TASK 4 – USER INTERFACE DEVELOPMENT

Ismael Juárez García

CIDEAD
HIGHER DEGREE MULTIPLATFORM APPLICATION DEVELOPMENT

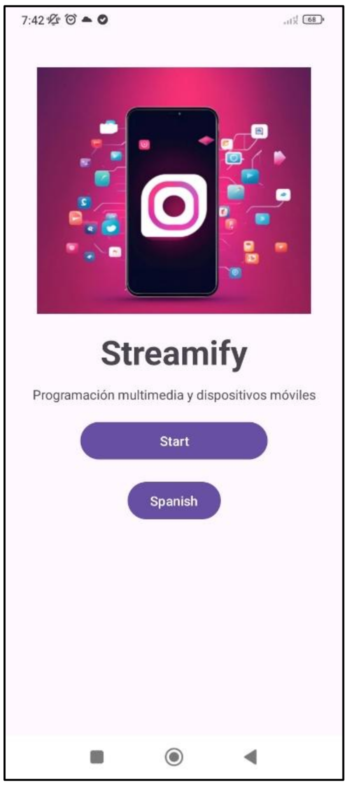
SCREEN DESIGN

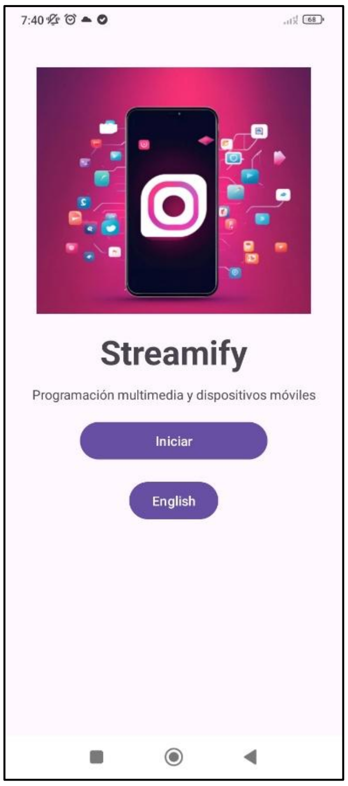
LAYOUT 1 – activity_main

The initial layout uses a ScrollView to scroll and see everything in the landscape view and a ConstraintLayout to organize the visual elements.

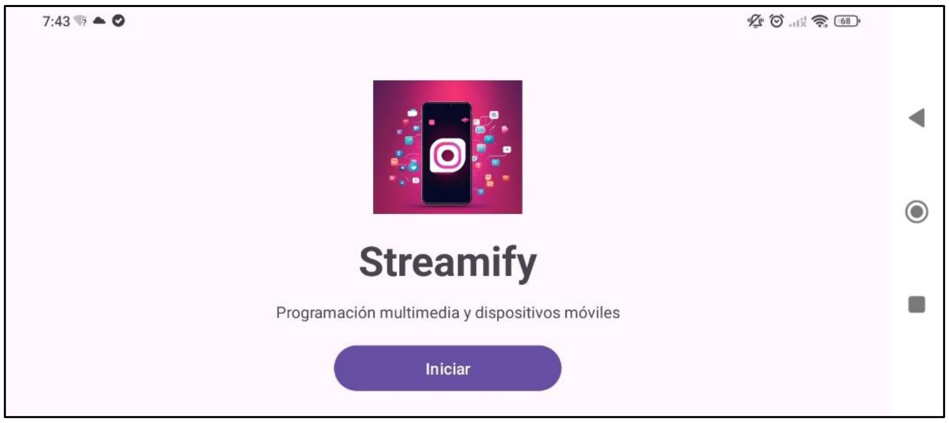
There is a button to start the application and another to change the language application (Spanish/English).

Gideline is also used to help position the elements proportionally.





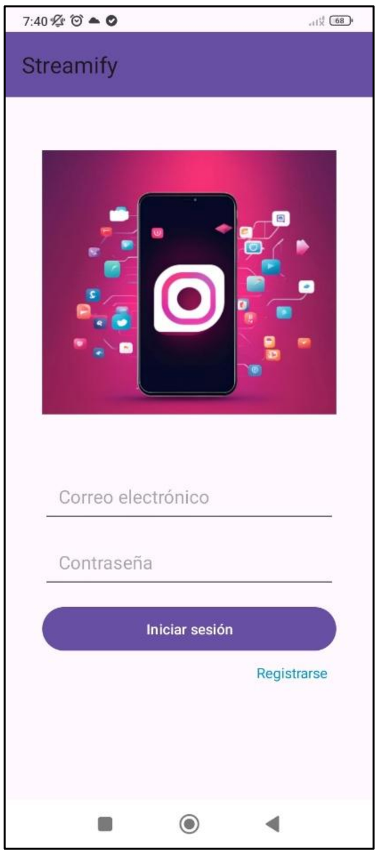
Vertical in English Vertical in Spanish



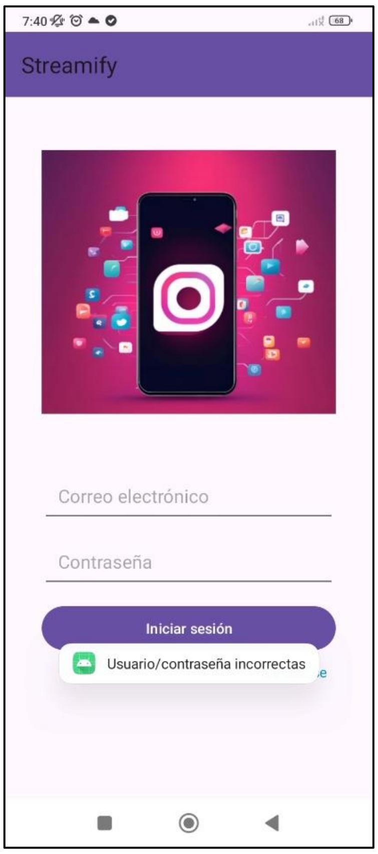
Landscape view

LAYOUT 2 – activity_login

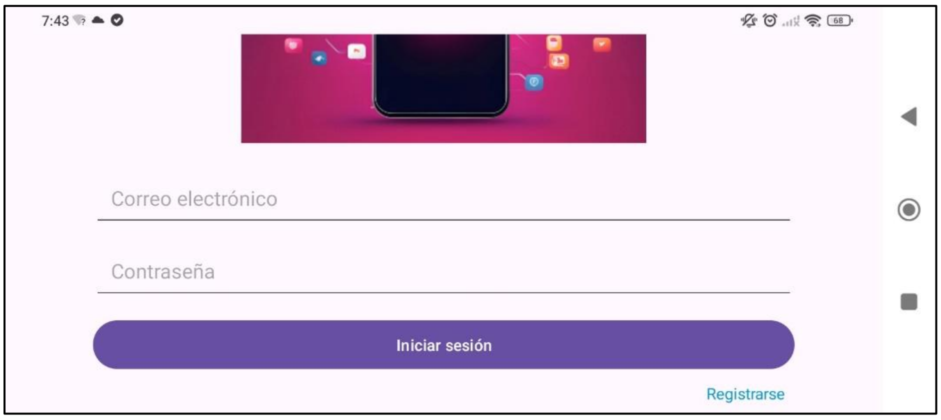
It defines the login screen, uses a ScrollView to move when the screen is horizontal and a ConstraintLayout to organize the elements, EditText is used for the email and password and there is a button and a Toolbar, which on this screen only shows the name of the App.



View Vertical



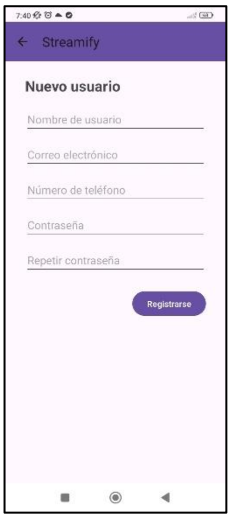
Vertical Error



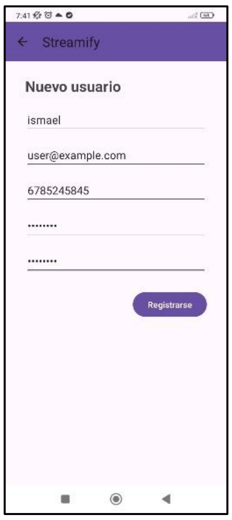
View Horizontal

LAYOUT 3 – activity_register

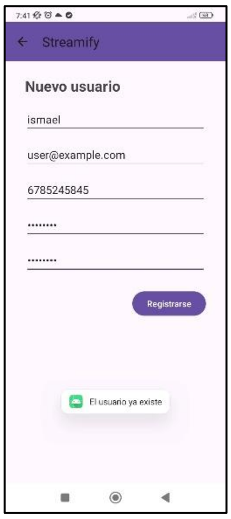
It defines the registration screen, uses a ScrollView with ConstraintLayout, EditText for the different switches and a registration button, the Toolbar has the back button.



Vertical

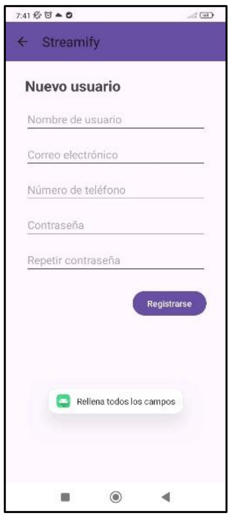


Vertical Fill



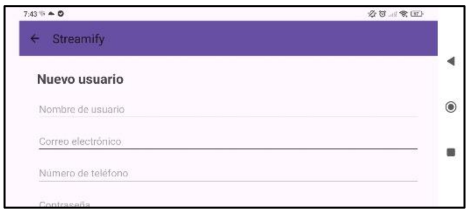
El usuario ya existe

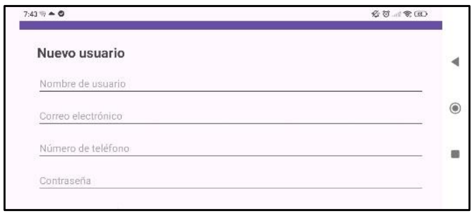
Error Exis



Rellena todos los campos

Error Not Fill



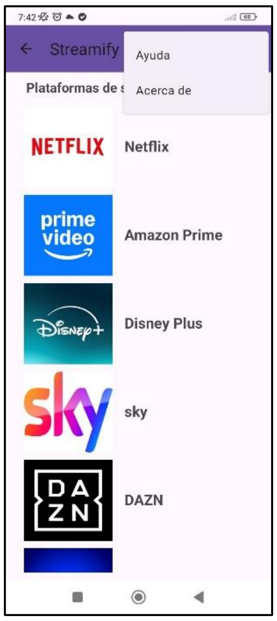


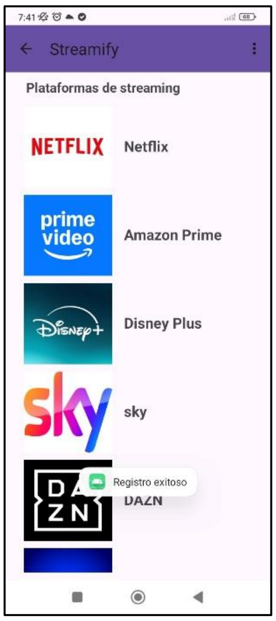
View Horizontal

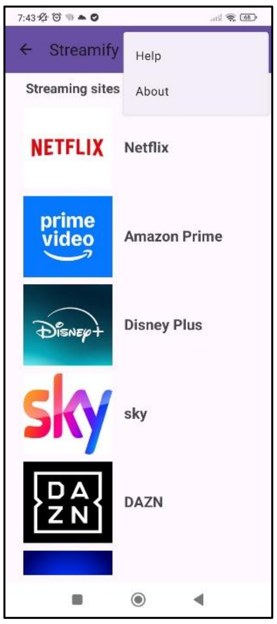
LAYOUT 4 – activity_platforms

It defines the platform screen, organized with a ConstraintLayout and wrapped in a ScrollView to allow horizontal scrolling when necessary, it includes a top Toolbar with the name of the app, a button to go back and the drop-down menu that differentiates us from the help and about activities.

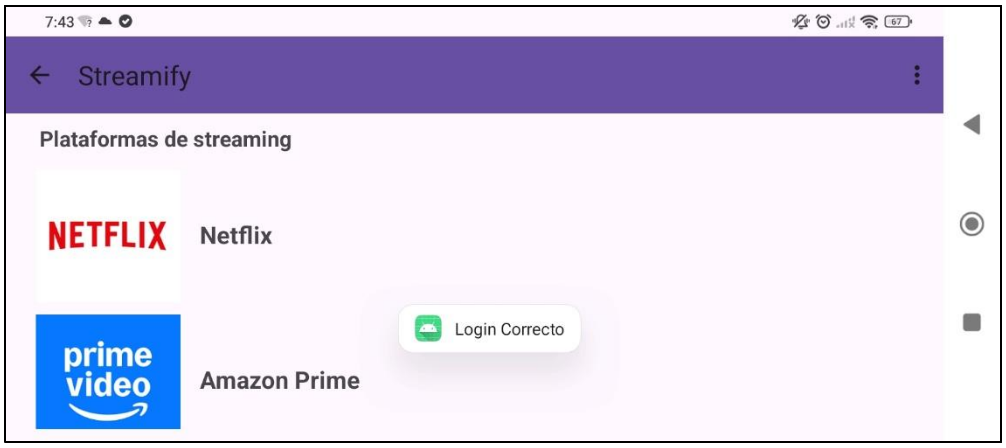
To display the platforms a RecyclerView is used, each row of the list is designed with a personalized layout (rec_row_platforms), which shows an image of the platform and its name.







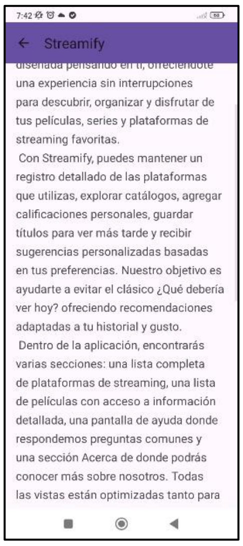
Vertical **Vertical Reg.** **Vertical Eng.**



Horizontal View

LAYOUT 5 – activity_help

ScrollView with a TextView with an app help text and features the Toolbar with a back button.

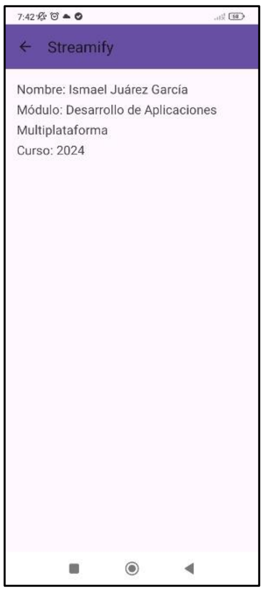

7:42

Streamify

diseñada pensando en ti, ofreciéndote
una experiencia sin interrupciones
para descubrir, organizar y disfrutar de
tus películas, series y plataformas de
streaming favoritas.
Con Streamify, puedes mantener un
registro detallado de las plataformas
que utilizas, explorar catálogos, agregar
calificaciones personales, guardar
títulos para ver más tarde y recibir
sugerencias personalizadas basadas
en tus preferencias. Nuestro objetivo es
ayudarte a evitar el clásico ¿Qué debería
ver hoy? ofreciendo recomendaciones
adaptadas a tu historial y gusto.
Dentro de la aplicación, encontrarás
varias secciones: una lista completa
de plataformas de streaming, una lista
de películas con acceso a información
detallada, una pantalla de ayuda donde
respondemos preguntas comunes y
una sección Acerca de donde podrás
conocer más sobre nosotros. Todas
las vistas están optimizadas tanto para

LAYOUT 6 - activity_about

Constraint Layout with ScrollView and a TextView that displays information about the app developer. It has the Toolbar.


7:42

Streamify

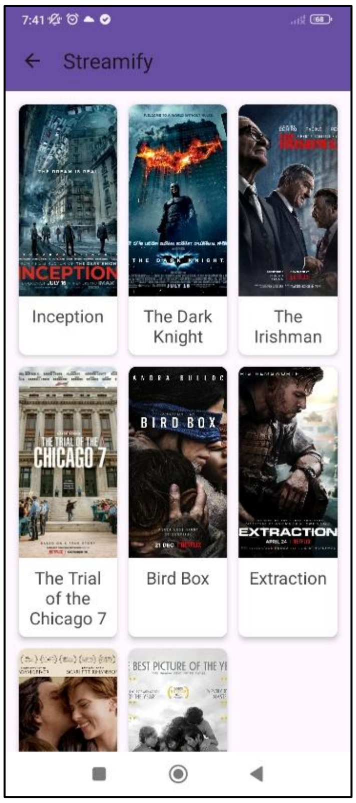
Nombre: Ismael Juárez García
Módulo: Desarrollo de Aplicaciones Multiplataforma
Curso: 2024

LAYOUT 7 – activity_films

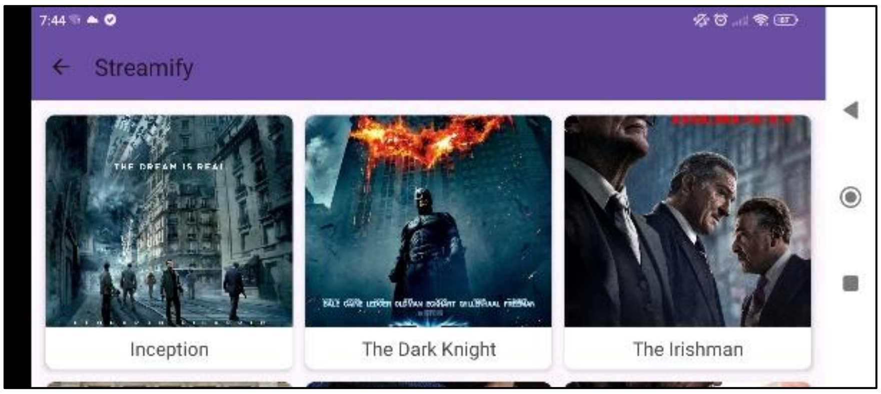
It defines the movie listing screen with a ConstraintLayout, includes the ToolBar and a RecyclerView configured with GridLayoutManager to display movies in a 3-column grid.

Each element has been designed with a custom CarView, which contains the image of the movie and its title underneath.

To allow all movies to be viewed in portrait/landscape mode, a ScrollView has been added.



Vertical View

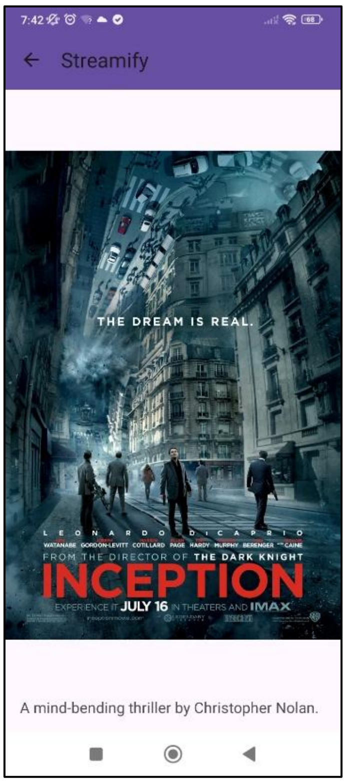


Horizontal View

LAYOUT 8 – activity_film_detail

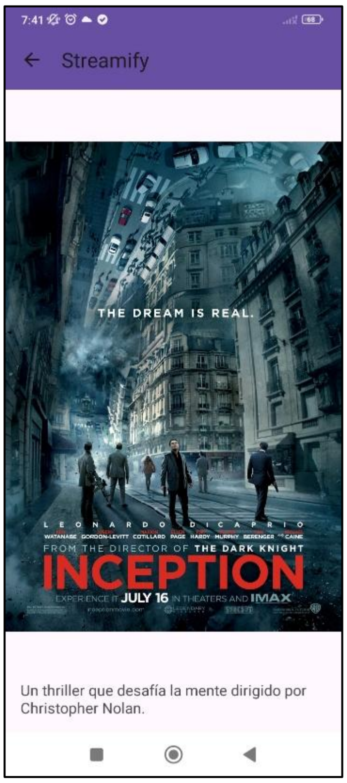
Define the detail screen of a movie with ConstraintLayout, include the top Toolbar, a featured image of the movie, and a Textview with its description.

Elements are arranged vertically and adjust to the margins to accommodate different screen sizes.



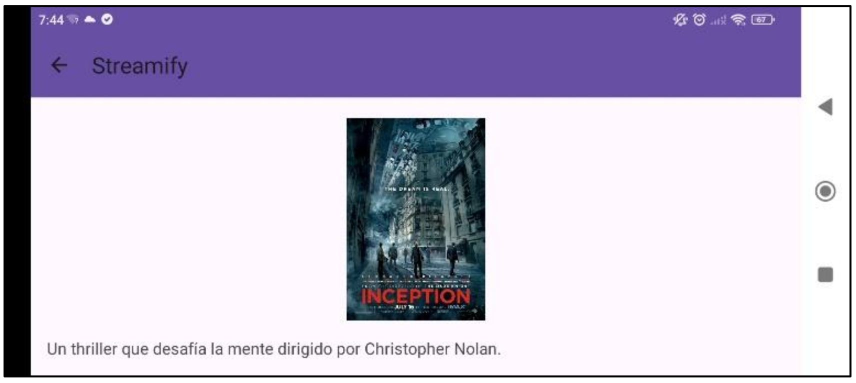
A mind-bending thriller by Christopher Nolan.

Vertical View Eng



Un thriller que desafía la mente dirigido por Christopher Nolan.

Vertical View Spa



Un thriller que desafía la mente dirigido por Christopher Nolan.

Horizontal View

PROGRAMMING OF FUNCTIONALITIES

TUITION

Platform Class, created to display streaming platforms, has three attributes (an identifier, name, and an image), the class implements the **Parcelable** interface to allow the passage of objects between activities using **Intent.**

PlatfomrList class, acts as a container and loader for platform data, manages a list, and provides the **loadPlatforms()** method for loading predefined platforms with their names and images.

Film Class represents a film with the attributes: title, identifier, description (which is referenced by a text resource), image (also referenced), identifier of the platform to which it belongs, the year of release and the genre.

Implements a **create** method to create the objects of type Film with automatic ids, using the currentID property.

FilmList class, manages the movies of the different platforms of the app. To obtain the movies of a specific platform, a map is used that contains the movies of each platform according to their ID.

The method **getFilmsByPlatfomr** receives an ID and returns the corresponding movie list to the corresponding platform, if the ID is not found, an empty list is returned.

UserManager class, this class simulates the management of user registration and authentication, implemented using a mutableMap, where the keys are the emails and the values are their passwords.

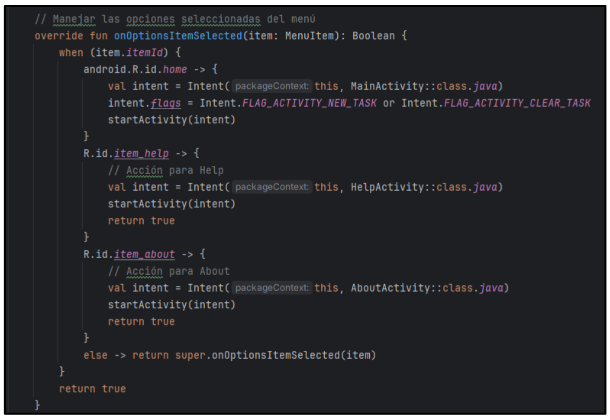
The class starts with the example username and password, which will be the only one in the application **user@example.com** With password **1234**, the methods are implemented **isUserRegistred**, which checks if the email is in users; The method **validateLogin**, to which a password and password are passed and checks if they match; and the method **registerUser**, to which the email and password are passed and if the email is not registered it returns true, thus making a simulation that the user has registered in the app.

ACTIVITIES

First of all, comment on the common parts in most of the activities so as not to repeat it in the different parts:

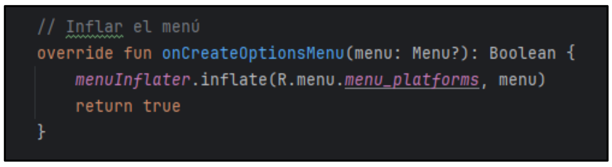
my_toolbar (androidx.appcompat.widget.Toolbar):

- To add it to the displayed method, the onCreate() setSupportActionBar(findViewById(R.id.my_tooblar) method is added.
- To activate the back button, add the OnCreate() supportActionBar? method. setDisplayHomeAsUpEnabled(True), which must override its functionality:



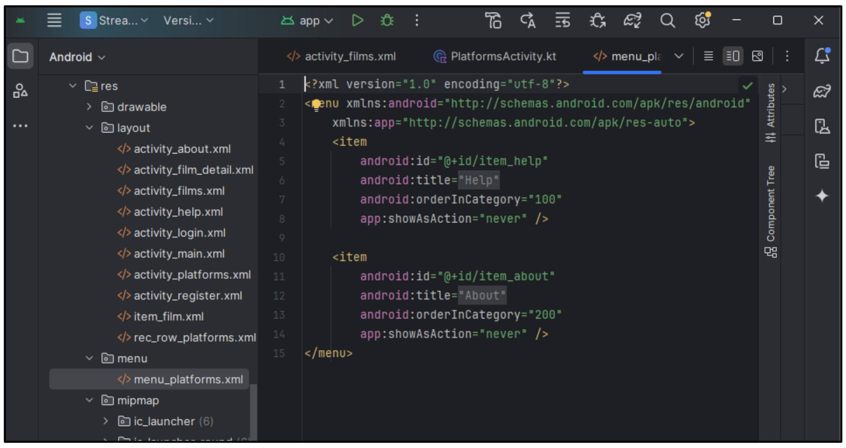
```kotlin
// Manejar las opciones seleccionadas del menú
override fun onOptionsItemSelected(item: MenuItem): Boolean {
    when (item.itemId) {
        android.R.id.home -> {
            val intent = Intent( packageContext: this, MainActivity::class.java)
            intent.flags = Intent.FLAG_ACTIVITY_NEW_TASK or Intent.FLAG_ACTIVITY_CLEAR_TASK
            startActivity(intent)
        }
        R.id.item_help -> {
            // Acción para Help
            val intent = Intent( packageContext: this, HelpActivity::class.java)
            startActivity(intent)
            return true
        }
        R.id.item_about -> {
            // Acción para About
            val intent = Intent( packageContext: this, AboutActivity::class.java)
            startActivity(intent)
            return true
        }
        else -> return super.onOptionsItemSelected(item)
    }
    return true
}
```

This is the most complete case of Toolbar, which is in the Platforms activity, where the home button and the help and about drop-down buttons are displayed.



```kotlin
// Inflar el menú
override fun onCreateOptionsMenu(menu: Menu?): Boolean {
    menuInflater.inflate(R.menu.menu_platforms, menu)
    return true
}
```

This method adds menu functionality, * the menu drop-down options are created in an xml resources menu on android:

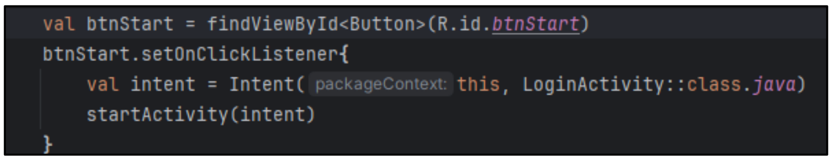


```xml
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">
    <item
        android:id="@+id/item_help"
        android:title="Help"
        android:orderInCategory="100"
        app:showAsAction="never" />

    <item
        android:id="@+id/item_about"
        android:title="About"
        android:orderInCategory="200"
        app:showAsAction="never" />
</menu>
```

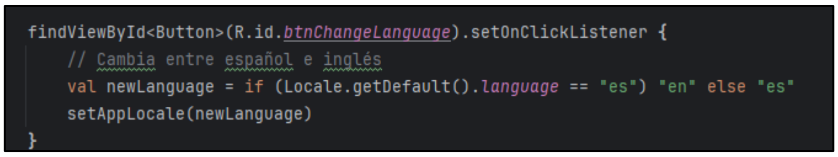
Buttons:

- **Start button**, the one that starts the application, which captures setOnClickListener and throws an intent to start the LoginActivity:



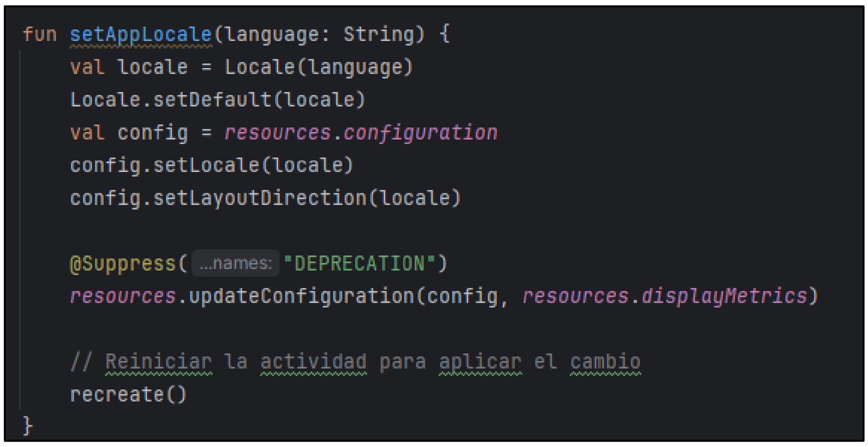
```
val btnStart = findViewById<Button>(R.id.btnStart)
btnStart.setOnClickListener{
    val intent = Intent( packageContext: this, LoginActivity::class.java)
    startActivity(intent)
}
```

- Language change button, the same setOnClickListener event is captured and the language is changed (to change the language there must be 2 string resources with the respective languages):



```
findViewById<Button>(R.id.btnChangeLanguage).setOnClickListener {
    // Cambia entre español e inglés
    val newLanguage = if (Locale.getDefault().language == "es") "en" else "es"
    setAppLocale(newLanguage)
}
```

The setAppLocate **function** changes the configuration of the resources:

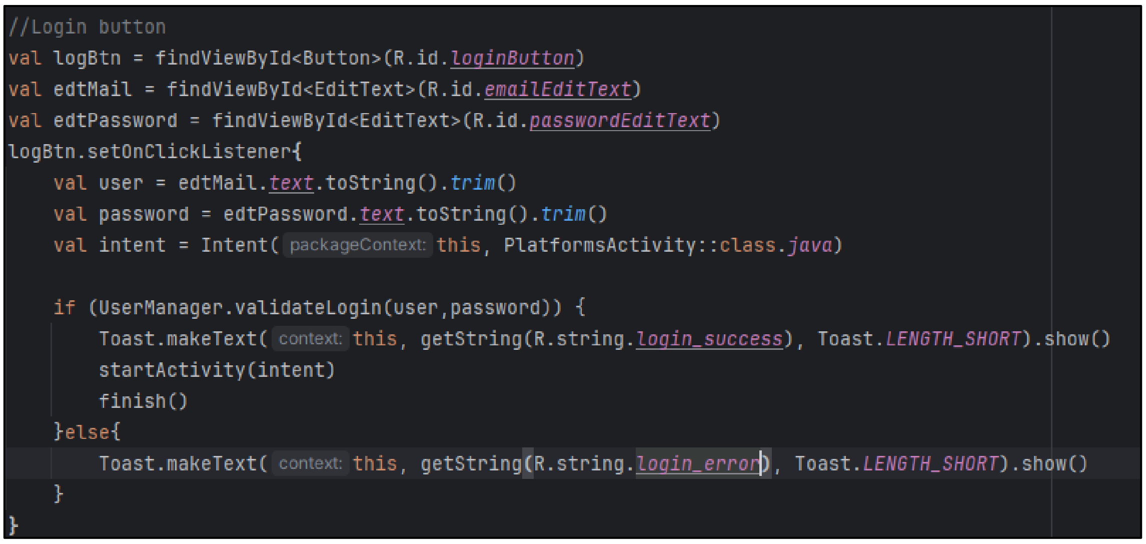


```
fun setAppLocale(language: String) {
    val locale = Locale(language)
    Locale.setDefault(locale)
    val config = resources.configuration
    config.setLocale(locale)
    config.setLayoutDirection(locale)

    @Suppress( ...names: "DEPRECATION")
    resources.updateConfiguration(config, resources.displayMetrics)

    // Reiniciar la actividad para aplicar el cambio
    recreate()
}
```

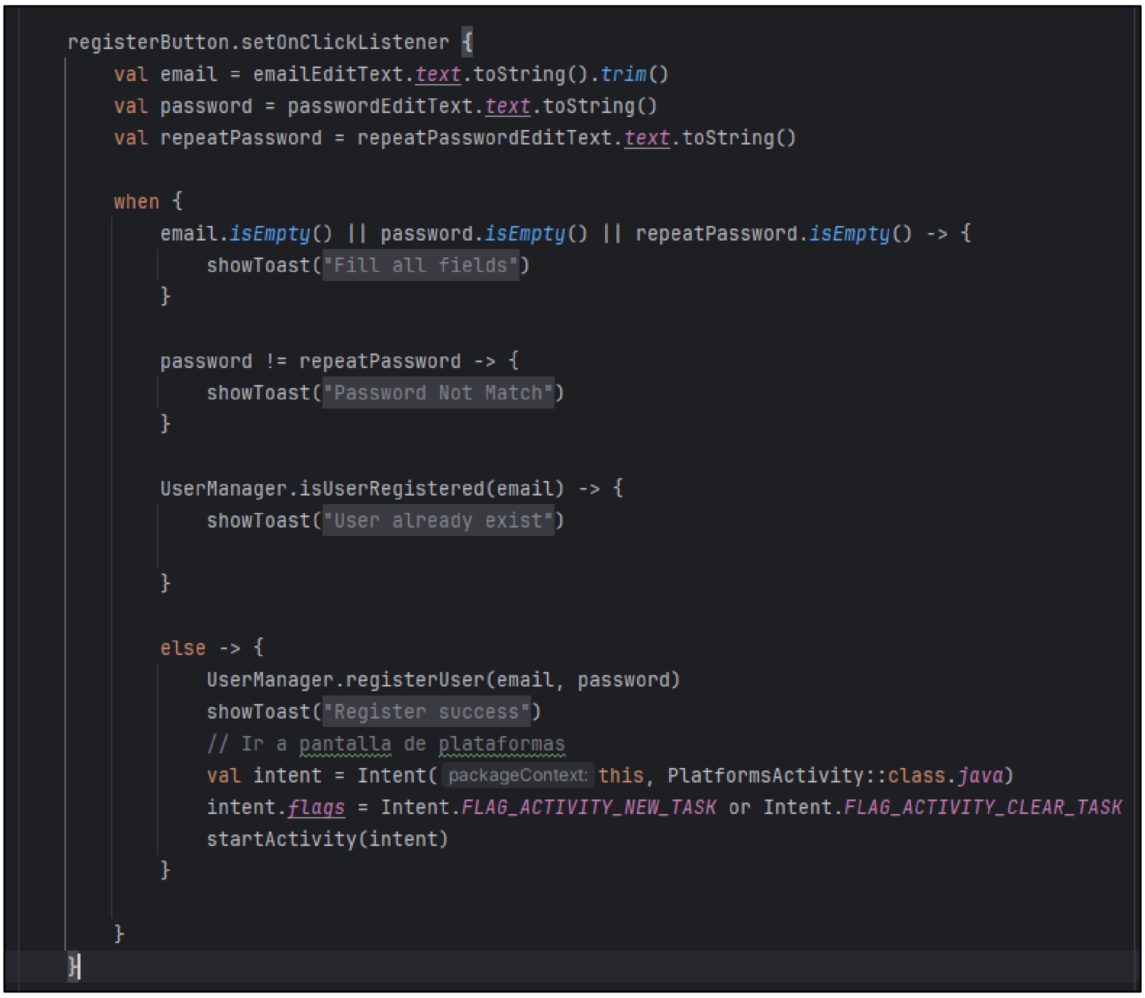
- **Login button**, makes use of the UserManager class to verify that the username and password are correct using the **validataLogin method** and throws a toast message with the result:



```
//Login button
val logBtn = findViewById<Button>(R.id.loginButton)
val edtMail = findViewById<EditText>(R.id.emailEditText)
val edtPassword = findViewById<EditText>(R.id.passwordEditText)
logBtn.setOnClickListener{
    val user = edtMail.text.toString().trim()
    val password = edtPassword.text.toString().trim()
    val intent = Intent( packageContext: this, PlatformsActivity::class.java)

    if (UserManager.validateLogin(user,password)) {
        Toast.makeText( context: this, getString(R.string.login_success), Toast.LENGTH_SHORT).show()
        startActivity(intent)
        finish()
    }else{
        Toast.makeText( context: this, getString(R.string.login_error), Toast.LENGTH_SHORT).show()
    }
}
```

- **Registration button**, just like the login button makes use of UserManager, in this case you must check that the user is not already registered, that there are no missing fields to fill in and that the passwords match:



```
registerButton.setOnClickListener {
    val email = emailEditText.text.toString().trim()
    val password = passwordEditText.text.toString()
    val repeatPassword = repeatPasswordEditText.text.toString()

    when {
        email.isEmpty() || password.isEmpty() || repeatPassword.isEmpty() -> {
            showToast("Fill all fields")
        }

        password != repeatPassword -> {
            showToast("Password Not Match")
        }

        UserManager.isUserRegistered(email) -> {
            showToast("User already exist")

        }

        else -> {
            UserManager.registerUser(email, password)
            showToast("Register success")
            // Ir a pantalla de plataformas
            val intent = Intent( packageContext: this, PlatformsActivity::class.java)
            intent.flags = Intent.FLAG_ACTIVITY_NEW_TASK or Intent.FLAG_ACTIVITY_CLEAR_TASK
            startActivity(intent)
        }

    }
}
```