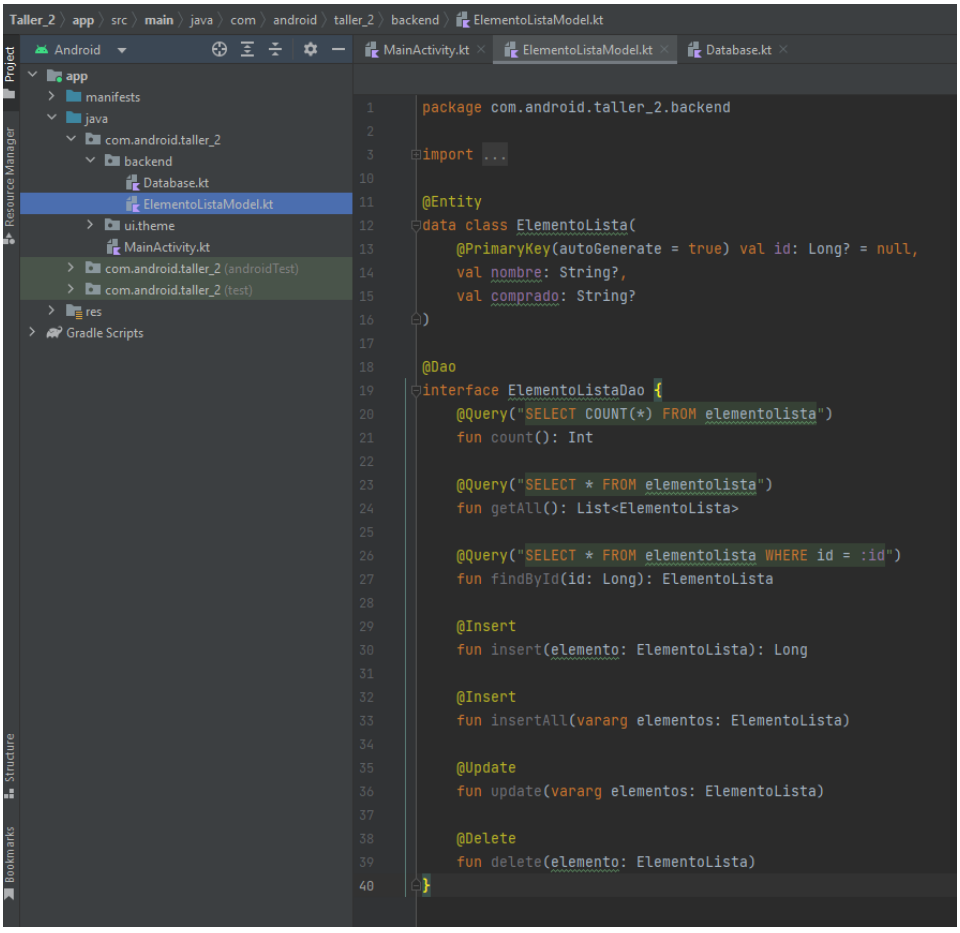
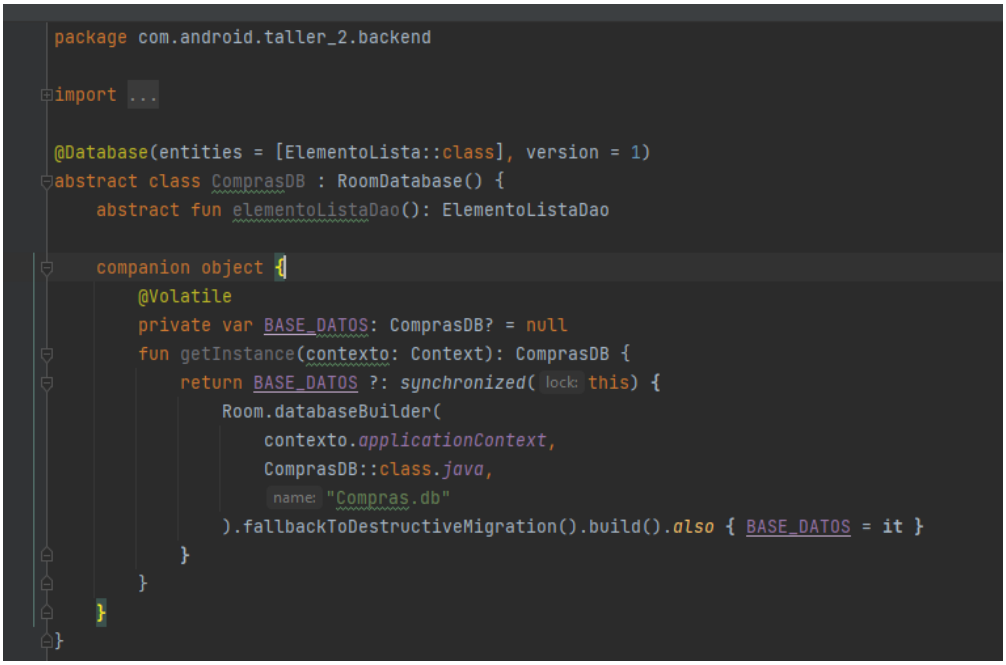


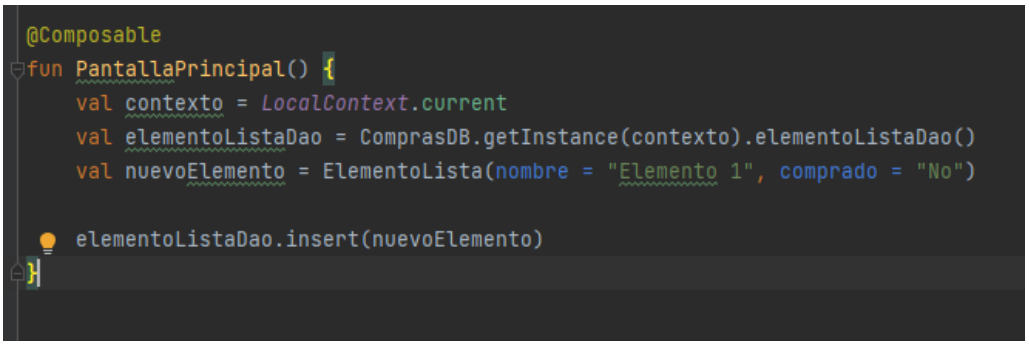
Imagen 1: ElementosListaModel.kt



```
package com.android.taller_2.backend

import ...

@Database(entities = [ElementoLista::class], version = 1)
abstract class ComprasDB : RoomDatabase() {
    abstract fun elementoListaDao(): ElementoListaDao

    companion object {
        @Volatile
        private var BASE_DATOS: ComprasDB? = null
        fun getInstance(contexto: Context): ComprasDB {
            return BASE_DATOS ?: synchronized( lock: this) {
                Room.databaseBuilder(
                    contexto.applicationContext,
                    ComprasDB::class.java,
                     name: "Compras.db"
                ).fallbackToDestructiveMigration().build().also { BASE_DATOS = it }
            }
        }
    }
}
```

Imagen 2: Database

Se establece la carpeta denominada "components" destinada a albergar las interfaces de la aplicación. En el interior de dicha carpeta, se genera el archivo llamado PantallaPrincipal.kt, el cual es invocado por el MainActivity.kt. La mencionada pantalla principal efectúa la instanciación de la base de datos y genera un objeto en la base de datos mediante una operación de inserción básica, cuyos resultados son visibles en el inspector de la aplicación.



```
@Composable
fun PantallaPrincipal() {
    val contexto = LocalContext.current
    val elementoListaDao = ComprasDB.getInstance(contexto).elementoListaDao()
    val nuevoElemento = ElementoLista(nombre = "Elemento 1", comprado = "No")

    elementoListaDao.insert(nuevoElemento)
}
```

Imagen 3: Pantalla principal

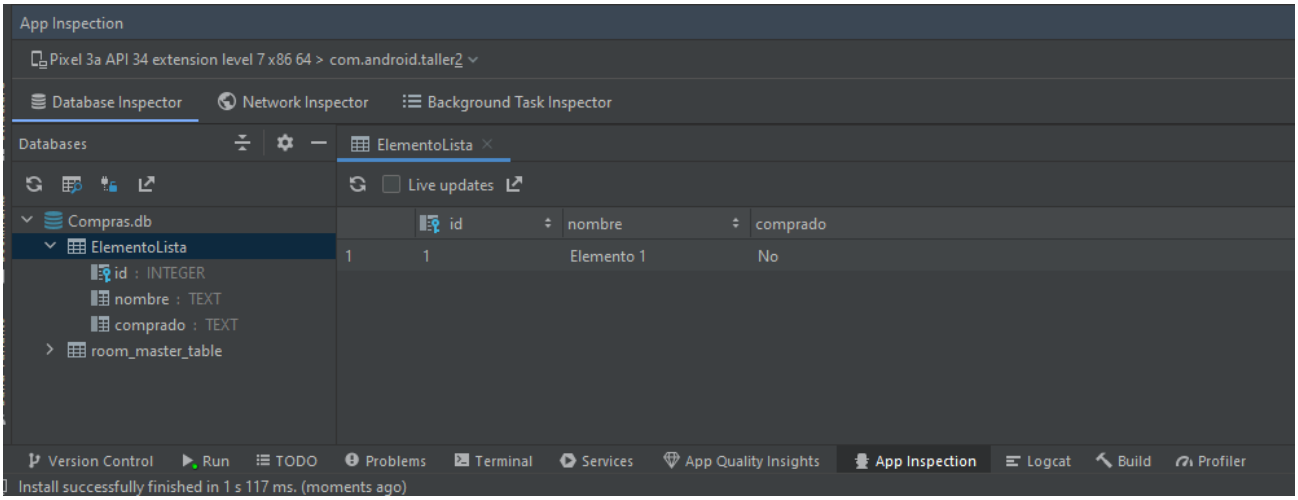


Imagen 4: Inspector de base de datos

Dado que estamos empleando ROOM para mantener una base de datos local, es esencial crear una instancia de esta en la pantalla principal para visualizar la información almacenada. Para lograrlo, emplearemos **LaunchedEffect** y **mutableStateOf** para administrar los cambios en los datos durante los distintos ciclos de vida. Asimismo, utilizaremos acciones para interactuar con la base de datos mediante la interfaz de usuario.

```
enum class Accion { LISTAR, CREAR, EDITAR }

@Composable
fun PantallaPrincipal() {
    val contexto = LocalContext.current

    var elementos by remember { mutableStateOf(emptyList<ElementoLista>()) }

    LaunchedEffect(elementos) { this: CoroutineScope
        withContext(Dispatchers.IO) { this: CoroutineScope
            val database = ComprasDB.getInstance(contexto)
            elementos = database.elementoListaDao().getAll()
        }
    }

    var accion by remember { mutableStateOf(Accion.LISTAR) }
    var seleccion by remember { mutableStateOf<ElementoLista?>( value: null) }
```

Imagen 5: Pantalla Principal

Se implementan las funciones correspondientes para las operaciones en la pantalla principal, donde la acción predeterminada es la de exhibir los elementos almacenados en la base de datos. Para ello, se crea una nueva función componible llamada "ListaElementos" en el mismo archivo. En esta función, se aprovecha el componente "LazyColumn" para presentar los elementos. Además, se incorpora un título, un mensaje predeterminado si no hay elementos en la base de datos, y un botón para añadir nuevos elementos a la lista.

Cada entrada en el "LazyColumn" utiliza otra pantalla para mostrar el nombre del elemento y un casillero de verificación que indica si el elemento ha sido adquirido o no. Este estado también se gestiona mediante el uso de "mutableStateOf" y se actualiza utilizando el patrón "Update" del objeto de acceso a datos (DAO).

```
@Composable
fun ListaElementos(
    elementos: List<ElementoLista>,
    addAction: () -> Unit,
    editAction: (ElementoLista) -> Unit
) {
    Column(
        modifier = Modifier.fillMaxSize()
    ) { this: ColumnScope
        Text(
            text = "Lista de Compras",
            modifier = Modifier
                .fillMaxWidth()
                .padding(16.dp),
            textAlign = TextAlign.Center,
            fontWeight = FontWeight.Bold,
            fontSize = 24.sp,
            color = MaterialTheme.colorScheme.secondary
        )

        val elementosDB: List<ElementoLista> = elementos.ifEmpty {
            val dao = ComprasDB.getInstance(LocalContext.current).elementoListaDao()
            dao.getAll() ^ifEmpty
        }
```

Imagen 7: ListaElementos

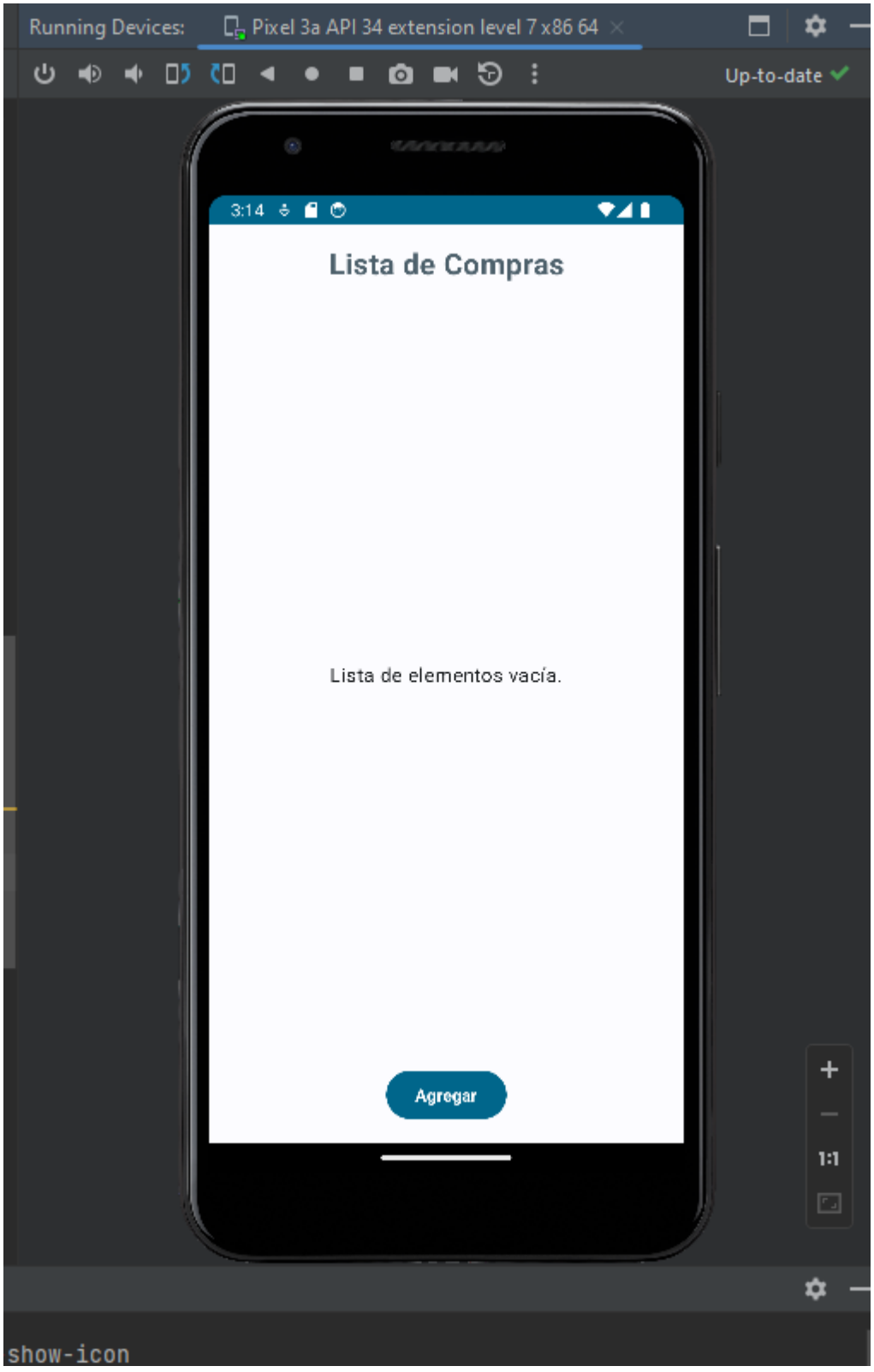
Dado que la aplicación fue ejecutada en varias ocasiones con el fin de verificar el funcionamiento del código, se generaron múltiples elementos de prueba en la base de datos. Esto permitió probar el correcto funcionamiento de los casilleros de verificación y su capacidad para reflejarse en la base de datos.

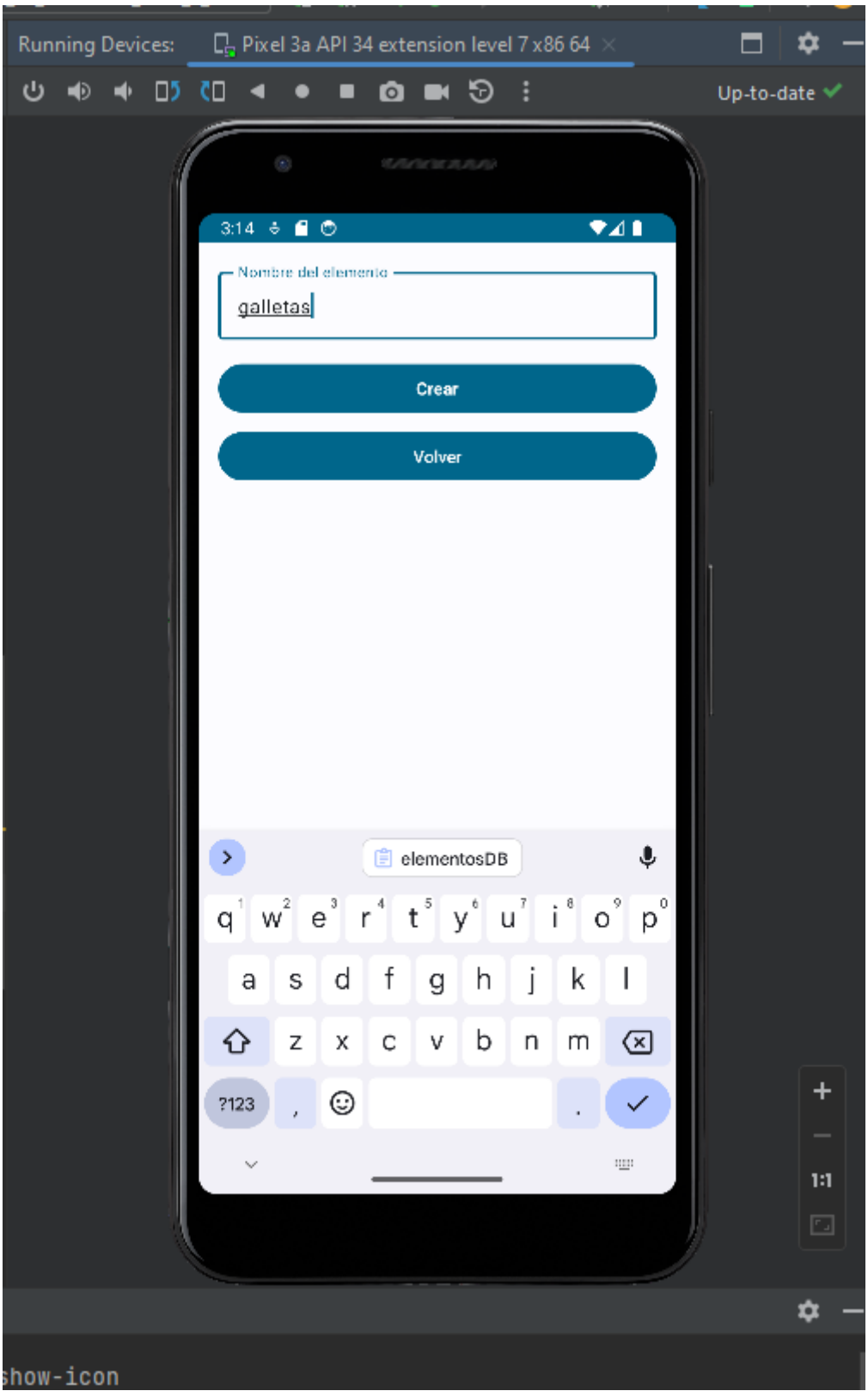
Ahora, procedemos a crear el último archivo denominado ElementoDetalles.kt. En este archivo, se proporcionará la funcionalidad para guardar un elemento, ya sea creándolo o actualizándolo, y también para eliminarlo si es necesario.

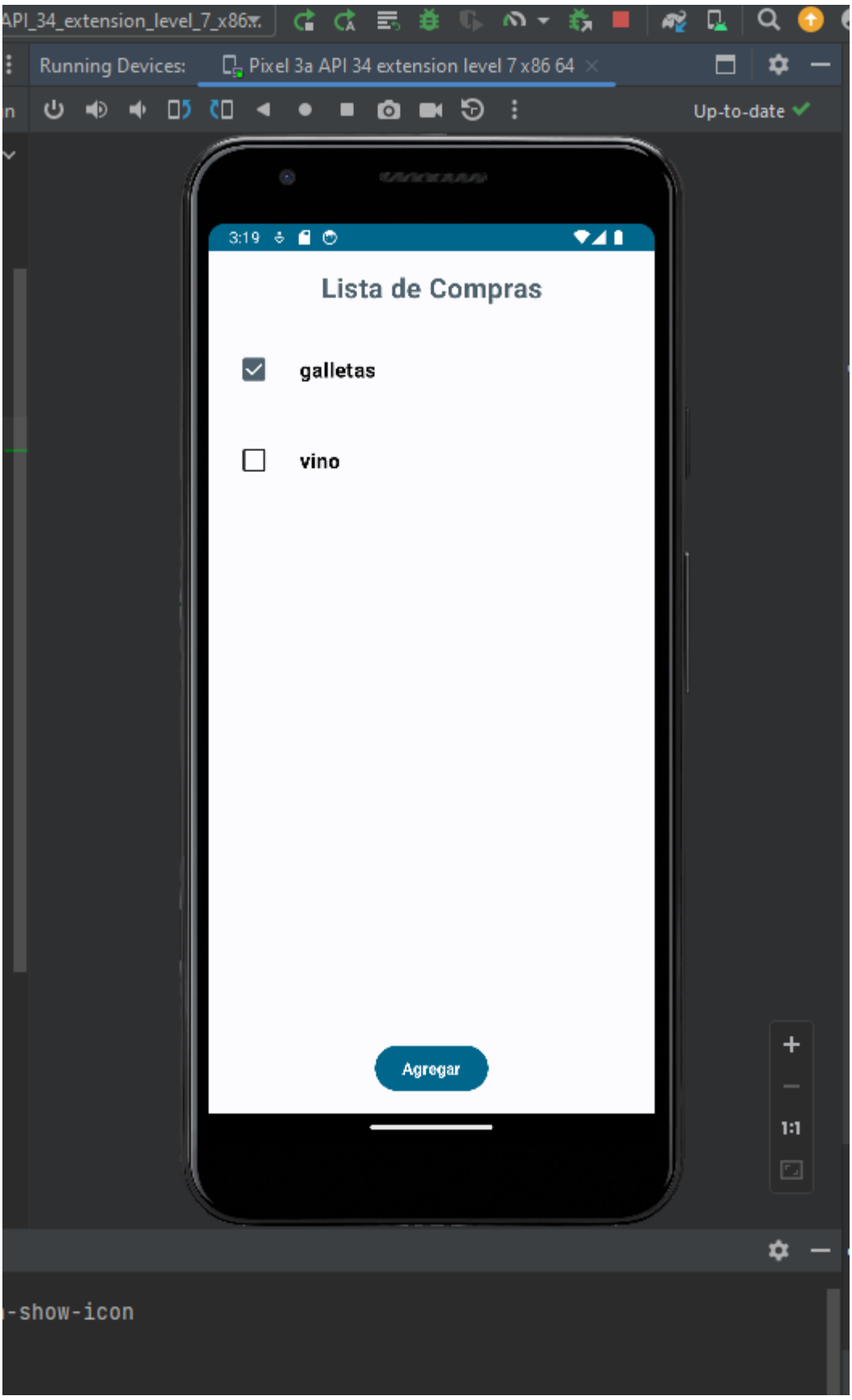
```kotlin
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun ElementoDetalles(elemento: ElementoLista?, saveAction: () -> Unit) {
    val elementoDao = ComprasDB.getInstance(LocalContext.current).elementoListaDao()
    val coroutineScope = rememberCoroutineScope()

    var nombre by remember { mutableStateOf( value: elemento?.nombre ?: "") }
    var buttonText by remember { mutableStateOf(if (elemento?.id != null) "Guardar" else "Crear") }

    Column(
        horizontalAlignment = Alignment.CenterHorizontally,
        modifier = Modifier
            .fillMaxSize()
            .padding(16.dp)
    ) { this: ColumnScope
        OutlinedTextField(
            value = nombre,
            onValueChange = { nombre = it },
            label = { Text( text: "Nombre del elemento") },
            singleLine = true,
            modifier = Modifier
                .fillMaxWidth()
                .padding(bottom = 16.dp)
        )
```

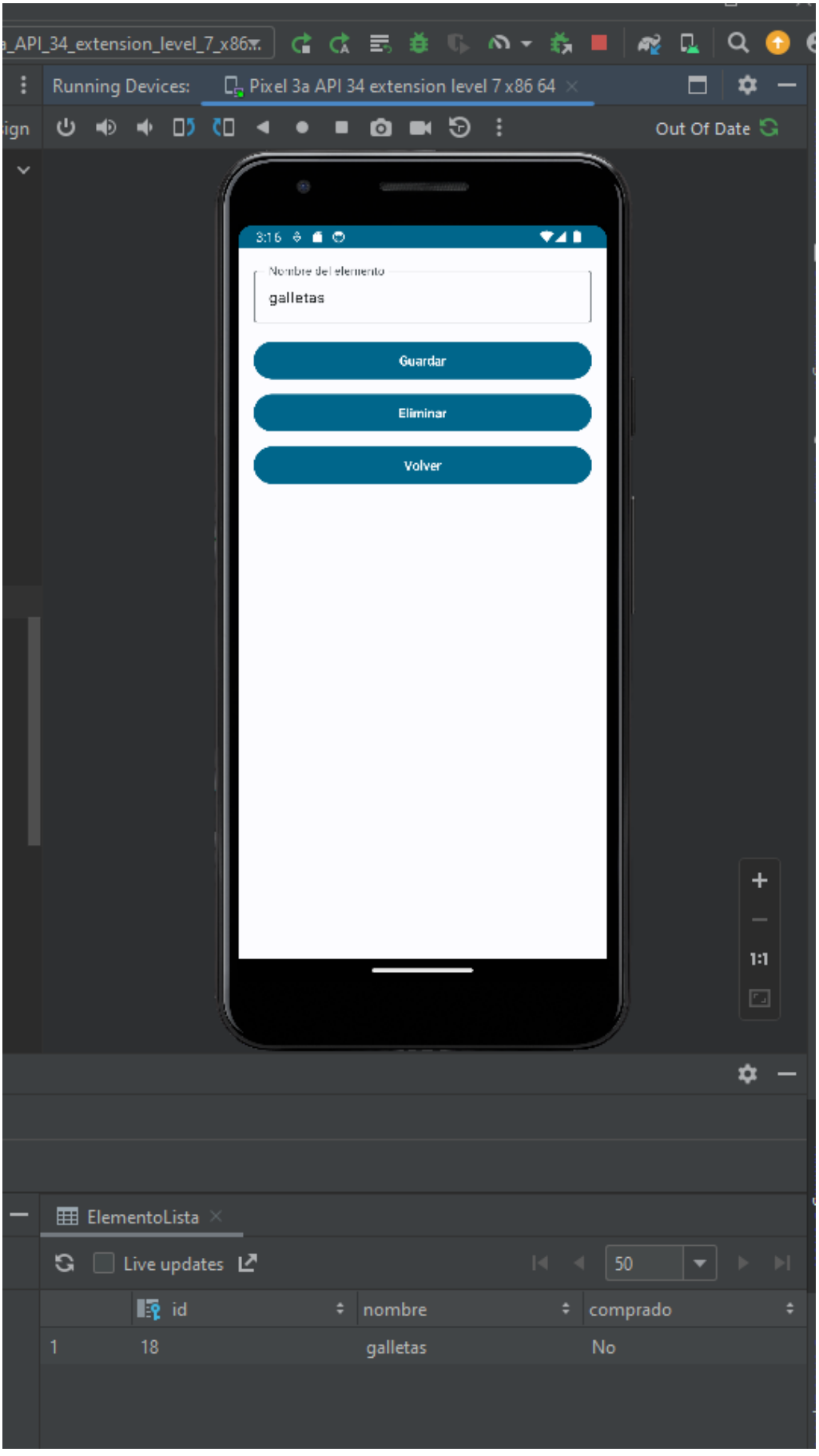
Imagen 8: ElementoDetalles

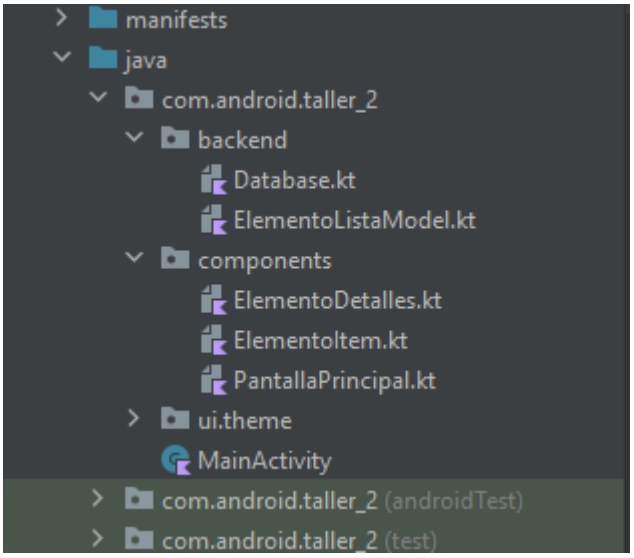
Ahora se puede probar la aplicación y verificar en el inspector de base de datos local.


Running Devices:
Pixel 3a API 34 extension level 7 x86 64
Up-to-date
3:14
Lista de Compras
Lista de elementos vacía.
Agregar
1:1
show-icon


Running Devices:
Pixel 3a API 34 extension level 7 x86 64
Up-to-date
3:14
Nombre del elemento
galletas
Crear
Volver
elementosDB
show-icon


Running Devices:
Pixel 3a API 34 extension level 7 x86 64
Up-to-date
3:19
Lista de Compras
galletas
vino
Agregar
1:1
-show-icon


Running Devices:
Pixel 3a API 34 extension level 7 x86 64
Out Of Date
3:16
Nombre del elemento
galletas
Guardar
Eliminar
Volver
1:1
ElementoLista
Live updates
50
id
nombre
comprado
1
18
galletas
No


manifests
java
com.android.taller_2
backend
Database.kt
ElementoListaModel.kt
components
ElementoDetalles.kt
ElementoItem.kt
PantallaPrincipal.kt
ui.theme
MainActivity
com.android.taller_2 (androidTest)
com.android.taller_2 (test)