Notas Droid

* Repositorio GitHub
* Video Youtube

* Pinchar para abrir el enlace

Florentina Santos Bueno

Indice

1. Introducción

¡Notas Droid es la única aplicación de estudio que necesitas! Sigue tu progreso académico con estadísticas completas. Lleva un registro de tus clases y añade eventos a tu horario semanal con facilidad.

1.1. Descripción

Notas Droid está diseñada para llevar un registro diario del progreso académico.

Podrás administrar tus asignaturas y gestionar tus tareas, tanto online como offline.

1.2. Características

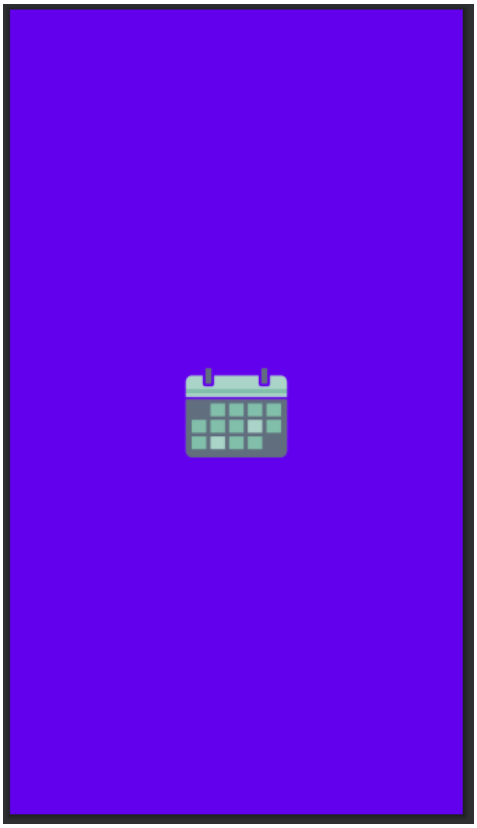
La aplicación se compone de:

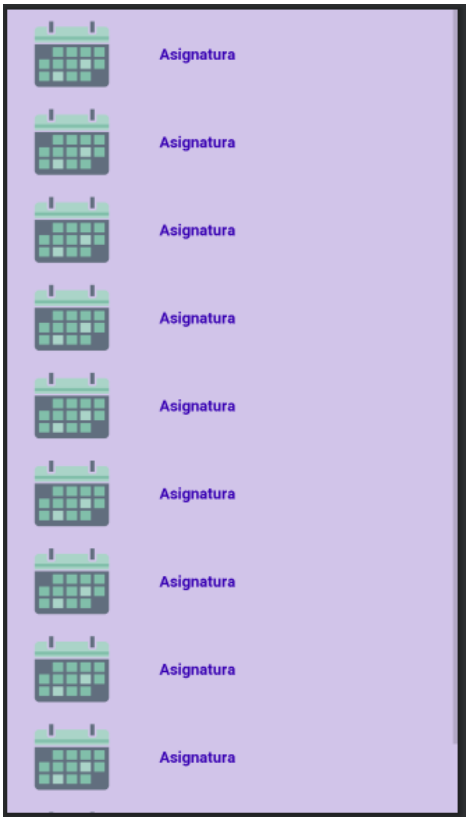
- **Pantalla de Presentación** (Splash Screen)
- **Pantalla de Login** en la que debemos identificarnos para acceder.
    - Si es la primera vez que entramos, debemos registrarnos en el sistema introduciendo los siguientes datos:
        - Nombre
        - Email
        - Contraseña (cifrada)
        - Fotografía (desde cámara o galería)
        - Ciclo que está cursando: DAM, DAW o ASIR
        - Curso: 1º ó 2º.
    - Si ya estamos registrados, entraremos poniendo únicamente los siguientes datos:
        - Email
        - Contraseña
    - La sesión se mantendrá abierta hasta que pulsemos el botón "Cerrar Sesión".
- **Pantalla Principal.**
    - Menú de opciones:
        - **Inicio**
        - **Mi Matrícula** → donde aparecerá una lista con los módulos en los que estamos matriculados. Organizada por ciclo, curso y nombre del módulo y una imagen.
        - **Mi Expediente** → aparecerán los datos del usuario logueado.
        - **Ajustes** → donde podremos modificar nuestros datos de usuario.
        - **Cerrar Sesión** → en la cual cerraremos sesión y volverá a la página de Login.

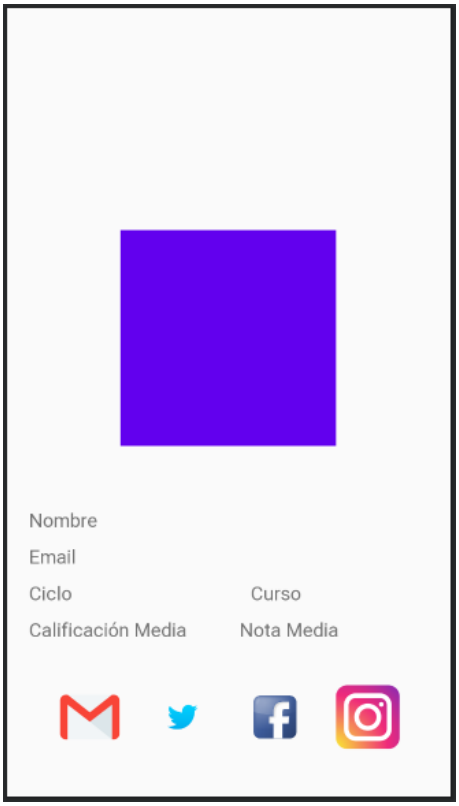
2. Documentación Técnica

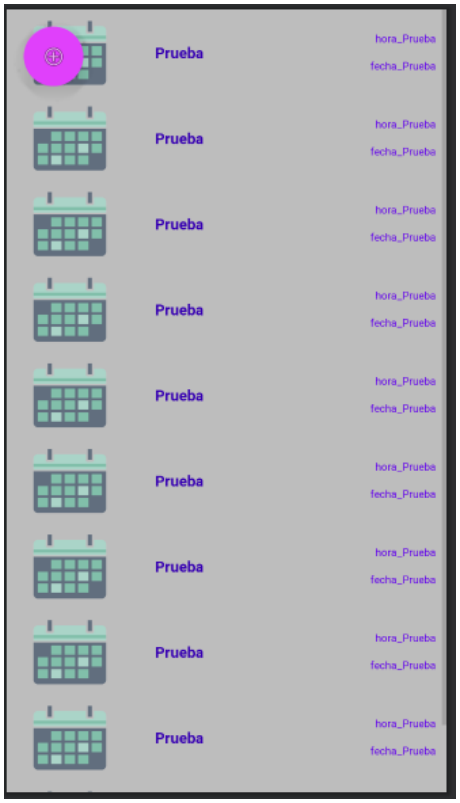
2.1. Documentación del Diseño

Primer Prototipo – (No pude terminarlo bien debido a que se me acabó el plazo de la prueba gratuita)









2.2. Documentación sobre el Código

2.2.1. Base de Datos

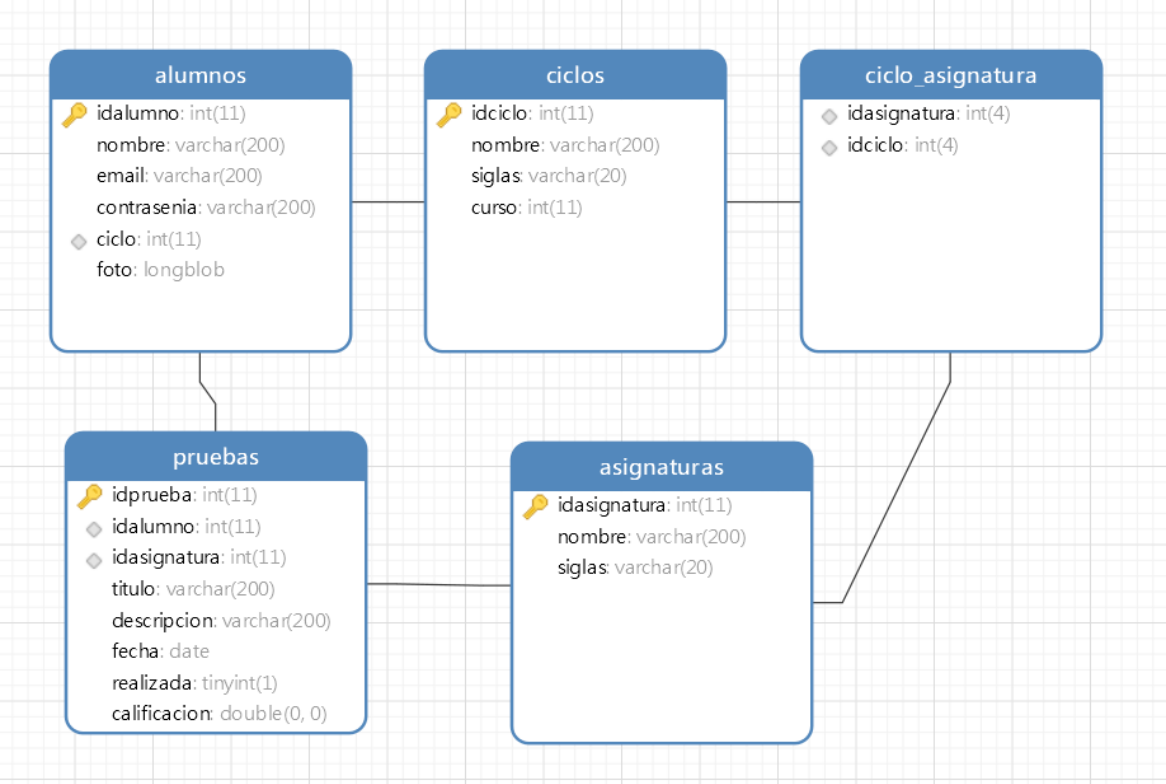
Para la Base de Datos he escogido Room, la cual proporciona una capa de abstracción sobre SQLite que permite acceder a la base de datos sin problemas y, al mismo tiempo, aprovechar toda la potencia de SQLite.

He escogido esta opción ya que es la recomendada en el Developer de Android.

Para poder utilizar esta base de datos lo primero que debemos hacer es implementar el siguiente código en el archivo build.gradle (:app)

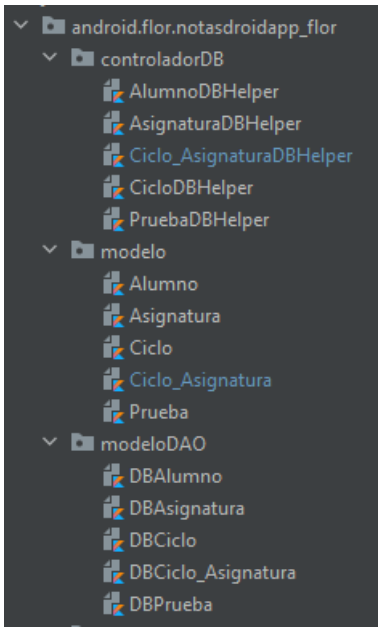
```
def room_version = "2.2.5"

implementation "androidx.room:room-runtime:$room_version"
implementation "androidx.room:room-ktx:$room_version"
kapt "androidx.room:room-compiler:$room_version"
```

Estructura de la base de datos



Implementación de la base de datos

Para cada tabla he implementado tres clases en kotlin, separadas en diferentes paquetes:



- Alumno → Representa a un usuario.
- Asignatura → Representa una asignatura.
- Ciclo → Representa un ciclo.
- Prueba → Representa una prueba.
- Ciclo_Asignatura → Representa la relación entre el ciclo y la asignatura.

- **controladorDB** → En estas clases se encuentra la gestión de las tablas de nuestra base de datos y en ellas se hacen las querys necesarias para nuestra aplicación.
  - **AlumnoDBHelper**

```kotlin
import android.content.ContentValues
import android.content.Context
import android.database.Cursor
import android.database.sqlite.SQLiteConstraintException
import android.database.sqlite.SQLiteDatabase
import android.database.sqlite.SQLiteException
import android.database.sqlite.SQLiteOpenHelper
import android.flor.notasdroidapp_flor.modelo.Alumno
import android.flor.notasdroidapp_flor.modeloDAO.DBAlumno

import java.util.ArrayList

class AlumnoDBHelper (context: Context) : SQLiteOpenHelper(context, DATABASE_NAME, null, DATABASE_VERSION) {

    companion object {
        // If you change the database schema, you must increment the database version.
        val DATABASE_VERSION = 1
        val DATABASE_NAME = "notas_droid_flor.db"

        private val SQL_CREATE_ENTRIES =
            "CREATE TABLE " + DBAlumno.AlumnoDAO.TABLE_NAME + " (" +
                    DBAlumno.AlumnoDAO.COLUMN_ID_ALUMNO + " INTEGER PRIMARY KEY AUTOINCREMENT," +
                    DBAlumno.AlumnoDAO.COLUMN_NOMBRE + " TEXT," +
                    DBAlumno.AlumnoDAO.COLUMN_EMAIL + " TEXT," +
                    DBAlumno.AlumnoDAO.COLUMN_CONTRASENIA + " TEXT," +
                    DBAlumno.AlumnoDAO.COLUMN_CICLO + " INTEGER," +
                    DBAlumno.AlumnoDAO.COLUMN_LOGUEADO + " INTEGER)"

        private val SQL_DELETE_ENTRIES = "DROP TABLE IF EXISTS " + DBAlumno.AlumnoDAO.TABLE_NAME
    }

    override fun onCreate(db: SQLiteDatabase) {
        db.execSQL(SQL_CREATE_ENTRIES)
    }

    override fun onUpgrade(db: SQLiteDatabase, oldVersion: Int, newVersion: Int) {
        // This database is only a cache for online data, so its upgrade policy is
        // to simply to discard the data and start over
        db.execSQL(SQL_DELETE_ENTRIES)
        onCreate(db)
    }

    override fun onDowngrade(db: SQLiteDatabase, oldVersion: Int, newVersion: Int) {
        onUpgrade(db, oldVersion, newVersion)
    }
```

```kotlin
/**
 * Funcion para insertar un Alumno en la BDD
 */
@Throws(SQLiteConstraintException::class)
fun insertAlumno(alumno: Alumno): Boolean {
    // Obtiene la BD en modo escritura
    val db = writableDatabase

    // Inserta un nuevo Alumno
    val values = ContentValues()
    // values.put(DBAlumno.AlumnoDAO.COLUMN_ID_CICLO, alumno.idalumno) ------ Se incrementa solo
    values.put(DBAlumno.AlumnoDAO.COLUMN_NOMBRE, alumno.nombre)
    values.put(DBAlumno.AlumnoDAO.COLUMN_EMAIL, alumno.email)
    values.put(DBAlumno.AlumnoDAO.COLUMN_CONTRASENIA, alumno.contrasenia)
    values.put(DBAlumno.AlumnoDAO.COLUMN_CICLO, alumno.ciclo)
    values.put(DBAlumno.AlumnoDAO.COLUMN_LOGUEADO, alumno.logueado)

    // Inserta el alumno nuevo devolviendo la primary key
    val nuevoAlumnoId = db.insert(DBAlumno.AlumnoDAO.TABLE_NAME, null, values)

    return true
}
```

```kotlin
/**
 * Funcion que devuelve un Alumno en una lista a partir de un idalumno pasado por parametro
 */
fun selectAlumnoID(idalumno: Int): ArrayList<Alumno> {
    val alumnos = ArrayList<Alumno>()
    val db = writableDatabase
    var cursor: Cursor? = null
    try {
        cursor = db.rawQuery("select * from " + DBAlumno.AlumnoDAO.TABLE_NAME + " WHERE " + DBAlumno.AlumnoDAO.COLUMN_ID_ALUMNO + "='" + idalumno + "'", null)
    } catch (e: SQLiteException) {
        // if table not yet present, create it
        db.execSQL(SQL_CREATE_ENTRIES)
        return ArrayList()
    }

    var nombre: String
    var email: String
    var contrasenia: String
    var ciclo: Int
    var log: Int

    if (cursor!!.moveToFirst()) {
        while (cursor.isAfterLast == false) {
            nombre = cursor.getString(cursor.getColumnIndex(DBAlumno.AlumnoDAO.COLUMN_NOMBRE))
            email = cursor.getString(cursor.getColumnIndex(DBAlumno.AlumnoDAO.COLUMN_EMAIL))
            contrasenia = cursor.getString(cursor.getColumnIndex(DBAlumno.AlumnoDAO.COLUMN_CONTRASENIA))
            ciclo = cursor.getInt(cursor.getColumnIndex(DBAlumno.AlumnoDAO.COLUMN_CICLO))
            log = cursor.getInt(cursor.getColumnIndex(DBAlumno.AlumnoDAO.COLUMN_LOGUEADO))

            alumnos.add(Alumno(idalumno, nombre, email, contrasenia, ciclo, log))
            cursor.moveToNext()
        }
    }
    return alumnos
}
```

```kotlin
/**
 * Funcion que devuelve todos los Alumnos de la BDD
 */
fun selectAllAlumnos(): ArrayList<Alumno> {
    val alumnos = ArrayList<Alumno>()
    val db = writableDatabase
    var cursor: Cursor? = null
    try {
        cursor = db.rawQuery("select * from " + DBAlumno.AlumnoDAO.TABLE_NAME, null)
    } catch (e: SQLiteException) {
        db.execSQL(SQL_CREATE_ENTRIES)
        return ArrayList()
    }

    var idalumno: Int
    var nombre: String
    var email: String
    var contrasenia: String
    var ciclo: Int
    var log: Int

    if (cursor!!.moveToFirst()) {
        while (cursor.isAfterLast == false) {
            idalumno = cursor.getInt(cursor.getColumnIndex(DBAlumno.AlumnoDAO.COLUMN_ID_ALUMNO))
            nombre = cursor.getString(cursor.getColumnIndex(DBAlumno.AlumnoDAO.COLUMN_NOMBRE))
            email = cursor.getString(cursor.getColumnIndex(DBAlumno.AlumnoDAO.COLUMN_EMAIL))
            contrasenia = cursor.getString(cursor.getColumnIndex(DBAlumno.AlumnoDAO.COLUMN_CONTRASENIA))
            ciclo = cursor.getInt(cursor.getColumnIndex(DBAlumno.AlumnoDAO.COLUMN_CICLO))
            log = cursor.getInt(cursor.getColumnIndex(DBAlumno.AlumnoDAO.COLUMN_LOGUEADO))

            alumnos.add(Alumno(idalumno, nombre, email, contrasenia, ciclo, log))
            cursor.moveToNext()
        }
    }
    return alumnos
}
```

```kotlin
/**
 * Funcion que devuelve un Alumno a partir de un email pasado por parametro
 */
fun selectAlumnoEmail(email: String): ArrayList<Alumno> {
    val alumno = ArrayList<Alumno>()
    val db = writableDatabase
    var cursor: Cursor? = null
    try {
        cursor = db.rawQuery("select * from " + DBAlumno.AlumnoDAO.TABLE_NAME + " WHERE " + DBAlumno.AlumnoDAO.COLUMN_EMAIL + "='" + email + "'", null)
    } catch (e: SQLiteException) {
        // if table not yet present, create it
        db.execSQL(SQL_CREATE_ENTRIES)
        return ArrayList()
    }

    var idalumno: Int
    var nombre: String
    var contrasenia: String
    var ciclo: Int
    var log: Int

    if (cursor!!.moveToFirst()) {
        while (cursor.isAfterLast == false) {
            idalumno = cursor.getInt(cursor.getColumnIndex(DBAlumno.AlumnoDAO.COLUMN_ID_ALUMNO))
            nombre = cursor.getString(cursor.getColumnIndex(DBAlumno.AlumnoDAO.COLUMN_NOMBRE))
            contrasenia = cursor.getString(cursor.getColumnIndex(DBAlumno.AlumnoDAO.COLUMN_CONTRASENIA))
            ciclo = cursor.getInt(cursor.getColumnIndex(DBAlumno.AlumnoDAO.COLUMN_CICLO))
            log = cursor.getInt(cursor.getColumnIndex(DBAlumno.AlumnoDAO.COLUMN_LOGUEADO))

            alumno.add(Alumno(idalumno, nombre, email, contrasenia, ciclo, log))
            cursor.moveToNext()
        }
    }
    return alumno
}
```

```kotlin
/**
 * Funcion que devuelve un Alumno logueado
 */
fun selectAlumnoLog(): ArrayList<Alumno> {
    val alumno = ArrayList<Alumno>()
    val db = writableDatabase
    var cursor: Cursor? = null
    try {
        cursor = db.rawQuery("select * from " + DBAlumno.AlumnoDAO.TABLE_NAME + " WHERE " + DBAlumno.AlumnoDAO.COLUMN_LOGUEADO + "=" + 1, null)
    } catch (e: SQLiteException) {
        // if table not yet present, create it
        db.execSQL(SQL_CREATE_ENTRIES)
        return ArrayList()
    }

    var idalumno: Int
    var nombre: String
    var email: String
    var contrasenia: String
    var ciclo: Int
    var log: Int

    if (cursor!!.moveToFirst()) {
        while (cursor.isAfterLast == false) {
            idalumno = cursor.getInt(cursor.getColumnIndex(DBAlumno.AlumnoDAO.COLUMN_ID_ALUMNO))
            nombre = cursor.getString(cursor.getColumnIndex(DBAlumno.AlumnoDAO.COLUMN_NOMBRE))
            email = cursor.getString(cursor.getColumnIndex(DBAlumno.AlumnoDAO.COLUMN_EMAIL))
            contrasenia = cursor.getString(cursor.getColumnIndex(DBAlumno.AlumnoDAO.COLUMN_CONTRASENIA))
            ciclo = cursor.getInt(cursor.getColumnIndex(DBAlumno.AlumnoDAO.COLUMN_CICLO))
            log = cursor.getInt(cursor.getColumnIndex(DBAlumno.AlumnoDAO.COLUMN_LOGUEADO))

            alumno.add(Alumno(idalumno, nombre, email, contrasenia, ciclo, log))
            cursor.moveToNext()
        }
    }
    return alumno
}
```

```kotlin
/**
 * Funcion que actualiza los datos de un alumno
 */
@Throws(SQLiteConstraintException::class)
fun updateAlumno(alumno: Alumno): Boolean {
    // Obtiene la BD en modo escritura
    val db = writableDatabase

    // Inserta un nuevo Alumno
    val values = ContentValues()
    //values.put(DBAlumno.AlumnoDAO.COLUMN_ID_CICLO, alumno.idalumno)
    values.put(DBAlumno.AlumnoDAO.COLUMN_NOMBRE, alumno.nombre)
    values.put(DBAlumno.AlumnoDAO.COLUMN_EMAIL, alumno.email)
    values.put(DBAlumno.AlumnoDAO.COLUMN_CONTRASENIA, alumno.contrasenia)
    values.put(DBAlumno.AlumnoDAO.COLUMN_CICLO, alumno.ciclo)
    values.put(DBAlumno.AlumnoDAO.COLUMN_LOGUEADO, alumno.logueado)

    val whereclause = "idalumno=?"
    val whereargs = arrayOf(alumno.idalumno.toString())
    val nuevoAlumnoId = db.update(DBAlumno.AlumnoDAO.TABLE_NAME, values, whereclause, whereargs)

    return true
}

}
```

- **AsignaturaDBHelper**
- **Ciclo_AsignaturaDBHelper**
- **CicloDBHelper**
- **PruebaDBHelper**

➢ **modelo** → En estas clases se encuentras los objetos necesarios para la comunicación entre nuestro código y la base de datos.

- **AlumnoDBHelper**

```
package android.flor.notasdroidapp_flor.modelo

class Alumno(val idalumno: Int, var nombre: String, var email: String, var contrasenia: String, var ciclo: Int, var logueado: Int)
```

- **AsignaturaDBHelper**
- **Ciclo_AsignaturaDBHelper**
- **CicloDBHelper**
- **PruebaDBHelper**

➢ **modeloDAO** → En estas clases se encuentran los esquemas de las propias tablas.

- **AlumnoDBHelper**

```
package android.flor.notasdroidapp_flor.modeloDAO

import android.provider.BaseColumns

object DBAlumno {

    class AlumnoDAO : BaseColumns {
        companion object {
            val TABLE_NAME = "alumnos"
            val COLUMN_ID_ALUMNO = "idalumno"
            val COLUMN_NOMBRE = "nombre"
            val COLUMN_EMAIL = "email"
            val COLUMN_CONTRASENIA = "contrasenia"
            val COLUMN_CICLO = "ciclo"
            val COLUMN_LOGUEADO = "logueado"
        }
    }
}
```

- **AsignaturaDBHelper**
- **Ciclo_AsignaturaDBHelper**
- **CicloDBHelper**
- **PruebaDBHelper**

- **Para poder crear el archivo .bd con nuestra base de datos, he creado un método en mi Activity principal donde cargamos todos los datos con los inserts necesarios.**

Creamos los objetos necesarios como atributos de la clase:

```
class MainActivity : AppCompatActivity() {
    lateinit var alumnosDBHelper: AlumnoDBHelper
    lateinit var asigDBHelper: AsignaturaDBHelper
    lateinit var cicloDBHelper: CicloDBHelper
    lateinit var relacionDBHelper: Ciclo_AsignaturaDBHelper
    lateinit var pruebaDBHelper: PruebaDBHelper
```

Y después llamamos a nuestro método cargarBBDD() en el onCreate de dicha actividad. Este método contiene lo siguiente:

```
    /**
     * Funcion que carga todos los datos necesarios de la BBDD
     */
    fun cargarBBDD(): Unit {

        // Tabla Alumnos
        alumnosDBHelper = AlumnoDBHelper(this)

        // Tabla Ciclos
        cicloDBHelper = CicloDBHelper(this)
        cicloDBHelper.insertarTodosCiclos()
//        var ciclos = cicloDBHelper.selectAllCiclos()

        // Tabla Asignaturas
        asigDBHelper = AsignaturaDBHelper(this)
        asigDBHelper.insertarTodasAsignaturas()
//        var asig = asigDBHelper.selectAllAsignaturas()

        // Tabla Ciclo_Asignatura
        relacionDBHelper = Ciclo_AsignaturaDBHelper(this)
        relacionDBHelper.insertarTodasRelaciones()
//        var relaciones = relacionDBHelper.selectAllCicloAsignaturas()

        // Tabla Pruebas
        pruebaDBHelper = PruebaDBHelper(this)
//        var pruebas = pruebaDBHelper.selectAllPruebas()
    }
```

Realiza las diferentes llamadas a los métodos para rellenar con la información necesaria para la correcta funcionalidad de la aplicación.

```
/**
 * Funcion que inserta los valores necesarios a la tabla Ciclos
 */
fun insertarTodosCiclos(): Unit {
    if (this.selectAllCiclos().isEmpty()) {
        insertCiclo(Ciclo(0, "Administración de Sistemas Informáticos en Red", "ASIR", 1))
        insertCiclo(Ciclo(0, "Administración de Sistemas Informáticos en Red", "ASIR", 2))
        insertCiclo(Ciclo(0, "Desarrollo de Aplicaciones Multiplataforma", "DAM", 1))
        insertCiclo(Ciclo(0, "Desarrollo de Aplicaciones Multiplataforma", "DAM", 2))
        insertCiclo(Ciclo(0, "Desarrollo de Aplicaciones Web", "DAW", 1))
        insertCiclo(Ciclo(0, "Desarrollo de Aplicaciones Web", "DAW", 2))
    }
}
```

```
/**
 * Funcion que inserta los valores necesarios a la tabla Asignaturas
 */
fun insertarTodasAsignaturas(): Unit {
    if (this.selectAllAsignaturas().isEmpty()){
    insertAsignatura(Asignatura(22,"Inglés técnico para ciclos de grado superior de la familia de informética y comunicaciones", "ING"))
    insertAsignatura(Asignatura(369,"Implantación de Sistemas Operativos", "ISO"))
    insertAsignatura(Asignatura(370,"Planificación y Administración de Redes", "PAR"))
    insertAsignatura(Asignatura(371,"Fundamentos de Hardware", "FH"))
    insertAsignatura(Asignatura(372,"Gestión de Bases de Datos", "GB"))
    insertAsignatura(Asignatura(373,"Lenguajes de Marcas y Sistemas de Gestión de Información", "LMSGI"))
    insertAsignatura(Asignatura(374,"Administración de Sistemas Operativos", "ASO"))
    insertAsignatura(Asignatura(375,"Servicios de Red e Internet", "SRI"))
    insertAsignatura(Asignatura(376,"Implantación de Aplicaciones Web", "IAQ"))
    insertAsignatura(Asignatura(377,"Administración de Sistemas Gestores de Bases de Datos", "ASGBD"))
    insertAsignatura(Asignatura(378,"Seguridad y Alta Disponibilidad", "SAD"))
    insertAsignatura(Asignatura(380,"Formación y Orientación Laboral", "FOL"))
    insertAsignatura(Asignatura(381,"Empresa e Iniciativa Emprendedora", "EIE"))
    insertAsignatura(Asignatura(483,"Sistemas Informáticos", "SISIN"))
    insertAsignatura(Asignatura(484,"Bases de Datos", "BDD"))
    insertAsignatura(Asignatura(485,"Programación", "PROG"))
    insertAsignatura(Asignatura(487,"Entornos de Desarrollo", "ENDES"))
    insertAsignatura(Asignatura(493,"Formación y Orientación Laboral", "FOL"))
    insertAsignatura(Asignatura(486,"Acceso a Datos", "AD"))
    insertAsignatura(Asignatura(488,"Desarrollo de Interfaces", "DI"))
    insertAsignatura(Asignatura(489,"Programación Multimedia y Dispositivos Móviles", "PMDM"))
    insertAsignatura(Asignatura(490,"Programación de Servicios y Procesos", "PSP"))
    insertAsignatura(Asignatura(491,"Sistemas de Gestión Empresarial", "SGE"))
    insertAsignatura(Asignatura(494,"Empresa e Iniciativa Emprendedora", "EIE"))
    insertAsignatura(Asignatura(617,"Formación y Orientación Laboral", "FOL"))
    insertAsignatura(Asignatura(612,"Desarrollo Web en Entorno Cliente", "DWEC"))
    insertAsignatura(Asignatura(613,"Desarrollo Web en Entorno Servidor", "DEWS"))
    insertAsignatura(Asignatura(614,"Despliegue de Aplicaciones Web", "DAW"))
    insertAsignatura(Asignatura(615,"Diseño de Interfaces Web", "DIW"))
    insertAsignatura(Asignatura(618,"Empresa e iniciativa Emprenderora", "EIE"))
    }
}
```

```kotlin
/**
 * Funcion que inserta los valores necesarios a la tabla Ciclo-Asignatura
 */
fun insertarTodasRelaciones(): Unit {
    if (this.selectAllCicloAsignaturas().isEmpty()) {
        insertCicloAsignatura(Ciclo_Asignatura(22,1))
        insertCicloAsignatura(Ciclo_Asignatura(22,3))
        insertCicloAsignatura(Ciclo_Asignatura(22,5))
        insertCicloAsignatura(Ciclo_Asignatura(369,1))
        insertCicloAsignatura(Ciclo_Asignatura(370,1))
        insertCicloAsignatura(Ciclo_Asignatura(371,1))
        insertCicloAsignatura(Ciclo_Asignatura(372,1))
        insertCicloAsignatura(Ciclo_Asignatura(373,1))
        insertCicloAsignatura(Ciclo_Asignatura(373,3))
        insertCicloAsignatura(Ciclo_Asignatura(373,5))
        insertCicloAsignatura(Ciclo_Asignatura(374,2))
        insertCicloAsignatura(Ciclo_Asignatura(375,2))
        insertCicloAsignatura(Ciclo_Asignatura(376,2))
        insertCicloAsignatura(Ciclo_Asignatura(377,2))
        insertCicloAsignatura(Ciclo_Asignatura(378,2))
        insertCicloAsignatura(Ciclo_Asignatura(380,1))
        insertCicloAsignatura(Ciclo_Asignatura(381,2))
        insertCicloAsignatura(Ciclo_Asignatura(483,3))
        insertCicloAsignatura(Ciclo_Asignatura(483,5))
        insertCicloAsignatura(Ciclo_Asignatura(484,3))
        insertCicloAsignatura(Ciclo_Asignatura(484,5))
        insertCicloAsignatura(Ciclo_Asignatura(485,3))
        insertCicloAsignatura(Ciclo_Asignatura(485,5))
        insertCicloAsignatura(Ciclo_Asignatura(487,3))
        insertCicloAsignatura(Ciclo_Asignatura(487,5))
        insertCicloAsignatura(Ciclo_Asignatura(493,3))
        insertCicloAsignatura(Ciclo_Asignatura(486,4))
        insertCicloAsignatura(Ciclo_Asignatura(488,4))
        insertCicloAsignatura(Ciclo_Asignatura(489,4))
        insertCicloAsignatura(Ciclo_Asignatura(490,4))
        insertCicloAsignatura(Ciclo_Asignatura(491,4))
        insertCicloAsignatura(Ciclo_Asignatura(494,4))
        insertCicloAsignatura(Ciclo_Asignatura(617,5))
        insertCicloAsignatura(Ciclo_Asignatura(612,6))
        insertCicloAsignatura(Ciclo_Asignatura(613,6))
        insertCicloAsignatura(Ciclo_Asignatura(614,6))
        insertCicloAsignatura(Ciclo_Asignatura(615,6))
        insertCicloAsignatura(Ciclo_Asignatura(618,6))
    }
}
```

2.2.2. Descomposición

Mi aplicación está preparada para varias resoluciones. He realizado pruebas en tablets de 10.1p, 9.94p y 7p; y en móviles de 7.3p, 6p y 5p y el cambio de orientación en todas ellas, en versiones de Android de de la 6 en adelante.

Está dividida en diferentes Activities y Fragments:

- **SplashScreen** → Para crear esta pantalla, hemos tenido que crear un nuevo estilo en el archivo styles.xml

```
<!-- Estilo de la Splash Screen -->
<style name="SplashTheme" parent="AppTheme">
    <item name="android:windowFullscreen">true</item>
    <item name="android:windowBackground">@drawable/splashscreen</item>
</style>
<style name="AppTheme.NoActionBar">
    <item name="windowActionBar">false</item>
    <item name="windowNoTitle">true</item>
</style>
<style name="AppTheme.AppBarOverlay" parent="ThemeOverlay.AppCompat.Dark.ActionBar"/>
<style name="AppTheme.PopupOverlay" parent="ThemeOverlay.AppCompat.Light"/>
```

Después creamos una actividad en el AndroidManifest.xml

```
<!-- Activity Splash -->
<activity
        android:name=".MainActivity"
        android:theme="@style/SplashTheme">
    <intent-filter>
        <action android:name="android.intent.action.MAIN"/>

        <category android:name="android.intent.category.LAUNCHER"/>
    </intent-filter>
</activity>
```

Y por último le damos un tiempo y llamamos al estilo creado en el onCreate de nuestra actividad principal

```
//Le damos tiempo
Thread.sleep(3000)
//Llamamos al tema creado para la splash
setTheme(R.style.AppTheme)
```

- **MainActivity** → En esta clase creamos la base de datos, llamamos a nuestra SplashScreen y validamos el login del usuario.

  Método para validar el login y que abra la pantalla de login o entre directamente a nuestra pantalla de inicio:

```
//Validar login
var alumno = alumnosDBHelper.selectAlumnoLog()

if (alumno.isEmpty()) {
    val intent = Intent(this, LoginActivity::class.java)
    startActivity(intent)
} else {
    //Una vez terminada la splash, y comprobado el login nos lleva a la pagina de Principal
    val intent = Intent(this, PrincipalActivity::class.java)
    intent.putExtra("Email", alumno[0].email)
    startActivity(intent)
}
```

- **LoginActivity** → En esta actividad el usuario tendrá que introducir sus credenciales para entrar a la aplicación. Tenemos dos eventos asociados a un botón y un text view, uno para entrar en la aplicación (implementación en la siguiente imagen) y el otro para registrarse, el cuál nos lleva a la activity de registro, respectivamente.

```
// Funcion onClick del boton Entrar
fun entrar(view: View) {
    try {

        alumnosDBHelper = AlumnoDBHelper(this)
        var alumno: Alumno
        var email: String
        var contrasenia: String

        email = editTextTextEmailAddressLogin.text.toString()
        contrasenia = editTextTextPasswordLogin.text.toString()

        if (email.equals("") || contrasenia.equals("")) {
            Toast.makeText(this, "Los campos no pueden estar vacíos", Toast.LENGTH_SHORT).show()

        } else {

            var alumnos = alumnosDBHelper.selectAlumnoEmail(email)

            if (alumnos.isEmpty()) {
                Toast.makeText(this, "No existe el usuario", Toast.LENGTH_SHORT).show()
            } else {
                alumno = alumnos[0]
                if (alumno.contrasenia.equals(contrasenia)) {
                    alumno.logueado = 1
                    alumnosDBHelper.updateAlumno(alumno)

                    val intent = Intent(this, PrincipalActivity::class.java).apply {
                    }
                    intent.putExtra("Email", alumno.email)
                    startActivity(intent)

                } else {
                    Toast.makeText(this, "Contraseña incorrecta", Toast.LENGTH_SHORT).show()
                }
            }
        }
    } catch (e: Exception) {
        Toast.makeText(this, "Se ha producido un Error", Toast.LENGTH_SHORT).show()
    }
}
```

- **RegistroActivity** → En esta actividad podremos crear un nuevo usuario alumno para poder entrar en la aplicación.

  Para cargar los combos con la información de los ciclos y los cursos hacemos lo siguiente dentro del onCreate:

```
//SPINNER CICLO
val listaCiclos = cicloDBHelper.selectCicloByCurso(1)

val spinnerCiclo = findViewById<Spinner>(R.id.spinnerRegistroCiclo)
if (spinnerCiclo != null) {
    val adapter = ArrayAdapter(
        this,
        android.R.layout.simple_spinner_item, listaCiclos
    )
    spinnerCiclo.adapter = adapter
}

//SPINNER CURSO
val listaCursos = ArrayList<String>()
listaCursos.add("1")
listaCursos.add("2")

val spinnerCurso = findViewById<Spinner>(R.id.spinnerRegistroCurso)
if (spinnerCurso != null) {
    val adapter = ArrayAdapter(
        this,
        android.R.layout.simple_spinner_item, listaCursos
    )
    spinnerCurso.adapter = adapter
}
```

  Para hacer el registro y guardar el usuario, primero debemos asignar en variables los textos introducidos en los distintos campos de nuestra actividad.

```
var alumno: Alumno
var nombre: String
var email: String
var contrasenia: String
var comproContrasenia: String
var ciclo: String
var curso: String
var idciclo: Int

nombre = editTextRegistroNombre.text.toString()
email = editTextRegistroEmail.text.toString()
ciclo = spinnerRegistroCiclo.selectedItem.toString()
curso = spinnerRegistroCurso.selectedItem.toString()
contrasenia = editTextRegistroContrasenia.text.toString()
comproContrasenia = editTextRegistroConfirmarContrasenia.text.toString()
```

Y después haremos las comprobaciones necesarias y si todo es correcto, insertamos el usuario y volvemos al LoginActivity:

```
if (nombre.equals("") || email.equals("") || ciclo.equals("") || contrasenia.equals("")
    || comproContrasenia.equals("")|| curso.equals("")) {

    Toast.makeText(this, "Los campos no pueden estar vacíos", Toast.LENGTH_SHORT).show()

} else {
    var alumnos = alumnosDBHelper.selectAlumnoEmail(email)

    // Si no devuelve alumnos, creamos uno nuevo
    if (alumnos.isEmpty()) {
        // Comprobamos el mail
        if (validarMail(email)) {
            // Si las contrasenias coinciden...
            if (contrasenia.equals(comproContrasenia)) {
                // Buscamos el id del ciclo
                var ciclos = cicloDBHelper.selectCicloByCursoNombre(ciclo, Integer.parseInt(curso))
                idciclo = ciclos[0].idciclo

                alumno = Alumno(0, nombre, email, contrasenia, idciclo, 0)

                alumnosDBHelper.insertAlumno(alumno)

                Toast.makeText(this, "Usuario creado correctamente", Toast.LENGTH_SHORT).show()

                val intent = Intent(this, LoginActivity::class.java).apply {
                }
                startActivity(intent)

            } else {
                Toast.makeText(this, "Las contraseñas no coindicen", Toast.LENGTH_SHORT).show()
            }
        } else {
            Toast.makeText(this, "El mail no tiene el formato correcto", Toast.LENGTH_SHORT).show()
        }

    } else {
        Toast.makeText(this, "El usuario ya existe", Toast.LENGTH_SHORT).show()

    }
}
```

- **IonBackPressed** → Interface creada para poder capturar el evento de ir para atrás para poder salirse de la aplicación en el InicioFragment.

```
interface IOnBackPressed {
    fun onBackPressed(): Boolean
}
```

- **PrincipalActivity** → Esta actividad está creada a partir de un Navigation Drawer Activity, la cual nos crea diferentes vistas y diferentes fragments:
  - **InicioFragment** → Se muestra un mensaje de bienvenida
  - **MiMatriculaFragment** → En este fragment se cargan todas las asignaturas que tiene el usuario logueado con CardViews
  - **PruebasFragment** → Aquí tenemos otro cardview con las distintas pruebas que tiene la asignatura del usuario logueado.
  - **PruebasDetalleFragment** → En este fragment se muestran los detalles de una prueba en concreto.
  - **MiExpedienteFragment** → En este fragment mostraremos todos los datos del usuario logueado y le damos la posibilidad de compartirlo por mail o en las redes sociales.
  - **AjustesFragment** → En este fragment mostraremos todos los datos del usuario logueado y le damos la posibilidad de modificarlos y actualizar su información.
    Primero cargamos todos los datos del usuario en los componentes y, al igual que en RegistroActivity, validamos todos los campos y hacemos un update en lugar de un insert.
  - **CerrarSesionFragment** → En este fragment preguntamos al usuario si de verdad desea cerrar sesión y, si acepta, hacemos un update del alumno modificando el campo de logueado y lo devolvemos al LoginActivity.

```
fun aceptar() {
    try {
        Toast.makeText(activity, "Aceptar.", Toast.LENGTH_SHORT)
        val intent = Intent(activity, LoginActivity::class.java)
        startActivity(intent)

        // Deslogueamos al usuario
        alumnosDBHelper = AlumnoDBHelper(requireContext())

        alumno = Alumno(alumno.idalumno, alumno.nombre, alumno.email, alumno.contrasenia, alumno.ciclo, 0)

        if (alumnosDBHelper.updateAlumno(alumno)) {
            Toast.makeText(requireContext(), "Sesión cerrada", Toast.LENGTH_SHORT).show()
        }

    } catch (e: Exception) {
        Toast.makeText(requireContext(), "Se ha producido un Error", Toast.LENGTH_SHORT).show()
    }

}
```

Enlace Repositorio GitHub → https://github.com/florius88/NotasDroid_Flor
Enlace Video Youtube → https://youtu.be/8vE7TLS5Qd8