



|  |  |
| --- | --- |
| **Nombre de la Unidad:** | **RECURSOS Y PERSISTENCIA** |
| **Nombre de la Asignatura:** | **PROGRAMACIÓN MOBILE** |
| **Alumno:** | **EDUARDO ANTONIO DÍAZ GALLARDO** |

**TALLER 2 /** Evaluación Unidad N° 2

**INTRODUCCION**

Según requerimientos se desarrolla aplicación para Android en Android Studio la cual consistente en una lista de compras que permite principalmente al usuario agregar, marcar como comprado y eliminar elementos de la lista, además de otras funcionalidades requeridas.

**ENLACE AL PROYECTO:**

[**https://github.com/eduarkdom/taller2-mobile**](https://github.com/eduarkdom/taller2-mobile)

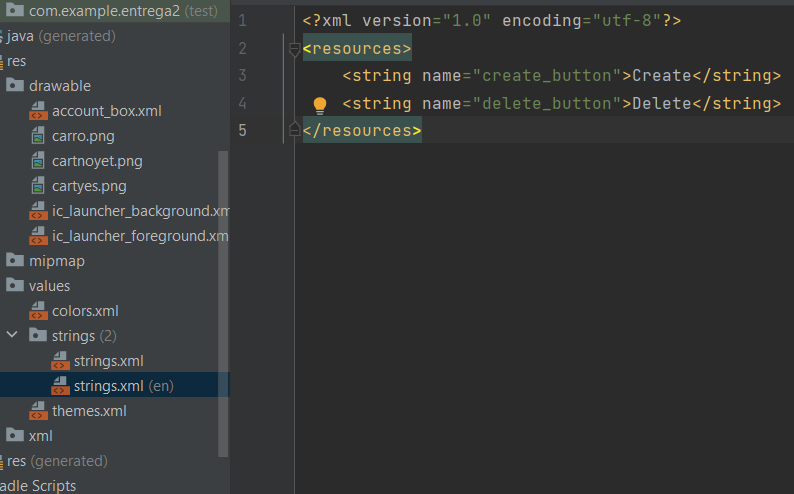
**EVIDENCIA DESARROLLO**

@file:Suppress("UNUSED\_EXPRESSION")  
  
package com.example.entrega2  
  
import android.os.Bundle  
import android.util.Log  
import androidx.activity.ComponentActivity  
import androidx.activity.compose.setContent  
import androidx.compose.foundation.Image  
import androidx.compose.foundation.background  
import androidx.compose.foundation.clickable  
import androidx.compose.foundation.layout.Arrangement  
import androidx.compose.foundation.layout.Box  
import androidx.compose.foundation.layout.Column  
import androidx.compose.foundation.layout.Row  
import androidx.compose.foundation.layout.Spacer  
import androidx.compose.foundation.layout.fillMaxSize  
import androidx.compose.foundation.layout.fillMaxWidth  
import androidx.compose.foundation.layout.height  
import androidx.compose.foundation.layout.padding  
import androidx.compose.foundation.layout.size  
import androidx.compose.foundation.layout.width  
import androidx.compose.foundation.lazy.LazyColumn  
import androidx.compose.foundation.lazy.items  
import androidx.compose.material.icons.Icons  
import androidx.compose.material.icons.filled.*Add*import androidx.compose.material3.Button  
import androidx.compose.material3.Checkbox  
import androidx.compose.material3.ExperimentalMaterial3Api  
import androidx.compose.material3.ExtendedFloatingActionButton  
import androidx.compose.material3.Icon  
import androidx.compose.material3.Scaffold  
import androidx.compose.material3.SnackbarHost  
import androidx.compose.material3.SnackbarHostState  
import androidx.compose.material3.Text  
import androidx.compose.material3.TextField  
import androidx.compose.runtime.Composable  
import androidx.compose.runtime.LaunchedEffect  
import androidx.compose.runtime.mutableStateOf  
import androidx.compose.runtime.remember  
import androidx.compose.runtime.rememberCoroutineScope  
import androidx.compose.ui.Alignment  
import androidx.compose.ui.Modifier  
import androidx.compose.ui.graphics.Color  
import androidx.compose.ui.platform.*LocalContext*import androidx.compose.ui.res.painterResource  
import androidx.compose.ui.res.stringResource  
import androidx.compose.ui.unit.dp  
import com.example.entrega2.db.AppDatabase.Companion.getInstance  
import com.example.entrega2.db.Producto  
import kotlinx.coroutines.Dispatchers  
import kotlinx.coroutines.launch  
import kotlinx.coroutines.withContext  
  
  
class MainActivity : ComponentActivity() {  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 *setContent* **{** AppProductosUI()  
 **}** }  
}  
enum class Accion {  
 *LISTAR*, *CREAR*, *EDITAR*}  
@Composable  
fun AppProductosUI() {  
 val contexto = *LocalContext*.current  
 val (productos, setProductos) = remember**{** *mutableStateOf*(  
 *emptyList*<Producto>() ) **}** val (seleccion, setSeleccion) = remember**{** *mutableStateOf*<Producto?>(null) **}** val (accion, setAccion) = remember**{** *mutableStateOf*(Accion.*LISTAR*) **}** LaunchedEffect(productos) **{** withContext(Dispatchers.IO) **{** val db = getInstance( contexto )  
 setProductos(db.ProductoDao().getAll() )  
 Log.v("AppProductosUI", "LaunchedEffect()")  
 **}  
 }** val onSave = **{** setAccion(Accion.*LISTAR*)  
 setProductos(*emptyList*())  
 **}** when(accion) {  
 Accion.*CREAR* -> ProductoFormUI(null, onSave)  
 Accion.*EDITAR* -> ProductoFormUI(seleccion, onSave)  
 else -> ProductosListadoUI(  
 productos,  
 onAdd = **{** setAccion( Accion.*CREAR* ) **}**,  
 onEdit = **{** producto **->** setSeleccion(producto)  
 setAccion( Accion.*EDITAR*)  
 **}** )  
 }  
}  
@OptIn(ExperimentalMaterial3Api::class)  
@Composable  
fun ProductosListadoUI(productos:List<Producto>, onAdd:() -> Unit = **{}**,  
 onEdit:(c:Producto) -> Unit = **{}**) {  
 Scaffold(  
 floatingActionButton = **{** ExtendedFloatingActionButton(  
 onClick = **{** onAdd() **}**,  
 icon = **{** Icon(  
 Icons.Filled.*Add*,  
 contentDescription = stringResource(id = R.string.*create\_button*))  
 **}**,  
 text = **{** Text(stringResource(id = R.string.*create\_button*)) **}** )  
 **}** ) **{** contentPadding **->** if( productos.*isNotEmpty*() ) {  
 LazyColumn(  
 modifier = Modifier.*fillMaxSize*()  
 ) **{** *items*(productos) **{** producto **->** ProductoItemUI(producto) **{** onEdit(producto)  
 **}  
 }  
 }** } else {  
 Box(modifier = Modifier  
 .*fillMaxSize*()  
 .*padding*(contentPadding), contentAlignment = Alignment.Center) **{** Text("No hay productos que mostrar")  
 **}** }  
 **}**}  
@Composable  
fun ProductoItemUI(producto: Producto, onClick: () -> Unit = **{}**) {  
 val contexto = *LocalContext*.current  
 Row(  
 verticalAlignment = Alignment.CenterVertically,  
 modifier = Modifier  
 .*fillMaxWidth*()  
 .*clickable* **{** onClick **}** ) **{** Spacer(modifier = Modifier.*width*(20.*dp*))  
 Image(  
 painter = painterResource(id = R.drawable.*carro*),  
 contentDescription = "Imagen Carro de Compras"  
 )  
 Spacer(modifier = Modifier.*width*(20.*dp*))  
 Column **{** Row(  
 verticalAlignment = Alignment.CenterVertically,  
 ) **{** Checkbox(  
 checked = producto.comprado,  
 onCheckedChange = **{** isChecked **->** producto.comprado = isChecked  
 val dao = getInstance(contexto).ProductoDao()  
 dao.update(producto)  
 **}**,  
 modifier = Modifier  
 .*size*(40.*dp*)  
 .*background*(Color.Transparent)  
 )  
 if (producto.comprado) {  
 Icon(  
 painter = painterResource(id = R.drawable.*cartyes*),  
 contentDescription = "Producto comprado"  
 )  
 } else {  
 Icon(  
 painter = painterResource(id = R.drawable.*cartnoyet*),  
 contentDescription = "Producto no comprado"  
 )  
 }  
 **}  
 }  
 }**}  
@OptIn(ExperimentalMaterial3Api::class)  
@Composable  
fun ProductoFormUI(c: Producto?, onSave: () -> Unit = **{}**) {  
 val contexto = *LocalContext*.current  
 val (producto, setProducto) = remember **{** *mutableStateOf*(c?.producto ?: "") **}** val (comprado, setComprado) = remember **{** *mutableStateOf*(c?.comprado.*toString*())  
 **}** val coroutineScope = rememberCoroutineScope()  
 val snackbarHostState = remember **{** SnackbarHostState() **}** Scaffold(  
 snackbarHost = **{** SnackbarHost(snackbarHostState) **}** ) **{** paddingValues **->** Column(  
 verticalArrangement = Arrangement.Center,  
 horizontalAlignment = Alignment.CenterHorizontally,  
 modifier = Modifier  
 .*fillMaxSize*()  
 .*padding*(paddingValues)  
 ) **{** Image(  
 painter = painterResource(id = R.drawable.*account\_box*),  
 contentDescription = "Imagen de usuario"  
 )  
 TextField(value = producto, onValueChange = **{** setProducto(**it**) **}**, label = **{** Text("Nombre Producto") **}**)  
 Spacer(modifier = Modifier.*height*(10.*dp*))  
 TextField(  
 value = comprado,  
 onValueChange = **{** setComprado(**it**) **}**,  
 label = **{** Text("Comprado") **}** )  
 Spacer(modifier = Modifier.*height*(20.*dp*))  
 Button(onClick = **{** coroutineScope.*launch*(Dispatchers.IO) **{** val dao = getInstance(contexto).ProductoDao()  
 val product = Producto(c?.id ?: 0, producto, comprado.*toBoolean*())  
 if (product.id > 0) {  
 dao.update(product)  
 } else {  
 dao.insert(product)  
 }  
 snackbarHostState.showSnackbar("Se ha guardado a ${product.producto}")  
 onSave()  
 **}  
 }**) **{** var textoGuardar = "Crear"  
 if ((c?.id ?: 0) > 0) {  
 textoGuardar = "Guardar"  
 }  
 Text(textoGuardar)  
 **}** if ((c?.id ?: 0) > 0) {  
 Button(onClick = **{** coroutineScope.*launch*(Dispatchers.IO) **{** val dao = getInstance(contexto).ProductoDao()  
 snackbarHostState.showSnackbar("Eliminando el producto de ${c?.producto}")  
 if (c != null) {  
 dao.delete(c)  
 }  
onSave()  
**}  
}**) **{** Text(stringResource(id = R.string.*delete\_button*))  
**}**}  
**}  
}**}

package com.example.entrega2.db  
  
import androidx.room.Dao  
import androidx.room.Delete  
import androidx.room.Insert  
import androidx.room.Query  
import androidx.room.Update  
  
@Dao  
interface ProductoDao {  
  
 @Query("SELECT COUNT(\*) FROM producto")  
 fun count():Int  
 @Query("SELECT \* FROM producto ORDER BY comprado ASC")  
 fun getAll():List<Producto>  
 @Query("SELECT \* FROM producto WHERE id = :id")  
 fun findById(id:Int):Producto  
 @Insert  
 fun insert(producto:Producto):Long  
 @Insert  
 fun insertAll(vararg productos:Producto)  
 @Update  
 fun update(vararg productos:Producto)  
 @Delete  
 fun delete(producto:Producto)  
}

package com.example.entrega2.db  
  
import androidx.room.Entity  
import androidx.room.PrimaryKey  
  
@Entity  
data class Producto(  
 @PrimaryKey(autoGenerate = true)  
 val id:Int,  
 var producto:String,  
 var comprado:Boolean  
)

package com.example.entrega2.db  
  
import android.content.Context  
import androidx.room.Database  
import androidx.room.Room  
import androidx.room.RoomDatabase  
  
@Database(entities = [Producto::class], version = 1)  
abstract class AppDatabase : RoomDatabase() {  
  
 abstract fun ProductoDao(): ProductoDao  
  
 companion object {  
 // Volatile asegura que sea actualizada la propiedad  
 // atómicamente  
 @Volatile  
 private var BASE\_DATOS : AppDatabase? = null  
  
 fun getInstance(contexto: Context): AppDatabase {  
 //synchronized previene el acceso de múltiples threads de manera simultánea  
 return BASE\_DATOS ?: *synchronized*(this) **{** Room.databaseBuilder(  
 contexto.*applicationContext*,  
 AppDatabase::class.*java*,  
 "productos.db"  
 )  
 .fallbackToDestructiveMigration()  
 .build()  
 .*also* **{** BASE\_DATOS = **it }  
 }** }  
 }  
}



**Pauta de Autoevaluación:**

A continuación, te invitamos a responder una pauta de autoevaluación, para que verifiques lo aprendido en esta actividad y el logro del o los objetivos planteados inicialmente. Recuerda, en caso de presentar dudas, debes participar de forma permanente en los foros, donde el docente te retroalimentará y dará respuesta a tus inquietudes.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Criterios de revisión del informe con la información recopilada:** | **Completamente logrado**  **(4)** | **Medianamente logrado**  **(3)** | **Parcialmente logrado**  **(2)** | **No logrado**  **(0)** |
| 1. Utilicé correctamente los recursos de texto para hacer mi app multilenguaje | 4 |  |  |  |
| 1. Utilicé recursos gráficos para hacer la interfaz más amigable y entendible | 4 |  |  |  |
| 1. Codifiqué todos los métodos necesarios para recuperar, crear, editar y eliminar registros desde la BD | 4 |  |  |  |
| 1. Utilicé correctamente Corrutinas para no bloquear la interfaz del dispositivo cuando era necesario hacer tareas sobre la BD | 4 |  |  |  |
| 1. Utilicé correctamente las listas dinámicas para mostrar los listados de información | 4 |  |  |  |
| **Puntaje Total** | **20 puntos** | | | |
| **Puntaje Obtenido** | **20 puntos obtenidos** | | | |

