UNIVERSIDAD CENTRAL DEL ECUADOR

“OMNIUM POTENTIOR EST SAPIENTIA”

FACULTAD DE INGENIERÍA Y CIENCIAS APLICADAS

DISPOSITIVOS MÓVILES

Integrantes:

1. Chamba Carrión Jordy Pedro
2. Loya Cadena Bryan Eduardo
3. Luna Grijalva Joel Joshua
4. Parra Vasques Rensso Nicolay
5. Pozo Maldonado Kevin Fernando
6. Tapia Rea Freddy Xavier

TEMA:

Diseño App Básica

SEMESTRE: Octavo

FECHA: 18/05/2025

Modelo de Navegación de la Aplicación

1. **Tipo de Navegación:**

   La aplicación utiliza un sistema de navegación basado en pantallas (**Screens**) implementado con el componente **NavController** del paquete **androidx.navigation:navigation-compose**, propio de **Jetpack Compose** para Android.

   Este tipo de navegación permite:

   1. Definir rutas fijas o dinámicas para cada pantalla.
   2. Manejar una pila de navegación (back stack) para retroceder entre pantallas.
   3. Realizar transiciones explícitas usando comandos como:
      1. **navController.navigate("ruta")**
      2. **navController.popBackStack()**
      3. **popUpTo {inclusive = true}**

2. **Estructura de Rutas:**

A continuación, se detallan las pantallas disponibles y sus respectivas rutas:

PANTALLA	RUTA	DESCRIPCIÓN
Login	"login"	Pantalla de inicio de sesión
Registro	"register"	Registro de nuevos usuarios
Home	"home"	Pantalla principal
Insertar Vehículo	"insertVehiculo"	Alta de vehículos
Editar Vehículo	"editVehiculo/{placa}"	Edición de vehículo por placa
Eliminar Vehículo	"eliminarVehiculo"	Baja de vehículos

Nota: *Se utilizan rutas estáticas y dinámicas, permitiendo la edición de vehículos mediante parámetros en la ruta.*

3. **Lógica de Navegación:**
   1. **Navegación Simple:**

      Se usa el método básico para navegar entre pantallas sin modificar la pila **navController.navigate("ruta")**

Ejemplo:

Ir desde el Login al Registro:

Text ("¿No tienes cuenta? Regístrate aquí").clickable {navController.navigate("register")}

2. **Regresar a una Pantalla Específica:**

Para limpiar la pila hasta cierta pantalla:

navController.navigate("ruta_destino") { popUpTo("ruta_inicio") { inclusive = true }}

Ejemplo:

Cerrar sesión y volver al Login:

navController.navigate("login") {popUpTo("home") {inclusive = true}}

3. **Regreso Automático:**

Después de operaciones exitosas (como guardar o editar), se regresa a la pantalla anterior:

navController.popBackStack()

Usado comúnmente en:

1. Guardar cambios en edición de vehículo.
2. Registrar usuario.
3. Agregar o eliminar vehículos.

4. **Transiciones Entre Pantallas:**

ORIGEN	DESTINO	ACCIÓN
Login	Registro	navigate("register")
Registro	Login	navigate("login")+popUpTo
Login	Home	navigate("home")+popUpTo
Home	Insertar Vehículo	navigate("insertVehiculo")
Home	Editar Vehículo	navigate("editVehiculo/${placa}")
Home	Eliminar Vehículo	navigajpchambacte("eliminarVehiculo")
Cerrar Sesión	Login	navigate("login")+popUpTo

5. **Parámetros Dinámicos:**
    1. **Edición del Vehículo:**
    2. Ruta: **"editVehiculo/{placa}"**
    3. Uso: Desde la pantalla de Home al hacer clic en “Editar” en una tarjeta.
    4. Ejemplo:
    5. **navController.navigate("editVehiculo/${vehiculo.placa}")**

6. **Pantallas Principales:**
    1. **LoginScreen:**

        Pantalla inicial donde el usuario ingresa credenciales.

        Valida:

        1. Usuario: solo letras, números o _ (4–16 caracteres)
        2. Contraseña: mínimo 6 caracteres, combinación de letras y números
        3. Si es válido, navega a Home.

    2. **RegisterScreen:**

        Permite registrar nuevos usuarios.

        1. Validaciones similares a Login.
        2. Si el registro es exitoso, navega a Login.

    3. **HomeScreen:**

        Pantalla principal con lista de vehículos.

        Muestra datos del vehículo: marca, modelo, color, costo, etc.

        Botones para:

        1. Agregar vehículo → "insertVehiculo"
        2. Editar vehículo → "editVehiculo/{placa}"
        3. Eliminar vehículo → "eliminarVehiculo"

    4. **InsertVehiculoScreen:**

        Formulario para insertar un nuevo vehículo.

        Validaciones:

        1. Placa: formato ABC123

2. Marca, modelo, color: solo letras y espacios
3. Año: entre 1990 y 2025
4. Costo: número positivo

5. **EdicionVehiculoScreen:**

    1. Formulario prellenado con los datos del vehículo seleccionado.

    2. Similar a Insertar, pero carga datos previos.

6. **EliminarVehiculoScreen:**

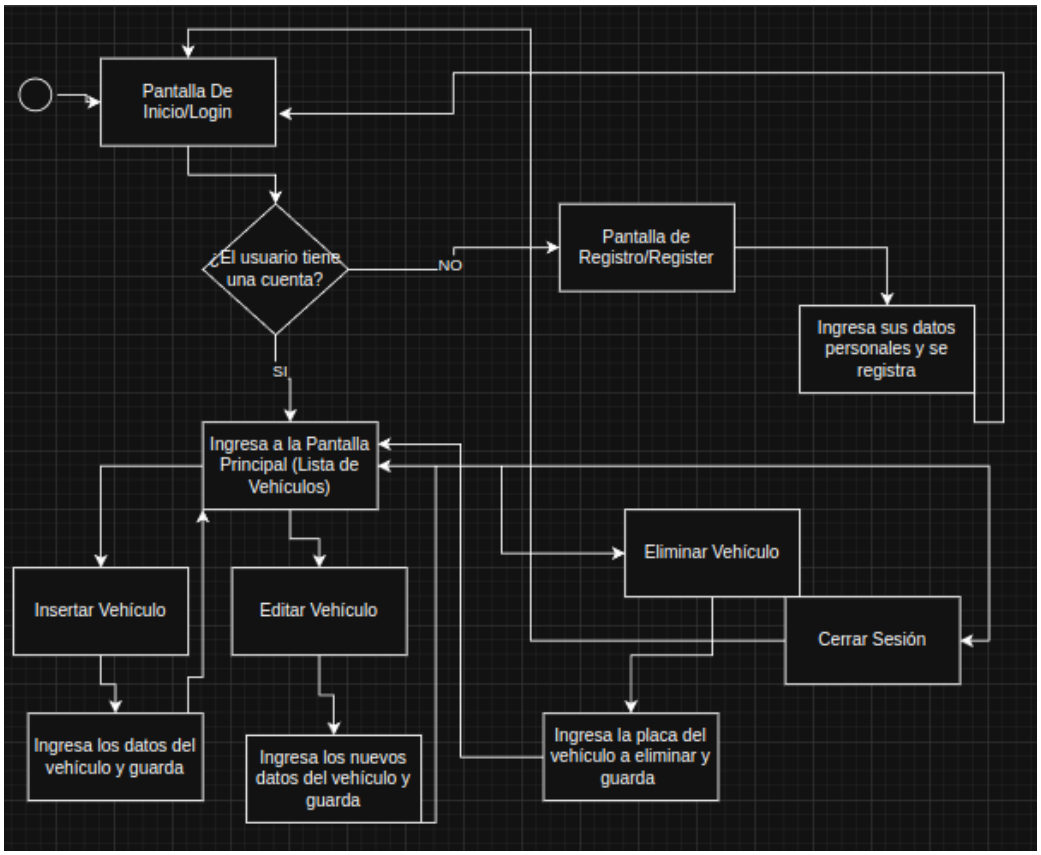
    Solicita la placa del vehículo a eliminar.

    1. Si existe, lo borra de la base de datos.

7. Consideraciones Técnicas:

1. **Validaciones locales:** Cada pantalla tiene su propia lógica de validación de campos.
2. **Gestión de estado local:** Se usan mutableStateOf() para manejar el estado dentro de cada pantalla.
3. **Persistencia:** Se usa una clase DBHelper para interactuar con SQLite.

8. Diagrama de Flujo de Navegación:



9. **Documentación del Código Fuente:**
    1. **Modelo:**

        Esta carpeta contiene la estructura de datos principal que representa a los vehículos registrados en la aplicación. En el patrón MVC (Modelo - Vista - Controlador), esta sección corresponde al Modelo, el cual define cómo se almacenan y manipulan los datos esenciales para el funcionamiento del sistema.

        1. **Vehiculo.kt:**

            En el desarrollo de VioDrive, una aplicación móvil enfocada en la vente de vehículos, la clase vehículo representa el modelo principal que almacena la información de cada automóvil registrado en el sistema. Este modelo permite manejar datos relevantes como la placa, marca, modelo, año, color, tarifa de alquiler y estado de disponibilidad. Además, cada vehículo incluye una imagen asociada para su visualización en la interfaz. Su correcta estructuración es clave para las operaciones de listado, filtrado y selección de vehículos dentro de la aplicación.
    2. **Controlador:**

        Esta carpeta contiene la lógica de control de la aplicación. Es el intermediario entre la vista (interfaz de usuario) y el modelo (datos). Aquí se gestionan las operaciones de base de datos relacionadas con usuarios y vehículos, implementando acciones como registro, verificación, inserción, actualización, eliminación y lectura de datos.

        1. **DBHelper.kt:**

            Dentro de la aplicación VioDrive, la clase DBHelper se encarga de gestionar toda la interacción con la base de datos local mediante SQLite. Permite registrar y verificar usuarios de forma segura aplicando hash SHA-256 a las contraseñas, reforzando así la privacidad. Asimismo, controla el CRUD de los vehículos, desde su inserción hasta la obtención de toda la lista de automóviles disponibles, lo que facilita su visualización en la interfaz. Esta clase representa la lógica de negocio central y forma parte del componente Controlador en la arquitectura MVC, sirviendo de puente entre los datos (modelo) y la presentación (vista).
    3. **Vista:**

        La carpeta View representa la interfaz de usuario (UI) de la aplicación. Aquí se ubican todas las clases, fragments, activities, layouts y componentes visuales que el usuario puede ver e interactuar con ellos. Su función principal es mostrar los datos que el modelo proporciona y recibir las acciones del usuario para luego enviarlas al controlador.

        1. **EdiciónVehiculoScreen.kt:**

            Permite editar los datos de un vehículo existente mostrando campos de texto para marca, modelo, año, color, costo por día y un checkbox para estado activo, con validaciones que aseguran

entradas válidas (solo letras para texto, año entre 1900 y 2100, costo positivo). Al guardar, actualiza el vehículo en la base de datos mediante dbHelper, muestra un mensaje Toast con el resultado y, si es exitoso, regresa a la pantalla anterior; todo esto en una interfaz con fondo degradado, scroll vertical y diseño centrado para facilitar la usabilidad.

2. **EliminarVehiculoScreen.kt:**
   Permite al usuario ingresar la placa de un vehículo para eliminarlo de la base de datos usando DBHelper, validando que la placa no esté vacía, mostrando mensajes Toast según el resultado (éxito, no encontrado o campo vacío) y navegando hacia atrás al eliminar correctamente, todo con un diseño sencillo que incluye un fondo degradado y scroll vertical para adaptarse a diferentes tamaños de pantalla.
3. **HomeScreen.kt:**
   Permite al usuario eliminar un vehículo de la base de datos ingresando su número de placa. Utiliza un campo de texto para capturar la placa, valida que no esté vacío y, al presionar el botón, invoca el método eliminarVehiculo() del controlador DBHelper. Según el resultado, muestra un mensaje mediante un Toast y, si la eliminación es exitosa, navega de regreso a la pantalla anterior utilizando NavController.
4. **InsertVehiculoScreen.kt:**
   Permite al usuario registrar un nuevo vehículo capturando datos como placa, marca, modelo, año, color, costo y estado activo. Valida los campos de entrada para asegurar datos correctos antes de crear un objeto Vehiculo y enviar la información al controlador DBHelper mediante el método insertarVehiculo(). Según el resultado, muestra un mensaje Toast y, si la inserción es exitosa, regresa a la pantalla anterior usando NavController, manteniendo la separación de responsabilidades conforme al patrón MVC.
5. **LoginScreen.kt:**
   Permite al usuario iniciar sesión ingresando su usuario y contraseña. Realiza validaciones detalladas en ambos campos para asegurar un formato correcto y seguro, luego verifica las credenciales en la base de datos local mediante DBHelper. Según el resultado, muestra mensajes Toast de éxito o error, y navega a la pantalla principal o al registro de usuario usando NavController, siguiendo el patrón MVC para mantener la lógica separada de la interfaz.
6. **RegisterScreen.kt:**
   Permite a nuevos usuarios registrarse ingresando un nombre de usuario y contraseña, valida ambos campos con reglas estrictas para asegurar formato y seguridad, guarda el usuario en una base de datos local mediante DBHelper, muestra mensajes Toast para

indicar éxito o error (como usuario ya existente) y permite navegar entre las pantallas de registro y login usando NavController.

4. **UI.Theme:**
   La carpeta ui.theme contiene la paleta de colores personalizada que define los tonos principales usados en la interfaz de usuario de la aplicación, asegurando consistencia visual mediante colores claros y oscuros.
   1. **Color.kt:**
      Define una serie de colores personalizados para la interfaz de usuario, especificando tonos claros y oscuros que se utilizan en el tema visual de la aplicación para mantener coherencia y estilo en el diseño.
5. **Main Activity:**
   Configura la navegación de la app usando Jetpack Compose Navigation, creando una instancia única de DBHelper para acceso a la base de datos y definiendo rutas para las pantallas de login, registro, inicio, inserción, edición y eliminación de vehículos; para la edición, recibe la placa del vehículo por argumento, busca el objeto correspondiente y muestra la pantalla de edición si existe, o regresa atrás si no, mientras que la eliminación usa una instancia de DBHelper basada en el contexto local, integrando así la gestión completa del flujo de la app con navegación declarativa y acceso centralizado a datos.

Carpeta Modelo

```kotlin
package uce.edu.ec.Models

import uce.edu.ec.R

data class Vehiculo(
    val placa: String,          // Identificador único del vehículo
    val marca: String,          // Marca fabricante
    val modelo: String,         // Modelo específico
    val anio: Int,              // Año de fabricación
    val color: String,          // Color del vehículo
    val costoPorDia: Double,    // Costo de alquiler diario
    val activo: Boolean,        // Estado de disponibilidad o uso
    val imagenRes: Int = R.drawable.a4, // Recurso de imagen predeterminado para el vehículo
)
```

Carpeta Controlador

```
package uce.edu.ec.Controller

import android.content.ContentValues
import android.content.Context
import android.database.sqlite.SQLiteDatabase
import android.database.sqlite.SQLiteOpenHelper
import uce.edu.ec.Models.Vehiculo
import java.security.MessageDigest

class DBHelper(context: Context) : SQLiteOpenHelper(context, "usuarios.db", null, 3) {

    override fun onCreate(db: SQLiteDatabase?) {
        // Crear tabla usuarios para almacenar credenciales
        db?.execSQL("""
            CREATE TABLE usuarios(
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                usuario TEXT,
                contrasenia TEXT
            )
        """.trimIndent())
```

```
        // Crear tabla vehiculos con sus atributos, placa es la clave primaria
        db?.execSQL("""
            CREATE TABLE vehiculos(
                placa TEXT PRIMARY KEY,
                marca TEXT,
                modelo TEXT,
                anio INTEGER,
                color TEXT,
                costoPorDia REAL,
                activo INTEGER,
                imagenRes INTEGER
            )
        """.trimIndent())
    }

    override fun onUpgrade(db: SQLiteDatabase?, oldVersion: Int, newVersion: Int) {
        // Actualización simple: eliminar tablas y recrearlas (pérdida de datos)
        db?.execSQL("DROP TABLE IF EXISTS usuarios")
        db?.execSQL("DROP TABLE IF EXISTS vehiculos")
        onCreate(db)
    }
```

```
    /**
     * Función privada para hashear contraseñas usando SHA-256
     * como medida de seguridad para almacenamiento seguro.
     */
    private fun hashPassword(password: String): String {
        val md = MessageDigest.getInstance("SHA-256")
        val bytes = md.digest(password.toByteArray())
        return bytes.joinToString("") { "%02x".format(it) }
    }

    /**
     * Registrar un nuevo usuario en la base de datos con la contraseña hasheada.
     * @return true si la inserción fue exitosa, false en caso contrario.
     */
    fun registrarUsuario(usuario: String, contrasenia: String): Boolean {
        val db = writableDatabase
        val hashedPassword = hashPassword(contrasenia)
        val values = ContentValues().apply {
            put("usuario", usuario)
            put("contrasenia", hashedPassword)
        }
        return db.insert("usuarios", null, values) != -1L
    }
```

```kotlin
    /**
     * Verificar existencia de usuario con contraseña correcta.
     * Se compara la contraseña hasheada para mayor seguridad.
     */
    fun verificarUsuario(usuario: String, contrasenia: String): Boolean {
        val db = readableDatabase
        val hashedPassword = hashPassword(contrasenia)
        val cursor = db.rawQuery(
            "SELECT * FROM usuarios WHERE usuario=? AND contrasenia=?",
            arrayOf(usuario, hashedPassword)
        )
        val exists = cursor.count > 0
        cursor.close()
        return exists
    }

    // --- Operaciones CRUD para Vehiculos ---

    /**
     * Insertar un nuevo vehículo en la base de datos.
     */
    fun insertarVehiculo(vehiculo: Vehiculo): Boolean {
        val db = writableDatabase
        val values = ContentValues().apply {
            put("placa", vehiculo.placa)
            put("marca", vehiculo.marca)
            put("modelo", vehiculo.modelo)
            put("anio", vehiculo.anio)
            put("color", vehiculo.color)
            put("costoPorDia", vehiculo.costoPorDia)
            put("activo", if (vehiculo.activo) 1 else 0)
            put("imagenRes", vehiculo.imagenRes)
        }
        return db.insert("vehiculos", null, values) != -1L
    }
```

```kotlin
/**
 * Actualizar la información de un vehículo existente.
 * La placa se usa para identificar el registro a modificar.
 */
fun actualizarVehiculo(vehiculo: Vehiculo): Boolean {
    val db = writableDatabase
    val values = ContentValues().apply {
        put("marca", vehiculo.marca)
        put("modelo", vehiculo.modelo)
        put("anio", vehiculo.anio)
        put("color", vehiculo.color)
        put("costoPorDia", vehiculo.costoPorDia)
        put("activo", if (vehiculo.activo) 1 else 0)
        put("imagenRes", vehiculo.imagenRes)
    }
    val updated = db.update("vehiculos", values, "placa=?", arrayOf(vehiculo.placa))
    return updated > 0
}

/**
 * Eliminar un vehículo de la base de datos dado su número de placa.
 */
fun eliminarVehiculo(placa: String): Boolean {
    val db = writableDatabase
    val deleted = db.delete("vehiculos", "placa=?", arrayOf(placa))
    return deleted > 0
}
```

```
/**
 * Obtener la lista completa de vehículos almacenados.
 * Mapea cada fila de la tabla a un objeto Vehiculo del modelo.
 */
fun obtenerVehiculos(): List<Vehiculo> {
    val db = readableDatabase
    val cursor = db.rawQuery("SELECT * FROM vehiculos", null)
    val lista = mutableListOf<Vehiculo>()
    if (cursor.moveToFirst()) {
        do {
            val vehiculo = Vehiculo(
                placa = cursor.getString(cursor.getColumnIndexOrThrow("placa")),
                marca = cursor.getString(cursor.getColumnIndexOrThrow("marca")),
                modelo = cursor.getString(cursor.getColumnIndexOrThrow("modelo")),
                anio = cursor.getInt(cursor.getColumnIndexOrThrow("anio")),
                color = cursor.getString(cursor.getColumnIndexOrThrow("color")),
                costoPorDia = cursor.getDouble(cursor.getColumnIndexOrThrow("costoPorDia")),
                activo = cursor.getInt(cursor.getColumnIndexOrThrow("activo")) == 1,
                imagenRes = cursor.getInt(cursor.getColumnIndexOrThrow("imagenRes"))
            )
            lista.add(vehiculo)
        } while (cursor.moveToNext())
    }
    cursor.close()
    return lista
}
```

```
@Composable
fun EdicionVehiculoScreen(navController: NavController, vehiculoEditar: Vehiculo, dbHelper: DBHelper) {
    val context = LocalContext.current  // Contexto para mostrar Toast

    // Estados para cada campo editable, inicializados con los valores del vehículo a editar
    var marca by remember { mutableStateOf(vehiculoEditar.marca) }
    var modelo by remember { mutableStateOf(vehiculoEditar.modelo) }
    var anio by remember { mutableStateOf(vehiculoEditar.anio.toString()) }
    var color by remember { mutableStateOf(vehiculoEditar.color) }
    var costoPorDia by remember { mutableStateOf(vehiculoEditar.costoPorDia.toString()) }
    var activo by remember { mutableStateOf(vehiculoEditar.activo) }

    // Variables para controlar errores de validación
    var marcaError by remember { mutableStateOf<String?>(null) }
    var modeloError by remember { mutableStateOf<String?>(null) }
    var anioError by remember { mutableStateOf<String?>(null) }
    var colorError by remember { mutableStateOf<String?>(null) }
    var costoError by remember { mutableStateOf<String?>(null) }

    // Colores y fondo con gradiente para la UI
    val rosa = Color(0xFFD7B8E8)
    val scrollState = rememberScrollState()
    val backgroundGradient = Brush.verticalGradient(colors = listOf(rosa, Color.White))

    Column(
        modifier = Modifier
            .fillMaxSize()
            .background(backgroundGradient)
            .padding(horizontal = 24.dp, vertical = 90.dp)
            .verticalScroll(scrollState),
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        Text(
            text = "Editar Vehículo - Placa: ${vehiculoEditar.placa}",
            style = MaterialTheme.typography.h6.copy(fontWeight =
androidx.compose.ui.text.font.FontWeight.Bold),
            modifier = Modifier.padding(bottom = 24.dp)
```

```
    // Campo Marca con validación y posible mensaje de error
    OutlinedTextField(
        value = marca,
        onValueChange = {
            marca = it
            marcaError = null  // Limpiar error cuando cambia el valor
        },
        label = { Text("Marca") },
        isError = marcaError != null,
        modifier = Modifier.fillMaxWidth(),
        singleLine = true
    )
    marcaError?.let { Text(it, color = Color.Red, style = MaterialTheme.typography.caption) }

    Spacer(modifier = Modifier.height(12.dp))

    // Campo Modelo
    OutlinedTextField(
        value = modelo,
        onValueChange = {
            modelo = it
            modeloError = null
        },
        label = { Text("Modelo") },
        isError = modeloError != null,
        modifier = Modifier.fillMaxWidth(),
        singleLine = true
    )
    modeloError?.let { Text(it, color = Color.Red, style = MaterialTheme.typography.caption) }

    Spacer(modifier = Modifier.height(12.dp))
```

Carpeta Vista

```
package uce.edu.ec.View

import android.widget.Toast
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.rememberScrollState
import androidx.compose.foundation.text.KeyboardOptions
import androidx.compose.foundation.verticalScroll
import androidx.compose.material.*
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Brush
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.text.input.KeyboardType
import androidx.compose.ui.unit.dp
import androidx.navigation.NavController
import uce.edu.ec.Controller.DBHelper
import uce.edu.ec.Models.Vehiculo
```

```
package uce.edu.ec.View

import android.widget.Toast
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.rememberScrollState
import androidx.compose.foundation.verticalScroll
import androidx.compose.material.*
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Brush
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.unit.dp
import androidx.navigation.NavController
import uce.edu.ec.Controller.DBHelper

// Pantalla para eliminar un vehículo de la base de datos
@Composable
fun EliminarVehiculoScreen(navController: NavController, dbHelper: DBHelper) {
    val context = LocalContext.current
    var placa by remember { mutableStateOf("") } // Estado para capturar la placa
    val scrollState = rememberScrollState()

    Box(
        modifier = Modifier
            .fillMaxSize()
            .background( // Fondo degradado
                Brush.verticalGradient(
                    colors = listOf(Color(0xFFD7B8E8), Color.White)
                )
            )
```

```
// Campo Costo por día, permite números y punto decimal
OutlinedTextField(
    value = costoPorDia,
    onValueChange = {
        costoPorDia = it.filter { c -> c.isDigit() || c == '.' }
        costoError = null
    },
    label = { Text("Costo por día") },
    isError = costoError != null,
    keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Number),
    modifier = Modifier.fillMaxWidth(),
    singleLine = true
)
costoError?.let { Text(it, color = Color.Red, style = MaterialTheme.typography.caption) }

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

// Checkbox para indicar si el vehículo está activo
Row(verticalAlignment = Alignment.CenterVertically) {
    Checkbox(
        checked = activo,
        onCheckedChange = { activo = it },
        colors = CheckboxDefaults.colors(checkedColor = MaterialTheme.colors.primary)
    )
    Spacer(modifier = Modifier.width(8.dp))
    Text("Activo")
}

Spacer(modifier = Modifier.height(32.dp))
```

```
            )

            Spacer(modifier = Modifier.height(32.dp))

            // Botón para ejecutar la eliminación
            Button(
                onClick = {
                    if (placa.isNotBlank()) {
                        val exito = dbHelper.eliminarVehiculo(placa) // Llama al controlador
                        if (exito) {
                            Toast.makeText(context, "Vehículo eliminado", Toast.LENGTH_SHORT).show()
                            navController.popBackStack() // Regresa a pantalla anterior
                        } else {
                            Toast.makeText(context, "No se encontró el vehículo",
Toast.LENGTH_SHORT).show()
                        }
                    } else {
                        Toast.makeText(context, "Ingrese la placa", Toast.LENGTH_SHORT).show()
                    }
                },
                modifier = Modifier
                    .fillMaxWidth()
                    .height(50.dp)
            ) {
                Text("Eliminar Vehículo")
            }
        }
    }
}
```

```
                    else -> {
                        // Construimos el objeto Vehiculo actualizado
                        val vehiculoActualizado = Vehiculo(
                            placa = vehiculoEditar.placa,
                            marca = marca,
                            modelo = modelo,
                            anio = anio.toInt(),
                            color = color,
                            costoPorDia = costoPorDia.toDouble(),
                            activo = activo,
                            imagenRes = vehiculoEditar.imagenRes
                        )

                        // Intentamos actualizar la base de datos con el helper
                        val exito = dbHelper.actualizarVehiculo(vehiculoActualizado)

                        if (exito) {
                            Toast.makeText(context, "Vehículo actualizado correctamente",
Toast.LENGTH_SHORT).show()
                            navController.popBackStack()  // Volver atrás en la navegación
                        } else {
                            Toast.makeText(context, "Error al actualizar el vehículo",
Toast.LENGTH_SHORT).show()
                        }
                    }
                }
            },
            modifier = Modifier
                .fillMaxWidth()
                .height(50.dp)
        ) {
            Text(text = "Guardar Cambios")
        }
    }
}
```

```
// Lista de tarjetas por cada vehículo
LazyColumn(
    modifier = Modifier
        .fillMaxSize()
        .background(Color(0xFFD7B8E8))
        .padding(16.dp)
) {
    item {
        Text(
            "Lista de Vehículos",
            style = MaterialTheme.typography.h5.copy(
                fontWeight = FontWeight.Bold,
                color = MaterialTheme.colors.primary
            )
        )
        Spacer(modifier = Modifier.height(16.dp))
    }

    // Renderiza cada vehículo en una tarjeta
    items(listaVehiculos) { vehiculo ->
        Card(
            modifier = Modifier
                .fillMaxWidth()
                .padding(vertical = 8.dp),
            shape = RoundedCornerShape(16.dp),
            elevation = 6.dp,
        ) {
            Column(modifier = Modifier.padding(16.dp)) {
                Image(
                    painter = painterResource(id = vehiculo.imagenRes),
                    contentDescription = "Imagen de ${vehiculo.marca} ${vehiculo.modelo}",
                    modifier = Modifier
                        .background(
```

```
        // Botón para guardar cambios, valida cada campo y actualiza si todo es correcto
        Button(
            onClick = {
                val regexLetrasEspacios = Regex("^[a-zA-ZáéíóúÁÉÍÓÚñÑ\\s]+$")

                when {
                    marca.isBlank() || !regexLetrasEspacios.matches(marca) -> {
                        marcaError = "Marca inválida. Solo letras y espacios."
                    }

                    modelo.isBlank() || !regexLetrasEspacios.matches(modelo) -> {
                        modeloError = "Modelo inválido. Solo letras y espacios."
                    }

                    anio.isBlank() || anio.length != 4 || anio.toIntOrNull() == null || anio.toInt() <
1900 || anio.toInt() > 2100 -> {
                        anioError = "Año inválido. Debe tener 4 dígitos entre 1900 y 2100."
                    }

                    color.isBlank() || !regexLetrasEspacios.matches(color) -> {
                        colorError = "Color inválido. Solo letras y espacios."
                    }

                    costoPorDia.isBlank() || costoPorDia.toDoubleOrNull() == null ||
costoPorDia.toDouble() <= 0 -> {
                        costoError = "Costo inválido. Debe ser mayor que 0."
                    }
```

```
        }

        item {
            Spacer(modifier = Modifier.height(24.dp))

            // Botón para agregar nuevo vehículo
            Button(
                onClick = { navController.navigate("insertVehiculo") },
                modifier = Modifier
                    .fillMaxWidth()
                    .height(50.dp),
                shape = RoundedCornerShape(12.dp),
                colors = ButtonDefaults.buttonColors(backgroundColor = Color(0xFF62D968))
            ) {
                Text("Agregar Vehículo", color = Color.White, fontWeight = FontWeight.Bold, fontSize =
16.sp)
            }

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

            // Botón para ir a la pantalla de eliminación
            Button(
                onClick = { navController.navigate("eliminarVehiculo") },
                modifier = Modifier
                    .fillMaxWidth()
                    .height(50.dp),
                shape = RoundedCornerShape(12.dp),
                colors = ButtonDefaults.buttonColors(backgroundColor = Color(0xFFD32F2F))
            ) {
                Text("Eliminar Vehículo", color = Color.White, fontWeight = FontWeight.Bold, fontSize =
16.sp)
            }

            Spacer(modifier = Modifier.height(32.dp))
```

```
                        )
                        .fillMaxWidth()
                        .height(180.dp)
                        .clip(RoundedCornerShape(12.dp))
                )

                Spacer(modifier = Modifier.height(12.dp))

                // Datos básicos del vehículo
                VehicleInfoRow(label = "Placa", value = vehiculo.placa)
                VehicleInfoRow(label = "Marca", value = vehiculo.marca)
                VehicleInfoRow(label = "Modelo", value = vehiculo.modelo)
                VehicleInfoRow(label = "Año", value = vehiculo.anio.toString())
                VehicleInfoRow(label = "Color", value = vehiculo.color)
                VehicleInfoRow(label = "Costo por dia", value = "\$${vehiculo.costoPorDia}")
                VehicleInfoRow(label = "Activo", value = if (vehiculo.activo) "Sí" else "No")

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

                // Botón para editar vehículo
                Button(
                    onClick = {
                        navController.navigate("editVehiculo/${vehiculo.placa}")
                    },
                    modifier = Modifier
                        .fillMaxWidth()
                        .height(48.dp),
                    shape = RoundedCornerShape(12.dp),
                    colors = ButtonDefaults.buttonColors(backgroundColor =
MaterialTheme.colors.primary)
                ) {
                    Text("Editar", color = Color.White, fontWeight = FontWeight.Bold)
                }
```

```
// Vehículos por defecto si la BD está vacía
val vehiculosPorDefecto = listOf(
    Vehiculo("ABC123", "Toyota", "Corolla", 2020, "Rojo", 45.0, true, R.drawable.a1),
    Vehiculo("XYZ789", "Chevrolet", "Spark", 2021, "Azul", 30.0, true, R.drawable.a2),
    Vehiculo("LMN456", "Kia", "Rio", 2019, "Negro", 35.0, false, R.drawable.a3)
)

// Al iniciar la pantalla: carga los vehículos desde la BD o inserta los por defecto
LaunchedEffect(Unit) {
    val vehiculosEnDb = dbHelper.obtenerVehiculos()
    if (vehiculosEnDb.isEmpty()) {
        vehiculosPorDefecto.forEach {
            dbHelper.insertarVehiculo(it)
        }
    }
    listaVehiculos = dbHelper.obtenerVehiculos()
}
```

```
            )
        )
        .padding(horizontal = 24.dp, vertical = 16.dp)
) {
    Column(
        modifier = Modifier
            .fillMaxSize()
            .verticalScroll(scrollState),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    ) {
package uce.edu.ec.View

import android.util.Log
import androidx.compose.foundation.Image
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material.*
import androidx.compose.runtime.*
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.graphics.Brush
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.navigation.NavController
import uce.edu.ec.Models.Vehiculo
import uce.edu.ec.R
import uce.edu.ec.Controller.DBHelper

// Pantalla principal que muestra la lista de vehículos registrados
@Composable
fun HomeScreen(navController: NavController, dbHelper: DBHelper) {
    var listaVehiculos by remember { mutableStateOf(emptyList<Vehiculo>()) }
```

```
package uce.edu.ec.View

import android.widget.Toast
import androidx.compose.foundation.Image
import androidx.compose.foundation.background
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.foundation.text.KeyboardOptions
import androidx.compose.material.*
import androidx.compose.runtime.*
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.text.input.KeyboardType
import androidx.compose.ui.text.input.PasswordVisualTransformation
import androidx.compose.ui.unit.dp
import androidx.navigation.NavController
import uce.edu.ec.R
import uce.edu.ec.Controller.DBHelper

@Composable
fun LoginScreen(navController: NavController) {
    val context = LocalContext.current // Contexto para mostrar mensajes
    val db = remember { DBHelper(context) } // Acceso a base de datos SQLite
```

```
            // Botón para cerrar sesión
            OutlinedButton(
                onClick = {
                    navController.navigate("login") {
                        popUpTo("home") { inclusive = true }
                    }
                },
                modifier = Modifier.fillMaxWidth(),
                shape = RoundedCornerShape(12.dp)
            ) {
                Text("Cerrar Sesión", color = MaterialTheme.colors.primary, fontWeight =
FontWeight.Bold)
            }

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

// Fila con etiqueta y valor para cada dato del vehículo
@Composable
fun VehicleInfoRow(label: String, value: String) {
    Row(
        modifier = Modifier
            .fillMaxWidth()
            .padding(vertical = 2.dp),
        horizontalArrangement = Arrangement.SpaceBetween
    ) {
        Text(label, fontWeight = FontWeight.Medium, color = Color.Gray)
        Text(value, fontWeight = FontWeight.SemiBold)
    }
}
```

```
    // Validaciones del campo contraseña
    if (contrasenia.isBlank()) {
        errorContrasenia = "La contraseña no puede estar vacía"
    } else if (contrasenia.length < 6) {
        errorContrasenia = "Mínimo 6 caracteres"
    } else if (!contrasenia.any { it.isDigit() } || !contrasenia.any { it.isLetter() }) {
        errorContrasenia = "Debe contener letras y números"
    } else if (contrasenia.first().isLetterOrDigit().not()) {
        errorContrasenia = "No puede empezar con símbolo"
    } else if (contrasenia.contains(" ")) {
        errorContrasenia = "No puede contener espacios"
    }

    return errorUsuario == null && errorContrasenia == null
}

// Diseño principal de la pantalla
Box(
    modifier = Modifier.fillMaxSize()
) {
    // Imagen de fondo
    Image(
        painter = painterResource(id = R.drawable.fondo),
        contentDescription = "Fondo",
        contentScale = androidx.compose.ui.layout.ContentScale.Crop,
        modifier = Modifier.matchParentSize()
    )

    // Capa blanca semitransparente sobre la imagen para mejorar la legibilidad
    Box(
        modifier = Modifier
            .matchParentSize()
            .background(Color.White.copy(alpha = 0.8f))
```

```
// Defino colores personalizados
val amarillo = Color(0xFFD7B8E8)
val morado = Color(0xFF5635DC)

// Gradiente de fondo para la pantalla
val backgroundGradient = Brush.verticalGradient(
    colors = listOf(amarillo, Color.White)
)

// Función que valida nombre de usuario y contraseña
fun validarCredenciales(usuario: String, contrasenia: String): Boolean {
    errorUsuario = null
    errorContrasenia = null

    // Validaciones para el usuario
    if (usuario.isBlank()) {
        errorUsuario = "El usuario no puede estar vacío"
    } else if (usuario.startsWith("-")) {
        errorUsuario = "No puede comenzar con '-'"
    } else if (!usuario.matches(Regex("^[a-zA-Z0-9_]{4,16}$"))) {
        errorUsuario = "Solo letras, números o '_' (4-16 caracteres)"
    } else if (usuario.all { it.isDigit() } || usuario.all { it.isLetter() }) {
        errorUsuario = "Debe contener letras y números"
    }

    // Validaciones para la contraseña
    if (contrasenia.isBlank()) {
        errorContrasenia = "La contraseña no puede estar vacía"
    } else if (contrasenia.length < 6) {
        errorContrasenia = "Mínimo 6 caracteres"
    } else if (!contrasenia.any { it.isDigit() } || !contrasenia.any { it.isLetter() }) {
        errorContrasenia = "Debe contener letras y números"
    } else if (contrasenia.first().isLetterOrDigit().not()) {
```

```
// Estados que almacenan el usuario y la contraseña ingresados
var usuario by remember { mutableStateOf("") }
var contrasenia by remember { mutableStateOf("") }

// Colores personalizados para la interfaz
val morado = Color(0xFF5635DC)
val rosa = Color(0xFFDA00FF)

// Estados para mostrar mensajes de error en los campos
var errorUsuario by remember { mutableStateOf<String?>(null) }
var errorContrasenia by remember { mutableStateOf<String?>(null) }

// Función para validar el formato de usuario y contraseña
fun validarCredenciales(usuario: String, contrasenia: String): Boolean {
    errorUsuario = null
    errorContrasenia = null

    // Validaciones del campo usuario
    if (usuario.isBlank()) {
        errorUsuario = "El usuario no puede estar vacío"
    } else if (usuario.startsWith("-")) {
        errorUsuario = "No puede comenzar con '-'"
    } else if (!usuario.matches(Regex("^[a-zA-Z0-9_]{4,16}$"))) {
        errorUsuario = "Solo letras, números o '_' (4-16 caracteres)"
    } else if (usuario.all { it.isDigit() } || usuario.all { it.isLetter() }) {
        errorUsuario = "Debe contener letras y números"
    }

    // Validaciones del campo contraseña
    if (contrasenia.isBlank()) {
        errorContrasenia = "La contraseña no puede estar vacía"
    } else if (contrasenia.length < 6) {
        errorContrasenia = "Mínimo 6 caracteres"
```

```
// Campo de texto para la contraseña
OutlinedTextField(
    value = contrasenia,
    onValueChange = { contrasenia = it },
    label = { Text("Contraseña") },
    singleLine = true,
    visualTransformation = PasswordVisualTransformation(),
    keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Password),
    modifier = Modifier.fillMaxWidth(),
    isError = errorContrasenia != null,
    colors = TextFieldDefaults.outlinedTextFieldColors(
        focusedBorderColor = morado,
        unfocusedBorderColor = morado.copy(alpha = 0.5f),
        focusedLabelColor = morado,
        errorBorderColor = Color.Red
    ),
    shape = RoundedCornerShape(12.dp)
)

// Mensaje de error si la contraseña no es válida
errorContrasenia?.let {
    Text(
        text = it,
        color = Color.Red,
        style = MaterialTheme.typography.caption,
        modifier = Modifier.align(Alignment.Start).padding(top = 4.dp)
    )
}

Spacer(modifier = Modifier.height(24.dp))

// Botón para registrar al usuario si todo está validado
Button(
```

```
// Defino colores personalizados
val amarillo = Color(0xFFD7B8E8)
val morado = Color(0xFF5635DC)

// Gradiente de fondo para la pantalla
val backgroundGradient = Brush.verticalGradient(
    colors = listOf(amarillo, Color.White)
)

// Función que valida nombre de usuario y contraseña
fun validarCredenciales(usuario: String, contrasenia: String): Boolean {
    errorUsuario = null
    errorContrasenia = null

    // Validaciones para el usuario
    if (usuario.isBlank()) {
        errorUsuario = "El usuario no puede estar vacío"
    } else if (usuario.startsWith("-")) {
        errorUsuario = "No puede comenzar con '-'"
    } else if (!usuario.matches(Regex("^[a-zA-Z0-9_]{4,16}$"))) {
        errorUsuario = "Solo letras, números o '_' (4-16 caracteres)"
    } else if (usuario.all { it.isDigit() } || usuario.all { it.isLetter() }) {
        errorUsuario = "Debe contener letras y números"
    }

    // Validaciones para la contraseña
    if (contrasenia.isBlank()) {
        errorContrasenia = "La contraseña no puede estar vacía"
    } else if (contrasenia.length < 6) {
        errorContrasenia = "Mínimo 6 caracteres"
    } else if (!contrasenia.any { it.isDigit() } || !contrasenia.any { it.isLetter() }) {
        errorContrasenia = "Debe contener letras y números"
    } else if (contrasenia.first().isLetterOrDigit().not()) {
```

```
            // Mostrar error en campo contraseña
            if (errorContrasenia != null) {
                Text(
                    text = errorContrasenia!!,
                    color = Color.Red,
                    style = MaterialTheme.typography.body2,
                    modifier = Modifier
                        .align(Alignment.Start)
                        .padding(start = 8.dp, top = 4.dp)
                )
            }

            Spacer(modifier = Modifier.height(32.dp))

            // Botón para iniciar sesión
            Button(
                onClick = {
                    if (validarCredenciales(usuario, contrasenia)) {
                        if (db.verificarUsuario(usuario, contrasenia)) {
                            Toast.makeText(context, "Bienvenido $usuario", Toast.LENGTH_SHORT).show()
                            navController.navigate("home") {
                                popUpTo("login") { inclusive = true } // Limpia la pila de navegación
                            }
                        } else {
                            Toast.makeText(context, "Usuario o contraseña incorrectos",
Toast.LENGTH_SHORT).show()
                        }
                    }
                },
                modifier = Modifier
                    .fillMaxWidth()
                    .height(50.dp),
                shape = RoundedCornerShape(12.dp),
```

```
    // Campo de texto para el usuario
    OutlinedTextField(
        value = usuario,
        onValueChange = { usuario = it },
        label = { Text("Nombre de usuario") },
        singleLine = true,
        modifier = Modifier.fillMaxWidth(),
        isError = errorUsuario != null,
        colors = TextFieldDefaults.outlinedTextFieldColors(
            focusedBorderColor = morado,
            unfocusedBorderColor = morado.copy(alpha = 0.5f),
            focusedLabelColor = morado,
            errorBorderColor = Color.Red
        ),
        shape = RoundedCornerShape(12.dp)
    )

    // Mensaje de error si el nombre de usuario no es válido
    errorUsuario?.let {
        Text(
            text = it,
            color = Color.Red,
            style = MaterialTheme.typography.caption,
            modifier = Modifier.align(Alignment.Start).padding(top = 4.dp)
        )
    }

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

    // Campo de texto para la contraseña
    OutlinedTextField(
        value = contrasenia,
```

```
package uce.edu.ec.View

import android.widget.Toast
import androidx.compose.foundation.background
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.foundation.text.KeyboardOptions
import androidx.compose.material.*
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Brush
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.text.input.KeyboardType
import androidx.compose.ui.text.input.PasswordVisualTransformation
import androidx.compose.ui.unit.dp
import androidx.navigation.NavController
import uce.edu.ec.Controller.DBHelper

@Composable
fun RegisterScreen(navController: NavController) {
    val context = LocalContext.current
    val db = remember { DBHelper(context) }

    // Estados para guardar el input del usuario y errores
    var usuario by remember { mutableStateOf("") }
    var contrasenia by remember { mutableStateOf("") }
    var errorUsuario by remember { mutableStateOf<String?>(null) }
    var errorContrasenia by remember { mutableStateOf<String?>(null) }
```

```
// Defino las rutas de navegación con sus respectivas pantallas
NavHost(navController = navController, startDestination = "login") {

    // Pantalla de inicio de sesión
    composable("login") { LoginScreen(navController) }

    // Pantalla de registro
    composable("register") { RegisterScreen(navController) }

    // Pantalla principal (home), le paso el DBHelper
    composable("home") { HomeScreen(navController, dbHelper) }

    // Pantalla para insertar vehículos
    composable("insertVehiculo") { InsertVehiculoScreen(navController, dbHelper) }

    // Pantalla para editar vehículo, recibe como parámetro la placa
    composable(
        route = "editVehiculo/{placa}",
        arguments = listOf(navArgument("placa") { type = NavType.StringType })
    ) { backStackEntry ->
        // Recupero la placa enviada por parámetro
        val placa = backStackEntry.arguments?.getString("placa") ?: ""

        // Busco el vehículo con esa placa en la base de datos
        val vehiculo: Vehiculo? = dbHelper.obtenerVehiculos().find { it.placa == placa }

        // Si existe, muestro la pantalla de edición, si no, regreso atrás
        if (vehiculo != null) {
            EdicionVehiculoScreen(navController, vehiculo, dbHelper)
        } else {
            navController.popBackStack()
        }
    }
```

```
// Contenido del formulario
Column(
    modifier = Modifier
        .fillMaxSize()
        .padding(24.dp),
    horizontalAlignment = Alignment.CenterHorizontally,
    verticalArrangement = Arrangement.Top
) {
    Spacer(modifier = Modifier.height(40.dp))

    // Logo de la app
    Image(
        painter = painterResource(id = R.drawable.a5),
        contentDescription = "Logo Autos",
        modifier = Modifier.size(350.dp)
    )

    Spacer(modifier = Modifier.height(40.dp))

    // Campo de texto para el usuario
    OutlinedTextField(
        value = usuario,
        onValueChange = { usuario = it },
        label = { Text("Usuario") },
        singleLine = true,
        modifier = Modifier.fillMaxWidth(),
        colors = TextFieldDefaults.outlinedTextFieldColors(
            focusedBorderColor = rosa,
            unfocusedBorderColor = morado,
            focusedLabelColor = rosa
        ),
        shape = RoundedCornerShape(12.dp),
        isError = errorUsuario != null
```

Main Activity

```
package uce.edu.ec

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.ui.platform.LocalContext
import androidx.navigation.NavType
import androidx.navigation.compose.NavHost
import androidx.navigation.compose.composable
import androidx.navigation.compose.rememberNavController
import androidx.navigation.navArgument
import uce.edu.ec.Controller.DBHelper
import uce.edu.ec.Models.Vehiculo
import uce.edu.ec.View.*

// Esta clase es el punto de entrada de la aplicación.
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        // Creo una sola instancia de DBHelper para reutilizarla en toda la app.
        val dbHelper = DBHelper(this)

        // Defino el contenido usando Jetpack Compose
        setContent {
            // Controlador de navegación para moverme entre pantallas
            val navController = rememberNavController()

            // Defino las rutas de navegación con sus respectivas pantallas
            NavHost(navController = navController, startDestination = "login") {
```

```
            Button(
                onClick = {
                    if (validarCredenciales(usuario, contrasenia)) {
                        if (db.verificarUsuario(usuario, contrasenia)) {
                            Toast.makeText(context, "Bienvenido $usuario", Toast.LENGTH_SHORT).show()
Toast.LENGTH_SHORT).show()
                            navController.navigate("login") {
                                popUpTo("register") { inclusive = true }
                            }
                        } else {
                            Toast.makeText(context, "Error: el usuario ya existe",
Toast.LENGTH_SHORT).show()
                        }
                    }
                },
                modifier = Modifier
                    .fillMaxWidth()
                    .height(50.dp),
                shape = RoundedCornerShape(12.dp),
                colors = ButtonDefaults.buttonColors(backgroundColor = morado)
            ) {
                Text("Registrarse", color = Color.White)
            }

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

            // Enlace para ir a la pantalla de login si ya tiene cuenta
            Text(
                text = "¿Ya tienes cuenta? Inicia sesión",
                color = morado,
                style = MaterialTheme.typography.body2,
                modifier = Modifier.clickable {
                    navController.navigate("login")
                }
            )
        }
    }
}
```

```
                    // Busco el vehículo con esa placa en la base de datos
                    val vehiculo: Vehiculo? = dbHelper.obtenerVehiculos().find { it.placa == placa }

                    // Si existe, muestro la pantalla de edición, si no, regreso atrás
                    if (vehiculo != null) {
                        EdicionVehiculoScreen(navController, vehiculo, dbHelper)
                    } else {
                        navController.popBackStack()
                    }
                }

                // Pantalla para eliminar vehículos. Aquí uso una nueva instancia de DBHelper por
contexto
                composable("eliminarVehiculo") {
                    EliminarVehiculoScreen(navController, DBHelper(LocalContext.current))
                }
            }
        }
    }
}
```

Color

```
package uce.edu.ec.ui.theme

import androidx.compose.ui.graphics.Color

// Colores principales en tonos claros (80%)
val Purple80 = Color(0xFFD0BCFF)        // Morado claro
val PurpleGrey80 = Color(0xFFCCC2DC)    // Morado grisáceo claro
val Pink80 = Color(0xFFEFB8C8)          // Rosa claro

// Colores principales en tonos oscuros (40%)
val Purple40 = Color(0xFF6650a4)        // Morado oscuro
val PurpleGrey40 = Color(0xFF625b71)    // Morado grisáceo oscuro
val Pink40 = Color(0xFF7D5260)          // Rosa oscuro
```

Aplicación Interactiva

Lista de Vehículos

Placa	**ABC123**
Marca	**Toyota**
Modelo	**Corolla**
Año	**2020**
Color	**Rojo**
Costo por día	**$45.0**
Activo	**Sí**

Editar

Placa	**LMN456**
Marca	**Kia**
Modelo	**Rio**
Año	**2019**
Color	**Negro**
Costo por día	**$35.0**
Activo	**No**

Editar

Agregar Vehículo

Eliminar Vehículo

Cerrar Sesión

Editar Vehículo – Placa: LMN456

Marca: Kia

Modelo: Rio

Año: 2019

Color: Negro

Costo por día: 35.0

☐ Activo

Guardar Cambios

Eliminar Vehículo

Placa del vehículo a eliminar

Eliminar Vehículo


Insertar Vehículo
Placa (ABC123)
Marca
Modelo
Año (2000-2050)
Color
Costo por día
Activo