

PG MULTIMEDIA Y DISPOSITIVOS MÓVILES - T1 - HI1

Sergio Camino Saiz
20/11/2022



ÍNDICE

Tecnologías utilizadas	2
Fase 1	3
Fase 2	4
Fase 3	5



Tecnologías utilizadas

Lenguaje : Kotlin.

IDE : Android Studio.

API de android : 33.

Versión SDK : 33.

Versión de Java : 11.

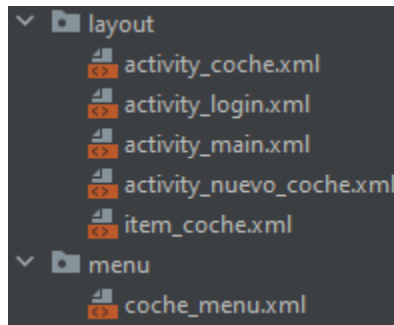
Versión de plugins de Gradle : 7.2.2.

Versión de gradle : 7.4.

“Curso” es el usuario y la contraseña preestablecidas.

Fase 1

Mi proyecto tiene 5 interfaces layout y 1 interfaz de menú.

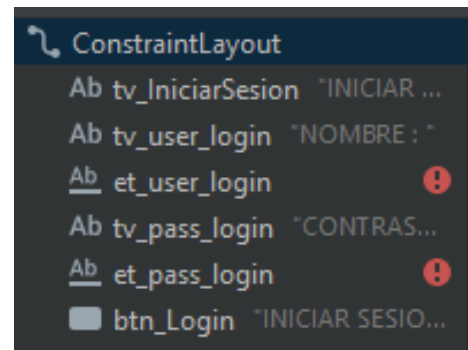


Como lenguaje voy a utilizar kotlin con la API 33. Como SDK voy a utilizar la versión 33 y como versión mínima la 21. Como IDE voy a utilizar Android Studio. La versión de Gradle es la versión 7.4 y para los plugins 7.2.2. La versión del JDK que voy a utilizar es la 11. Los componentes utilizados varían entre 'TextView', 'EditText', 'Button', 'Image', entre otras. Las librerías utilizadas son librerías de 'android', 'jetbrains', 'kotlin', entre otras.

appcompat:1.5.1	implementation
constraintlayout:2.1.4	implementation
core-ktx:1.9.0	implementation
espresso-core:3.5.0	androidTestImplementation
junit:1.1.4	androidTestImplementation
junit:4.13.2	testImplementation
material:1.7.0	implementation
navigation-fragment-ktx:2.5.3	implementation
navigation-ui-ktx:2.5.3	implementation
room-compiler:2.4.3	kapt
room-ktx:2.4.3	implementation
libs	implementation

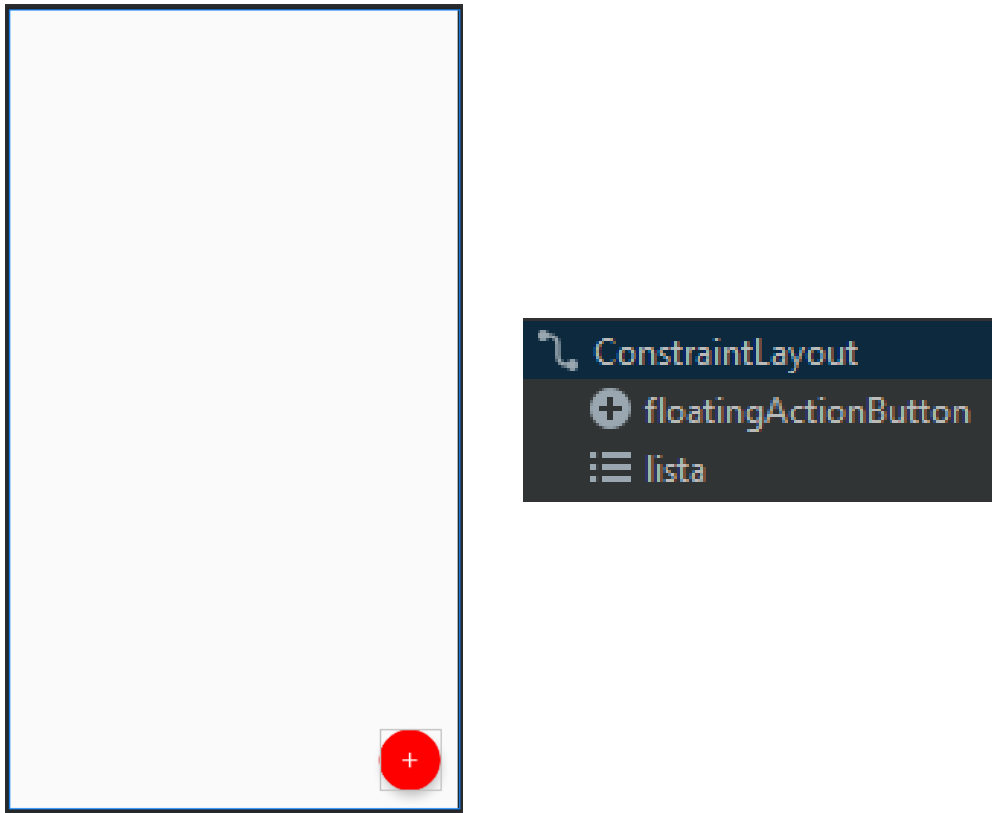
- Appcompat -> Nos permite acceder a APIs con diferentes versiones a la que estemos utilizando nosotros.
- Constraintlayout -> Nos permite crear diseños visuales de forma sencilla.
- Core-ktx -> Proporciona características necesarias para desarrollar el proyecto como 'valores predeterminados de parámetros', 'parámetros con nombres', 'lambdas', entre otras.
- Espresso-core -> Nos permite realizar test sobre las interfaces de nuestra aplicación.
- Junit -> Nos permite realizar test de nuestras clases Java.
- Material -> Nos ofrece una guía completa de diseño visual e interactivo para nuestras interfaces.
- Navigation -> Nos proporciona la opción de poder 'navegar' entre diferentes interfaces dentro de nuestra aplicación.
- Room -> Funciona como intermediaria entre la base de datos donde se guardaran nuestros elementos que guardemos en la aplicación y la parte "cliente" que sería nuestra aplicación.

- activity_login.xml :



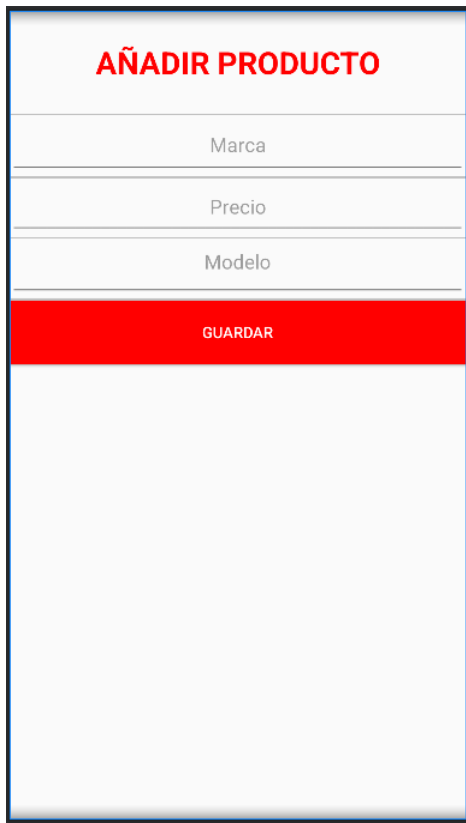
He utilizado un 'ConstraintLayout'. He utilizado TextView para utilizarlo como títulos y nombres (Iniciar sesión, Nombre, Contraseña). Para la respuesta del usuario he utilizado EditText tanto para el nombre de usuario como para la contraseña. La casilla de contraseña podría mejorarse y que fuera de tipo 'Password' para que cuando escribimos la contraseña salgan puntos y no se vea en texto plano. Para el botón de login he utilizado un 'Button' normal. Al pulsar el botón, si el usuario y la contraseña no son "curso" nos sale un mensaje emergente que nos notifica del error a la hora de iniciar sesión. Si iniciamos sesión correctamente saldrá otro mensaje emergente notificandonos de ello.

- Activity_main.xml :



He utilizado una view 'ConstraintLayout'. Para el botón de añadir coche he utilizado un botón flotante 'floatingActionButton' el cual al pulsarlo te redirige a la view de 'activity_nuevo_coche.xml'.

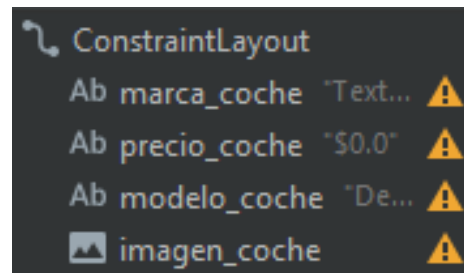
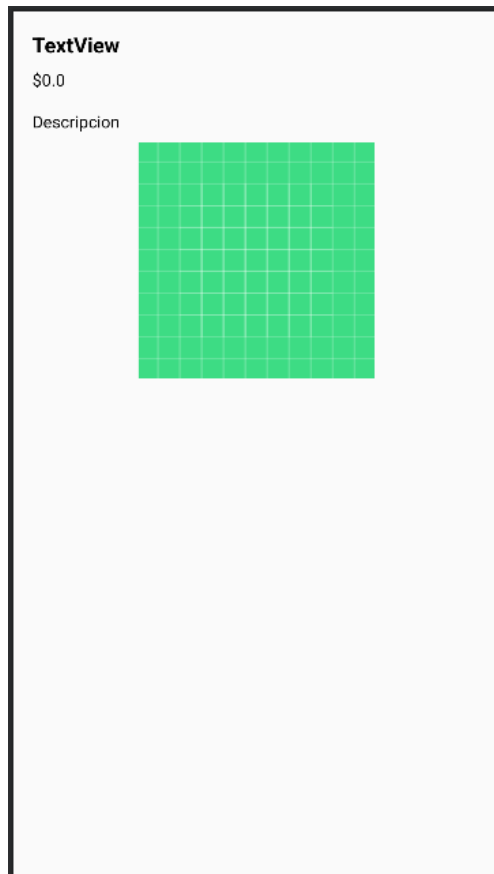
- Activity_nuevo_coche.xml :



```
LinearLayout (vertical)
  Ab tvAñadirProductos "@str...
  Ab et_marca
  Ab et_precio (Number (Deci...
  Ab et_modelo (Multiline Text)
  save_btn "@string/guard...
```

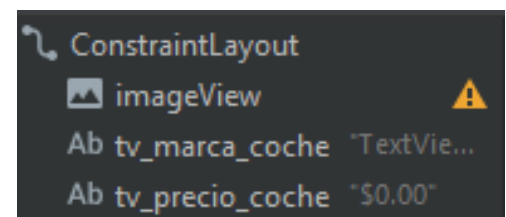
He utilizado una view 'ConstraintLayout'. He utilizado componentes TextView como títulos (Añadir producto). He utilizado EditText para que el usuario pueda introducir los datos que quiera los cuales en el código Java pasarán a ser una variable. El botón lo que hace es utilizar la librería 'Room' y así almacenar el producto creado. Al guardar el producto nos lleva a la View 'activity_coche.xml'.

- Activity_coche.xml :



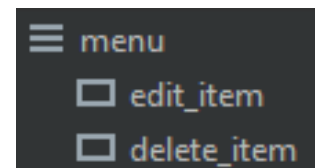
He utilizado una view 'ConstraintLayout'. He utilizado 'TextView' para pintar los datos que hemos almacenado como el objeto 'coche' en la view 'activity_nuevo_coche.xml'. El producto tiene una imagen por defecto que podemos modificar mediante una opción en el menú.

- Item_coche.xml :



He utilizado una view 'ConstraintLayout'. He utilizado 'TextView' para pintar la marca del coche y el precio. He utilizado 'ImageView' para pintar la imagen con la que guardamos nuestro coche.

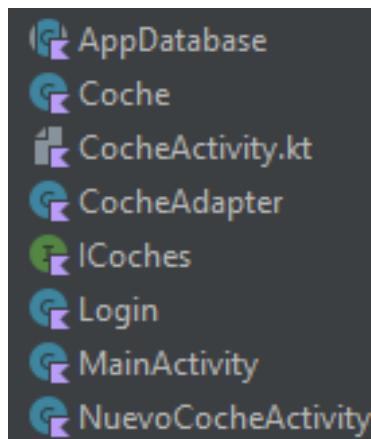
- Coche_menu.xml :



Mediante un menú he creado las opciones 'Update' y 'Delete' para el producto. La opción de update no actualiza el producto lo que hace es crear uno nuevo (cosa a mejorar). Delete si funciona correctamente.

Fase 2

Estas son las clases de mi proyecto. Tengo clases para las interfaces y tengo la clase ICoches la cual es una clase de tipo DAO (Data Access Object). La clase AppDatabase es la clase la cual utilizó para la “base de datos” donde se guardan los coches.



- Clase coche :

```
package com.hito

import androidx.room.Entity
import androidx.room.PrimaryKey
import java.io.Serializable

@Entity(tableName = "coches")
class Coche(
    var marca:String,
    var precio: Double,
    var modelo: String,
    var imagen: Int,
    @PrimaryKey(autoGenerate = true)
    var idCoche: Int = 0
) : Serializable
```

Se ha generado una clase a partir de la tabla ‘coches’ la cual especifica los atributos de los objetos y los tipos de datos que van a ser.

- Clase CocheAdapter :

Esta clase es la que utilizó en la interfaz de 'item_coche.xml', es decir, la interfaz donde muestro todos los coches con su imagen, marca y precio.

```
package com.hito

import android.content.Context
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.ArrayAdapter
import kotlinx.android.synthetic.main.item_coche.view.*

class CocheAdapter(private val mContext: Context, private val listaCoches: List<Coche>) : ArrayAdapter<Coche>(mContext, 0, listaCoches) {

    override fun getView(position: Int, convertView: View?, parent: ViewGroup): View {

        val layout = LayoutInflater.from(mContext).inflate(R.layout.item_coche, parent, false)

        val coche = listaCoches[position]

        layout.tv_marca_coche.text = coche.marca
        layout.tv_precio_coche.text = "$${coche.precio}"

        layout.imageView.setImageResource(coche.imagen)

        return layout
    }
}
```

- CocheActivity :

Es la clase de la interfaz 'activity_coche.xml' la cual nos muestra el coche creado con su imagen, marca, precio y modelo. No solo nos muestra el coche sino que también se encarga de interactuar con la base de datos. Esta clase interactúa con clases como Coche, NuevoCocheActivity, AppDatabase, entre otras.

```
package com.hito

import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.view.Menu
import android.view.MenuItem
import androidx.lifecycle.LiveData
import androidx.lifecycle.Observer
import kotlinx.android.synthetic.main.activity_nuevo_coche.*
import kotlinx.android.synthetic.main.activity_coche.*
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch

class CocheActivity : AppCompatActivity() {

    private lateinit var database: AppDatabase
    private lateinit var coche: Coche
    private lateinit var cocheLiveData: LiveData<Coche>

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_coche)

        database = AppDatabase.getDatabase(this)

        val idCoche = intent.getIntExtra("id", 0)

        cocheLiveData = database.coches().get(idCoche)

        cocheLiveData.observe(this, Observer {
            coche = it

            marca_coche.text = coche.marca
            precio_coche.text = "${coche.precio} €"
```

```

        modelo_coche.text = coche.modelo
        imagen_coche.setImageResource(coche.imagen)
    })
}

override fun onCreateOptionsMenu(menu: Menu?): Boolean {
    menuInflater.inflate(R.menu.coche_menu, menu)

    return super.onCreateOptionsMenu(menu)
}

override fun onOptionsItemSelected(item: MenuItem): Boolean
{
    when (item.itemId) {
        R.id.edit_item -> {
            val intent = Intent(this,
NuevoCocheActivity::class.java)
            intent.putExtra("coche", coche)
            startActivity(intent)
        }

        R.id.delete_item -> {
            cocheLiveData.removeObservers(this)

            CoroutineScope(Dispatchers.IO).launch {
                database.coches().delete(coche)
                this@CocheActivity.finish()
            }
        }
    }

    return super.onOptionsItemSelected(item)
}
}

```

- Clase ICoches :

Es la clase DAO (Data Access Object). Esta clase es la interfaz de métodos para la conexión entre la base de datos y el cliente. Tiene métodos CRUD para la creación de un nuevo coche.

```
package com.hito

import androidx.lifecycle.LiveData
import androidx.room.*

@Dao
interface ICoches {
    @Query("SELECT * FROM coches")
    fun getAll(): LiveData<List<Coche>>

    @Query("SELECT * FROM coches WHERE idCoche = :id")
    fun get(id: Int): LiveData<Coche>

    @Insert
    fun insertAll(vararg coches: Coche)

    @Update
    fun update(coche: Coche)

    @Delete
    fun delete(coche: Coche)
}
```


- Clase Login :

Esta clase es de la interfaz 'activity_login.xml' la cual es la primera View que carga cuando iniciamos la aplicación. Esta clase nos permite escribir nuestro usuario y contraseña ("Curso") y si coincide nos permite iniciar sesión y nos dirige a 'activity_main.xml' si no nos salta un mensaje emergente con un Toast que nos indica que hubo un error al inciar sesión.

```
package com.hito

import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.Toast
import kotlinx.android.synthetic.main.activity_login.*

class Login : AppCompatActivity() {
    var user:String = R.id.et_user_login.toString()
    var pass:String = R.id.et_pass_login.toString()
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_login)
        btn_Login.setOnClickListener() {
            if(et_user_login.text.toString()=="curso" &&
et_pass_login.text.toString()=="curso") {
                var toast = Toast.makeText(this, "Sesión
iniciada", Toast.LENGTH_LONG)
                toast.show()
                val intent = Intent(this,
MainActivity::class.java)
                startActivity(intent)
            }else{
                var toast = Toast.makeText(this, "Error al
iniciar sesión", Toast.LENGTH_LONG)
                toast.show()
            }
        }
    }
}
```

- Clase MainActivity :

Es la clase de java de la interfaz 'activity_main.xml' la cual una vez iniciamos sesión nos lleva a ella. En esta clase definimos el botón el cual nos permite crear un nuevo coche y después nos redirige a la view de 'Activity_nuevo_coche.xml'.

```
package com.hito

import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import androidx.lifecycle.Observer
import kotlinx.android.synthetic.main.activity_main.*

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        var listaCoches = emptyList<Coche>()

        val database = AppDatabase.getDatabase(this)

        database.coches().getAll().observe(this, Observer {
            listaCoches = it
            val adapter = CocheAdapter(this, listaCoches)
            lista.adapter = adapter
        })

        lista.setOnItemClickListener { parent, view, position, id ->
            val intent = Intent(this, CocheActivity::class.java)
            intent.putExtra("id", listaCoches[position].idCoche)
            startActivity(intent)
        }

        floatingActionButton.setOnClickListener {
            val intent = Intent(this, NuevoCocheActivity::class.java)
            startActivity(intent)
        }
    }
}
```

- Clase NuevoCocheActivity :

Es la clase de la interfaz 'activity_nuevo_coche.xml'. Esta clase nos permite crear un nuevo coche con su marca, precio y modelo. Una vez creamos el coche nos redirige a la View 'item_coche.xml' la cual muestra todos los objetos Coche que tengamos.

```
package com.hito

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import kotlinx.android.synthetic.main.activity_nuevo_coche.*
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch

class NuevoCocheActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_nuevo_coche)

        var idCoche: Int? = null

        if (intent.hasExtra("coches")) {
            val coche = intent.extras?.getSerializable("coches")
            as Coche

            et_marca.setText(coche.marca)
            et_precio.setText(coche.precio.toString())
            et_modelo.setText(coche.modelo)
            idCoche = coche.idCoche
        }

        val database = AppDatabase.getDatabase(this)

        save_btn.setOnClickListener {
            val marca = et_marca.text.toString()
            val precio = et_precio.text.toString().toDouble()
            val modelo = et_modelo.text.toString()

            val coche = Coche(marca, precio, modelo,
                R.drawable.ic_launcher_background)

            if (idCoche != null) {
                CoroutineScope(Dispatchers.IO).launch {
                    coche.idCoche = idCoche
                }
            }
        }
    }
}
```

```
database.coches().update(coche)

        this@NuevoCocheActivity.finish()
    }
} else {
    CoroutineScope(Dispatchers.IO).launch {
        database.coches().insertAll(coche)

        this@NuevoCocheActivity.finish()
    }
}
}
}
}
```

Fase 3

La aplicación ha sido ejecutada en un emulador con la API 33. Antes de lanzar la aplicación debe ser compilada y a continuación un ejemplo de que la compilación ha sido ejecutada con éxito.

```
Deprecated Gradle features were used in this build, making it incompatible with Gradle 8.0.

You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from your own scripts or plugins.

See https://docs.gradle.org/7.4/userguide/command\_line\_interface.html#sec:command\_line\_warnings

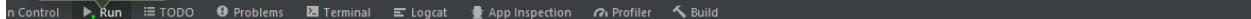
BUILD SUCCESSFUL in 4s
38 actionable tasks: 2 executed, 36 up-to-date

Build Analyzer results available
```

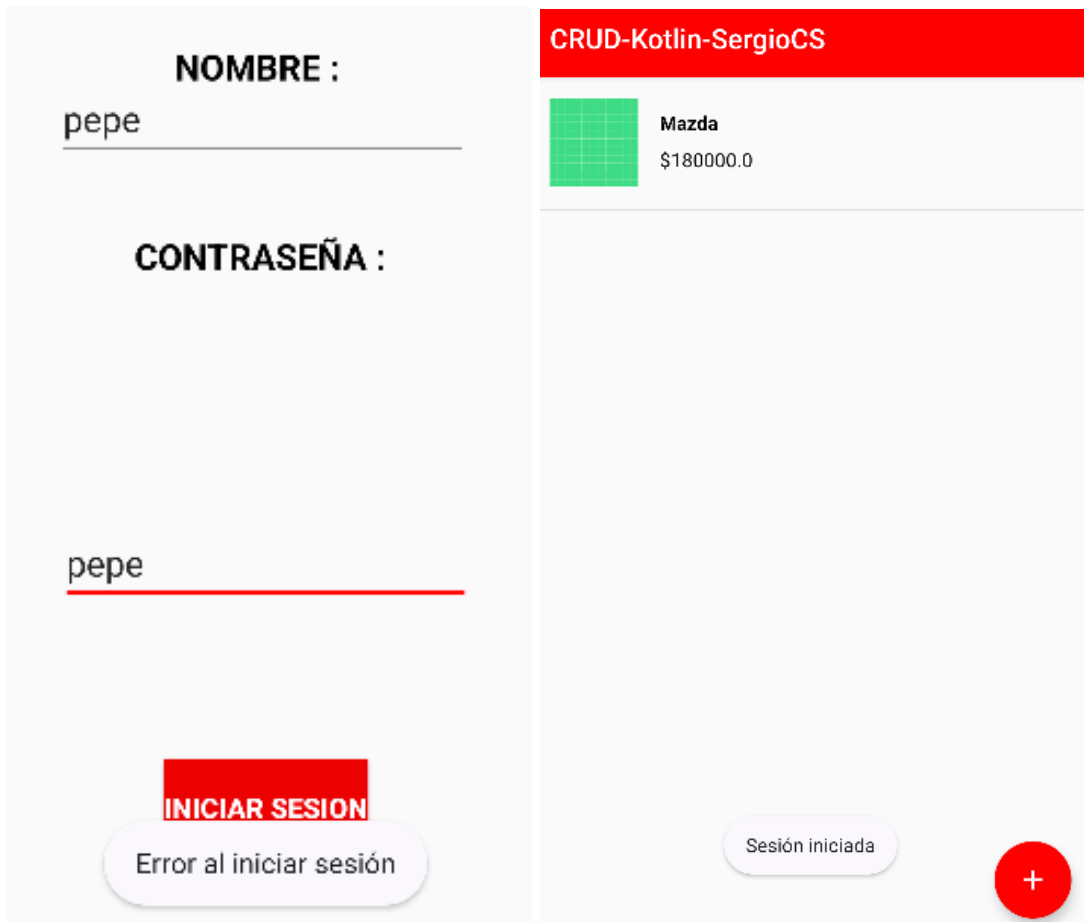
A continuación, se instalará la aplicación en el emulador y este es un ejemplo de que se instala correctamente.

```
11/25 09:47:31: Launching 'app' on Pixel XL API 33.
Install successfully finished in 476 ms.
$ adb shell am start -n "com.hito/com.hito.Login" -a android.intent.action.MAIN -c android.intent.category.LAUNCHER
Connected to process 23090 on device 'Pixel_XL_API_33 [emulator-5554]'.

Launch succeeded
```



Cuando se instala la aplicación y se muestra en nuestro emulador, la primera vista es el login como he dicho anteriormente. Si escribimos el nombre y contraseña incorrectos nos sale un mensaje emergente de error y si iniciamos sesión con los datos correctos nos saldrá un mensaje emergente igual notificandonos del estado de inicio de sesión.



Si pinchamos en el botón de más que vemos en rojo en la parte inferior derecha nos permitirá crear un coche. Debemos escribir cifras numéricas en el precio si no nos permitirá escribir en esa caja.

CRUD-Kotlin-SergioCS

AÑADIR PRODUCTO

Ferrari

312

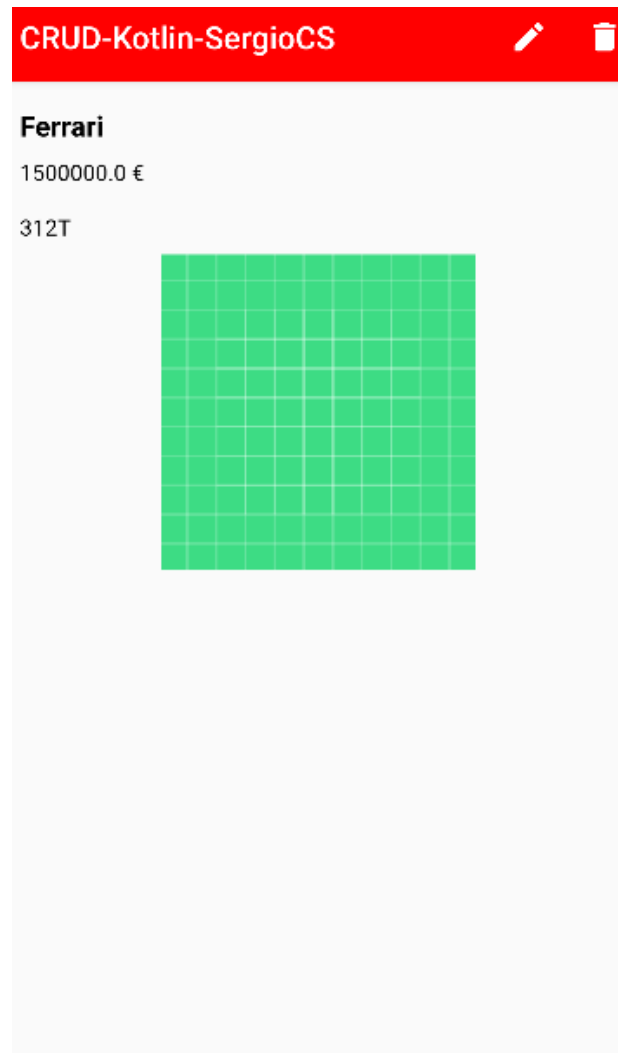
1500000

GUARDAR

Una vez lo creamos nos redirige a la View donde nos muestra todos nuestros coches.

CRUD-Kotlin-SergioCS	
	Mazda \$180000.0
	Ferrari \$1500000.0

Si pinchamos encima de cualquier coche creado podemos ver que tenemos la opción de actualizar los datos o borrar el coche.





PROBLEMAS O MEJORAS

La primera mejora sería cambiar la caja de texto plano de la contraseña en el Login por una caja de tipo Password la cual no muestre la contraseña visualmente en texto plano.

Un fallo es que cuando intentamos actualizar los datos de un producto nos dirige a la View de NuevoCocheActivity y nos hace crear un nuevo coche en vez de actualizar el que ya teníamos.

El diseño visual es bastante mejorable.

En tema de rendimiento la aplicación mejora considerablemente su rendimiento cuando tenemos un móvil con más de 2048 mb de RAM y más de 6 GB de almacenamiento.

En la interfaz de Login se puede mejorar bastante el diseño visual como por ejemplo añadiendo el logo de la aplicación en la parte superior.

Una mejora podría ser tener la base de datos bien gestionada en una nube y no tenerla en la memoria del dispositivo.

Bibliografía / Webgrafía

	Enlace de la página web
Kotlin	enlace
Android	enlace
Login example	enlace
Database	enlace 1 enlace 2 enlace 3 enlace 4 enlace 5
CRUD	enlace 1 enlace 2 enlace 3 enlace 4
Ejemplo de proyectos	enlace 1 enlace 2