

# ***Manual de funcionalidad App “Energy Smart”***

**Sección:** D-B50-N4-P12-C1

**Docente:** Cristofher Andrés Rojas Rojas

**Integrantes:** Cristian Muñoz, Bastián Flores, Marcelo Oses

**Repositorio GitHub:** [ev2IoTFinal](#)

- **Elementos Importantes:**

- **Función “Try-Catch”:** Los "try-catch" se utilizan para manejar excepciones y errores que pueden ocurrir durante la ejecución del código. Esto permite evitar que la aplicación se cierre abruptamente, es común registrar las excepciones usando `Log.e("Error", e.message.toString())` para facilitar el diagnóstico de problemas en la aplicación.

```
btnActualizarUsuario.setOnClickListener {
    try {
        val usuarioAnterior = inputUsuarioAnterior.text.toString()
        val nuevoUsuario = inputNuevoUsuario.text.toString()

        val sharedPreferences = getSharedPreferences("guardado", Context.MODE_PRIVATE)
        val usuarioGuardado = sharedPreferences.getString("usuario", "")

        if (usuarioAnterior == usuarioGuardado) {
            val editor = sharedPreferences.edit()
            editor.putString("usuario", nuevoUsuario)
            editor.apply()
            Toast.makeText(context: this, text: "Usuario actualizado con éxito", Toast.LENGTH_SHORT).show()
        } else {
            Toast.makeText(context: this, text: "Usuario anterior incorrecto", Toast.LENGTH_SHORT).show()
        }
    } catch (e: Exception) {
        Log.e(tag: "Error", e.message.toString())
    }
}
```

*Estructura de una función “try-catch”*

- **Navegación entre vistas/pantallas:** Dentro del "OnClickListener", se crea un "Intent" que especifica la actividad de destino, en este caso, "OpcionesCuenta". Este "Intent" es el medio que permite iniciar la nueva actividad. Al llamar a "startActivity(intent)", la aplicación cambia a la actividad "OpcionesCuenta", permitiendo al usuario acceder a las opciones relacionadas con su cuenta.

```
val irOpciones: Button = findViewById(R.id.btnOpcionesCuenta)
irOpciones.setOnClickListener {
    val intent = Intent(packageContext: this, OpcionesCuenta::class.java)
    startActivity(intent)
}
```

- ***Vista 1: Inicio de sesión***



**Descripción:** Pantalla de inicio de sesión que cuenta con los campos para ingresar los datos de nombre de usuario y contraseña junto al botón para verificar los datos e ingresar al sistema, adicionalmente cuenta con los botones para acceder a la pantalla de registro y al menú de gestión de cuentas.

### ***Funciones clave en archivo MainActivity.kt:***

#### **1\_fn “irDashboard”**

```
val irDashboard: Button = findViewById(R.id.botoniniciar)
irDashboard.setOnClickListener {
    try {
        val usuario: String = findViewById<EditText>(R.id.input_usuario).text.toString()
        val contraseña: String =
            findViewById<EditText>(R.id.input_contraseña).text.toString()
        if (usuario.isNotEmpty() && contraseña.isNotEmpty()) {
            val compartirprefe = getSharedPreferences( name: "guardado", Context.MODE_PRIVATE)
            val guardarusuario = compartirprefe.getString("usuario", null)
            val guardarcontra = compartirprefe.getString("contraseña", null)
            if (usuario == guardarusuario && contraseña == guardarcontra) {
                val intent = Intent( packageContext: this, Dashboard::class.java)
                startActivity(intent)
            } else {
                Toast.makeText( context: this, text: "Datos incorrectos", Toast.LENGTH_SHORT).show()
            }
        } else {
            if (usuario.isEmpty()) {
                Toast.makeText( context: this, text: "No has ingresado un usuario", Toast.LENGTH_SHORT)
                    .show()
            }
            if (contraseña.isEmpty()) {
                Toast.makeText( context: this, text: "No has ingresado una contraseña", Toast.LENGTH_SHORT)
                    .show()
            }
        }
    }
} catch (e: Exception){
    Log.e( tag: "Error", e.message.toString())
}
```

**Descripción:** Este bloque de código gestiona el evento de clic en el botón de inicio de sesión, permitiendo a los usuarios ingresar su nombre de usuario y contraseña para acceder a la aplicación.

Al hacer clic en el botón, se recuperan los valores ingresados en los campos de texto para el nombre de usuario y la contraseña. A continuación, se verifica que ambos campos no estén vacíos.

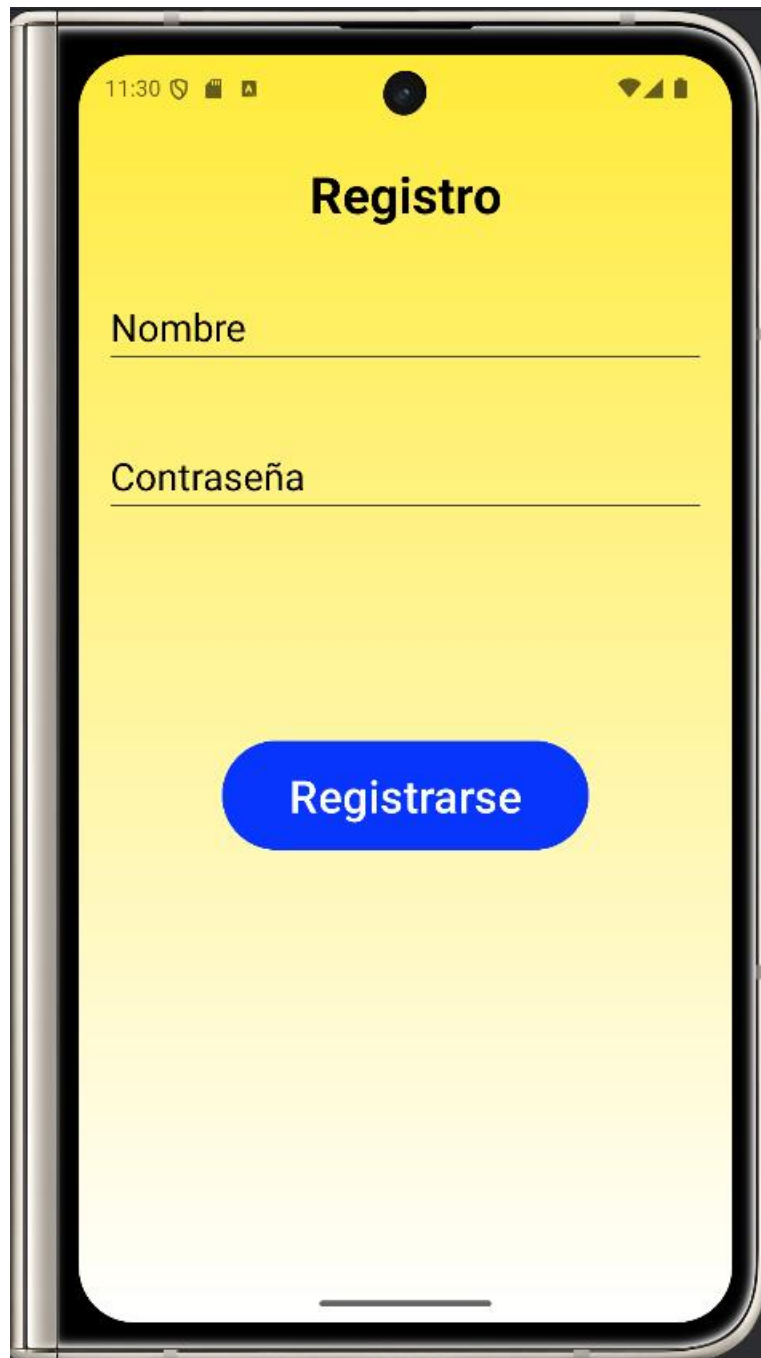
Si los campos están llenos, se accede a “SharedPreferences” para recuperar el nombre de usuario y la contraseña almacenados previamente. Estos datos se comparan con las credenciales ingresadas por el usuario.

Si las credenciales coinciden, se crea un “Intent” para iniciar la actividad “Dashboard”, permitiendo al usuario acceder a las funciones de la aplicación. En caso de que las credenciales no coincidan, se muestra un mensaje de error mediante un “Toast”, informando al usuario que los datos son incorrectos.

Si alguno de los campos está vacío, se proporciona un mensaje de advertencia específico para cada campo vacío, indicando que el usuario debe completar los datos requeridos.

Además, se incluye un bloque “try-catch” para manejar cualquier excepción que pueda ocurrir durante el proceso, registrando el error si se presenta uno.

- ***Vista 2: Registro***



11:30

**Registro**

Nombre

Contraseña

**Registrarse**

**Descripción:** Pantalla de registro, que cuenta con los campos para ingresar los datos de nombre de usuario y contraseña y el botón “Registrarse” que ejecuta la acción de lectura, validación y guardado de datos para posteriormente dirigir al usuario a la pantalla principal.

## ***Funciones clave en archivo Registrar.kt:***

### **1\_fn “irAlInicio”**

```
val irAlInicio: Button = findViewById(R.id.botonderegistro)
irAlInicio.setOnClickListener {
    try {
        val nombre: String = findViewById<EditText>(R.id.input_name).text.toString()
        val contrasena: String = findViewById<EditText>(R.id.input_password).text.toString()

        if (nombre.isNotEmpty() && contrasena.isNotEmpty()) {
            val compartirprefe = getSharedPreferences( name: "guardado", Context.MODE_PRIVATE)
            val editor = compartirprefe.edit()
            editor.putString("usuario", nombre)
            editor.putString("contrasena", contrasena)
            editor.apply()

            val intent = Intent( packageContext: this, MainActivity::class.java)
            startActivity(intent)
        } else {
            if (nombre.isEmpty()) {
                Toast.makeText( context: this, text: "No has ingresado un nombre", Toast.LENGTH_SHORT).show()
            }
            if (contrasena.isEmpty()) {
                Toast.makeText( context: this, text: "No has ingresado una contraseña", Toast.LENGTH_SHORT).show()
            }
        }
    }
} catch (e: Exception) {
    Log.e( tag: "Error", e.message.toString())
}
}
```

**Descripción:** Este código gestiona el evento de clic en el botón de registro, permitiendo a los usuarios ingresar su nombre y contraseña para crear una cuenta. Al hacer clic en el botón, se recuperan los valores ingresados en los campos de texto para el nombre y la contraseña.

Se verifica que ambos campos no estén vacíos. Si están completos, se accede a “SharedPreferences” para guardar el nombre de usuario y la contraseña,

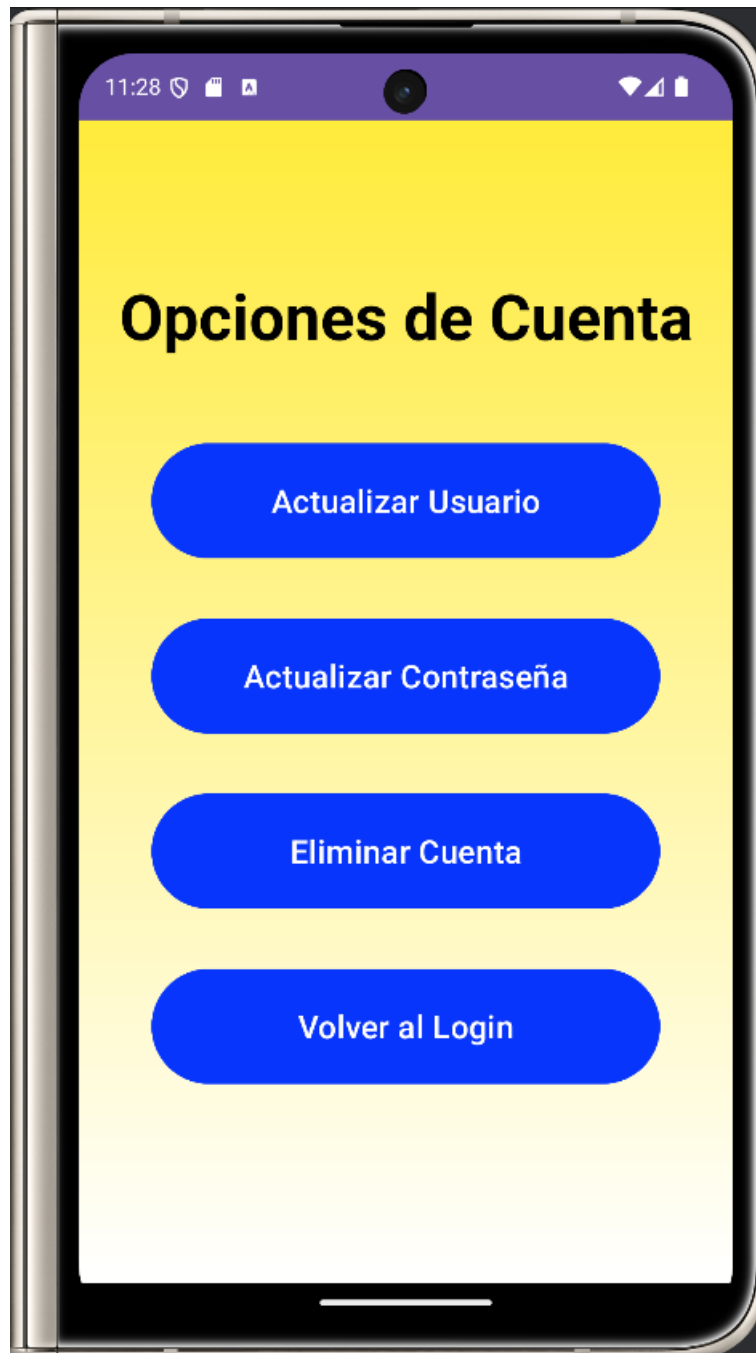
utilizando el nombre "guardado" como referencia. Esto permite que las credenciales sean persistentes y accesibles en futuros inicios de sesión.

Después de almacenar los datos, se crea un "Intent" que inicia la actividad "MainActivity", llevando al usuario a la pantalla de inicio de sesión. Si alguno de los campos está vacío, se muestra un mensaje de error informando al usuario que debe completar el nombre y/o la contraseña.

Además, se incluye un bloque "try-catch" para manejar cualquier excepción que pueda ocurrir durante el proceso, registrando el error si se presenta uno.

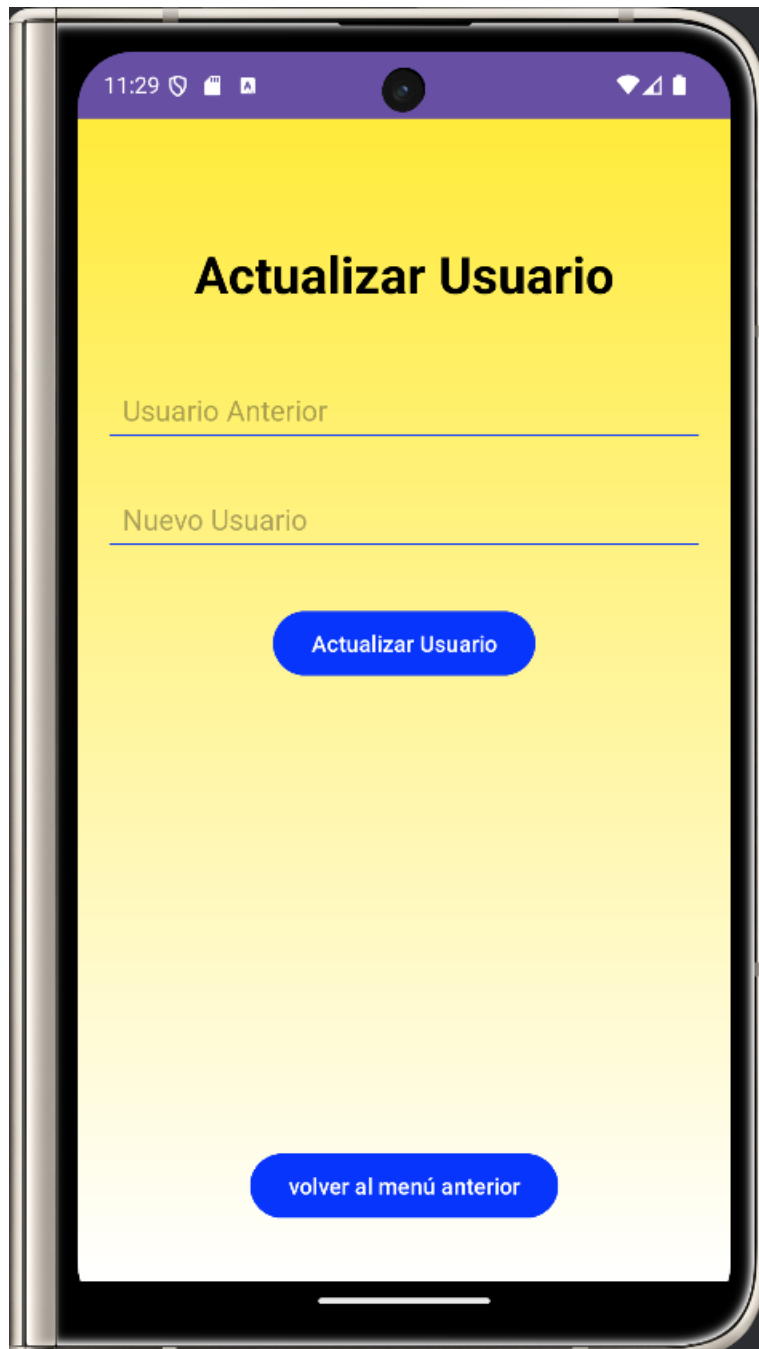


- ***Vista 3: Gestión de usuarios***



**Descripción:** la vista solo cuenta con botones que permiten la navegación entre la misma y las paginas con las que interactúa, las cuales se describen a continuación.

- ***Vista 4: Actualizar usuario***



11:29

## Actualizar Usuario

Usuario Anterior

Nuevo Usuario

Actualizar Usuario

volver al menú anterior

**Descripción:** pantalla de actualización de nombre de usuario, cuenta con los campos para ingresar el antiguo nombre de usuario y el nombre por el que se quiere reemplazar, además se incluyen el botón “actualizar usuario” el cual inicia la función que lee, valida y guarda los cambios.

## ***Funciones clave en archivo ActualizarUsuario.kt:***

### **1\_fn “btnActualizarUsuario”**

```
val inputUsuarioAnterior = findViewById<EditText>(R.id.inputUsuarioAnterior)
val inputNuevoUsuario = findViewById<EditText>(R.id.inputNuevoUsuario)
val btnActualizarUsuario = findViewById<Button>(R.id.btnActualizarUsuario)

btnActualizarUsuario.setOnClickListener {
    try {
        val usuarioAnterior = inputUsuarioAnterior.text.toString()
        val nuevoUsuario = inputNuevoUsuario.text.toString()

        val sharedPreferences = getSharedPreferences("guardado", Context.MODE_PRIVATE)
        val usuarioGuardado = sharedPreferences.getString("usuario", "")

        if (usuarioAnterior == usuarioGuardado) {
            val editor = sharedPreferences.edit()
            editor.putString("usuario", nuevoUsuario)
            editor.apply()
            Toast.makeText(context: this, text: "Usuario actualizado con éxito", Toast.LENGTH_SHORT).show()
        } else {
            Toast.makeText(context: this, text: "Usuario anterior incorrecto", Toast.LENGTH_SHORT).show()
        }
    } catch (e: Exception) {
        Log.e(tag: "Error", e.message.toString())
    }
}
```

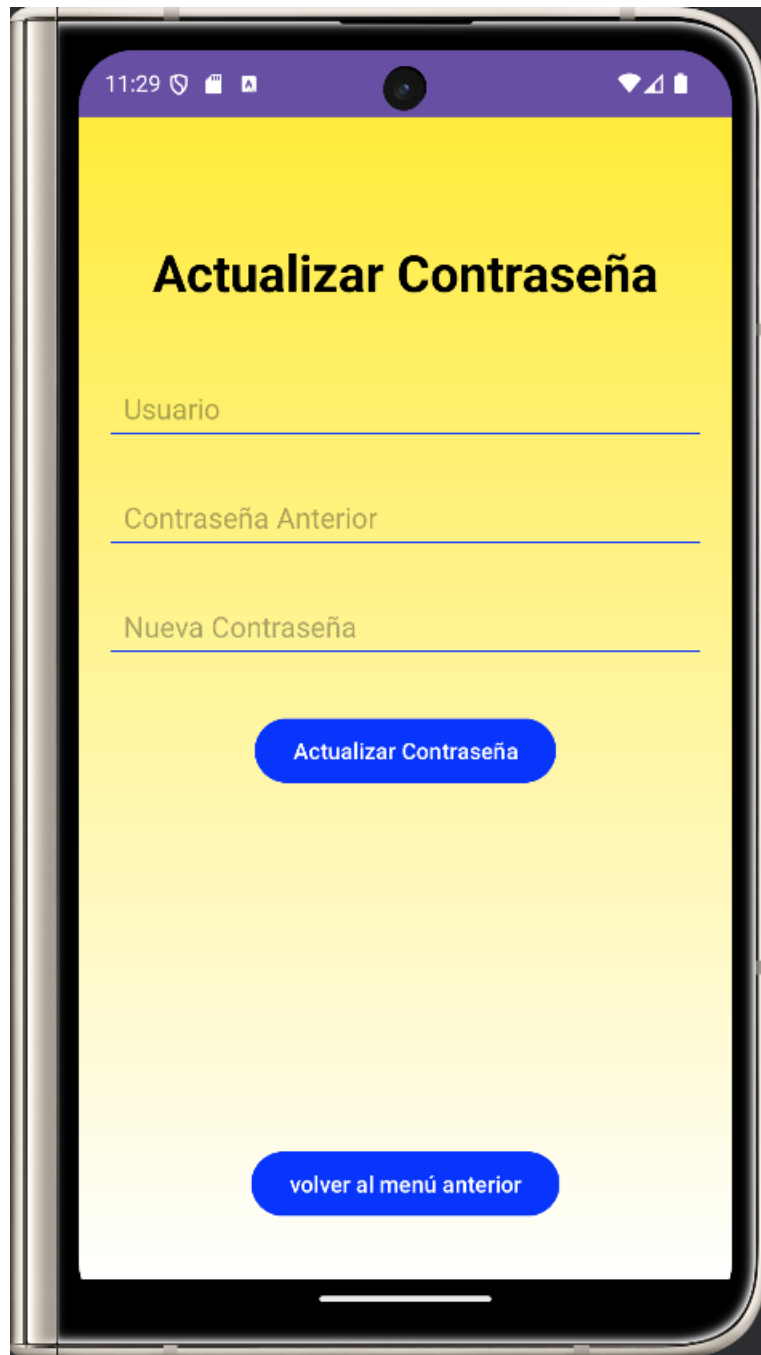
**Descripción:** Este bloque de código gestiona la actualización del nombre de usuario. Primero, se obtienen las referencias a los campos de entrada necesarios: el nombre de usuario anterior y el nuevo nombre de usuario, así como al botón para confirmar la actualización.

Cuando el usuario hace clic en el botón para actualizar el nombre de usuario, se recuperan los valores ingresados en los campos de texto. A continuación, se accede a “SharedPreferences” para obtener el nombre de usuario almacenado previamente.

Se verifica si el nombre de usuario anterior ingresado coincide con el valor guardado. Si coinciden, se procede a actualizar el nombre de usuario almacenado con el nuevo valor ingresado. Esto se realiza utilizando el editor de “SharedPreferences”, que permite guardar los cambios de forma persistente. Una vez que se actualiza el nombre, se muestra un mensaje de éxito al usuario mediante un “Toast”.

Si el nombre de usuario anterior no coincide con el guardado, se muestra un mensaje de error informando al usuario que el nombre ingresado es incorrecto. Además, se incluye un bloque “try-catch” para manejar cualquier excepción que pueda ocurrir durante el proceso, registrando el error si se presenta uno.

- ***Vista 5: Actualizar contraseña***



The image shows a smartphone screen with a yellow background. At the top, there is a purple status bar with the time 11:29 and various icons. The main title 'Actualizar Contraseña' is centered in bold black text. Below the title are three input fields with labels 'Usuario', 'Contraseña Anterior', and 'Nueva Contraseña'. Each field has a blue underline. Below the input fields are two blue buttons with white text: 'Actualizar Contraseña' and 'volver al menú anterior'.

11:29

## Actualizar Contraseña

Usuario

Contraseña Anterior

Nueva Contraseña

Actualizar Contraseña

volver al menú anterior

**Descripción:** Pantalla de actualización de contraseña, cuenta con los campos para ingresar los datos de nombre de usuario, contraseña anterior y nueva contraseña y con el botón “Actualizar contraseña” que ejecuta la función de leer los datos, validarlos y guardar los cambios.

## ***Funciones clave en archivo ActualizarPassword.kt:***

### **1\_fn “btnActualizarPassword”**

```
val inputUsuario = findViewById<EditText>(R.id.inputUsuarioPassword)
val inputPasswordAnterior = findViewById<EditText>(R.id.inputPasswordAnterior)
val inputNuevaPassword = findViewById<EditText>(R.id.inputNuevaPassword)
val btnActualizarPassword = findViewById<Button>(R.id.btnActualizarPassword)

val btnVolverLogin = findViewById<Button>(R.id.btnVolverLoginPassword) //función para el btn regresar al menu anterior

btnActualizarPassword.setOnClickListener {
    try{
        val usuario = inputUsuario.text.toString()
        val passwordAnterior = inputPasswordAnterior.text.toString()
        val nuevaPassword = inputNuevaPassword.text.toString()

        val sharedPreferences = getSharedPreferences( name: "guardado", Context.MODE_PRIVATE)
        val usuarioGuardado = sharedPreferences.getString("usuario", "")
        val contrasenaGuardada = sharedPreferences.getString("contrasena", "")

        if (usuario == usuarioGuardado && passwordAnterior == contrasenaGuardada) {
            val editor = sharedPreferences.edit()
            editor.putString("contrasena", nuevaPassword)
            editor.apply()
            Toast.makeText( context: this, text: "Contraseña actualizada con éxito", Toast.LENGTH_SHORT).show()

            finish()
        } else {
            Toast.makeText( context: this, text: "Usuario o contraseña anterior incorrectos", Toast.LENGTH_SHORT).show()
        }catch (e:Exception){
            Log.e( tag: "Error", e.message.toString())
        }
    }
}
```

**Descripción:** Este código gestiona la actualización de la contraseña del usuario. Primero, se obtienen las referencias a los campos de entrada necesarios: el nombre de usuario, la contraseña anterior y la nueva contraseña, así como al botón para actualizar la contraseña.

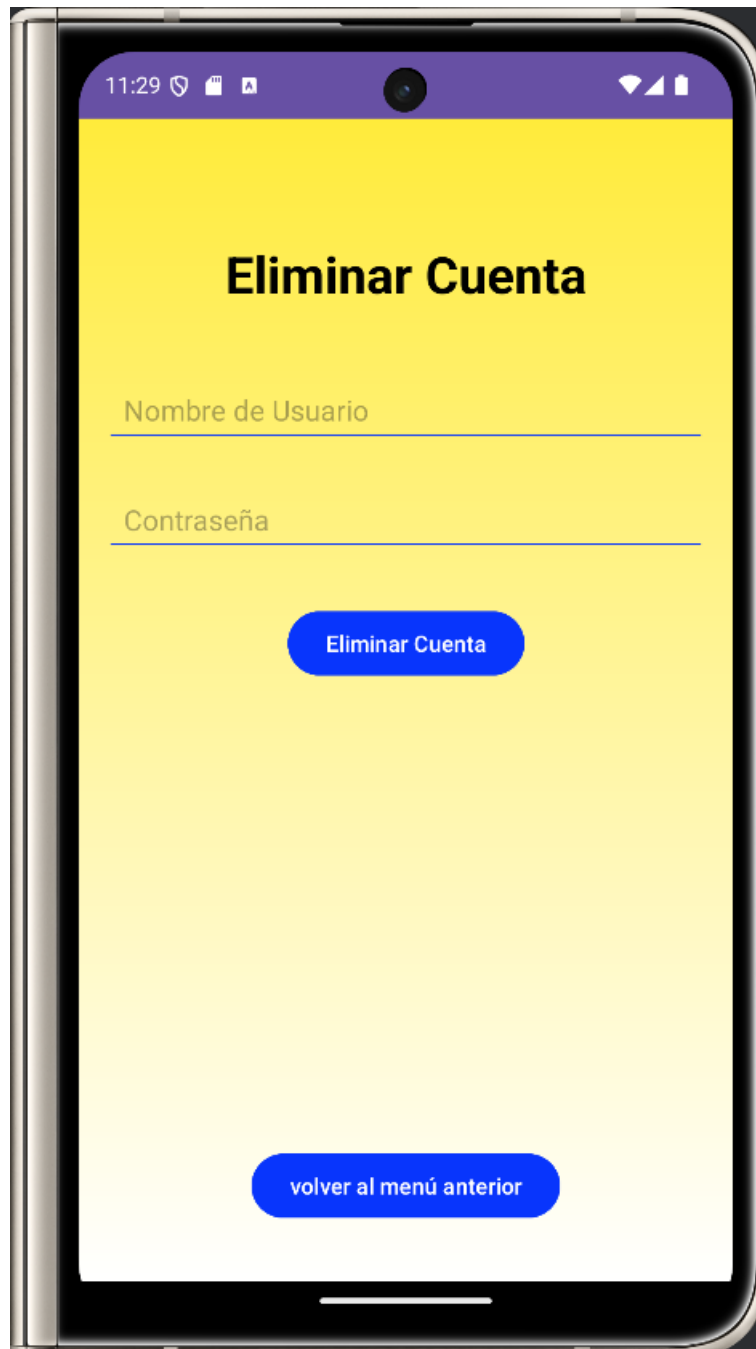
Se define un segundo botón que permite al usuario regresar al menú anterior, aunque la función que ejecuta este botón no está relacionada con el cambio de contraseña.

Al hacer clic en el botón para actualizar la contraseña, se recuperan los valores ingresados en los campos de texto. A continuación, se accede a “SharedPreferences” para obtener el nombre de usuario y la contraseña almacenados previamente.

Se realiza una verificación para comprobar si el nombre de usuario y la contraseña anterior ingresados coinciden con los valores guardados. Si coinciden, se procede a actualizar la contraseña almacenada con la nueva contraseña ingresada, utilizando el editor de “SharedPreferences” para guardar el cambio. Se muestra un mensaje de éxito al usuario mediante un “Toast”.

Si el usuario o la contraseña anterior son incorrectos, se informa al usuario con un mensaje de error. Además, se incluye un bloque “try-catch” para manejar cualquier excepción que pueda ocurrir durante el proceso, registrando el error si se presenta uno.

- ***Vista 6: Eliminar usuario***



The image shows a smartphone screen with a yellow background. At the top, there is a purple status bar with the time 11:29 and various icons. The main title 'Eliminar Cuenta' is centered in bold black text. Below the title are two input fields: 'Nombre de Usuario' and 'Contraseña', both with blue underlines. A blue button labeled 'Eliminar Cuenta' is centered below the input fields. At the bottom, there is another blue button labeled 'volver al menú anterior'.

11:29

## Eliminar Cuenta

Nombre de Usuario

Contraseña

Eliminar Cuenta

volver al menú anterior



**Descripción:** Pantalla de eliminación de cuenta, cuenta con los campos para ingresar los datos de nombre de usuario y contraseña y con el botón “Eliminar cuenta” que ejecuta la función de leer los datos, validarlos y eliminar la cuenta.

## ***Funciones clave en archivo EliminarCuenta.kt:***

### **1\_fn “btnConfirmarEliminarCuenta”**

```
val inputUsuarioEliminar = findViewById<EditText>(R.id.inputUsuarioEliminar)
val inputPasswordEliminar = findViewById<EditText>(R.id.inputPasswordEliminar)
val btnConfirmarEliminarCuenta = findViewById<Button>(R.id.btnConfirmarEliminarCuenta)

val btnVolverLoginEliminar = findViewById<Button>(R.id.btnVolverLoginEliminar)

btnConfirmarEliminarCuenta.setOnClickListener {
    try {

        val usuarioEliminar = inputUsuarioEliminar.text.toString()
        val passwordEliminar = inputPasswordEliminar.text.toString()

        val sharedPreferences = getSharedPreferences( name: "guardado", Context.MODE_PRIVATE)
        val usuarioGuardado = sharedPreferences.getString("usuario", "")
        val contrasenaGuardada = sharedPreferences.getString("contrasena", "")

        if (usuarioEliminar == usuarioGuardado && passwordEliminar == contrasenaGuardada) {
            // Eliminar los datos de la cuenta
            val editor = sharedPreferences.edit()
            editor.clear() // Elimina todas las preferencias
            editor.apply()

            Toast.makeText( context: this, text: "Cuenta eliminada con éxito", Toast.LENGTH_SHORT).show()
            // Regresar a la pantalla de inicio de sesión
            finish() // Cierra la actividad actual
        } else {
            Toast.makeText( context: this, text: "Usuario o contraseña incorrectos", Toast.LENGTH_SHORT).show()
        }
    } catch (e: Exception){
        Log.e( tag: "Error", e.message.toString())
    }
}
```

**Descripción:** Este código gestiona la eliminación de la cuenta de un usuario. Primero, se obtienen las referencias a los campos de entrada necesarios: el nombre de usuario y la contraseña del usuario que desea eliminar su cuenta, así como al botón para confirmar la eliminación.

Se define un segundo botón que permite al usuario regresar al menú anterior, aunque la función que ejecuta este botón no está relacionada directamente con la eliminación de la cuenta.

Al hacer clic en el botón para confirmar la eliminación de la cuenta, se recuperan los valores ingresados en los campos de texto. A continuación, se accede a “SharedPreferences” para obtener el nombre de usuario y la contraseña almacenados previamente.

Se realiza una verificación para comprobar si el nombre de usuario y la contraseña ingresados coinciden con los valores guardados. Si coinciden, se procede a eliminar los datos de la cuenta utilizando el editor de “SharedPreferences”. El método “clear()” se invoca para eliminar todas las preferencias almacenadas, y se muestra un mensaje de éxito al usuario mediante un “Toast”.

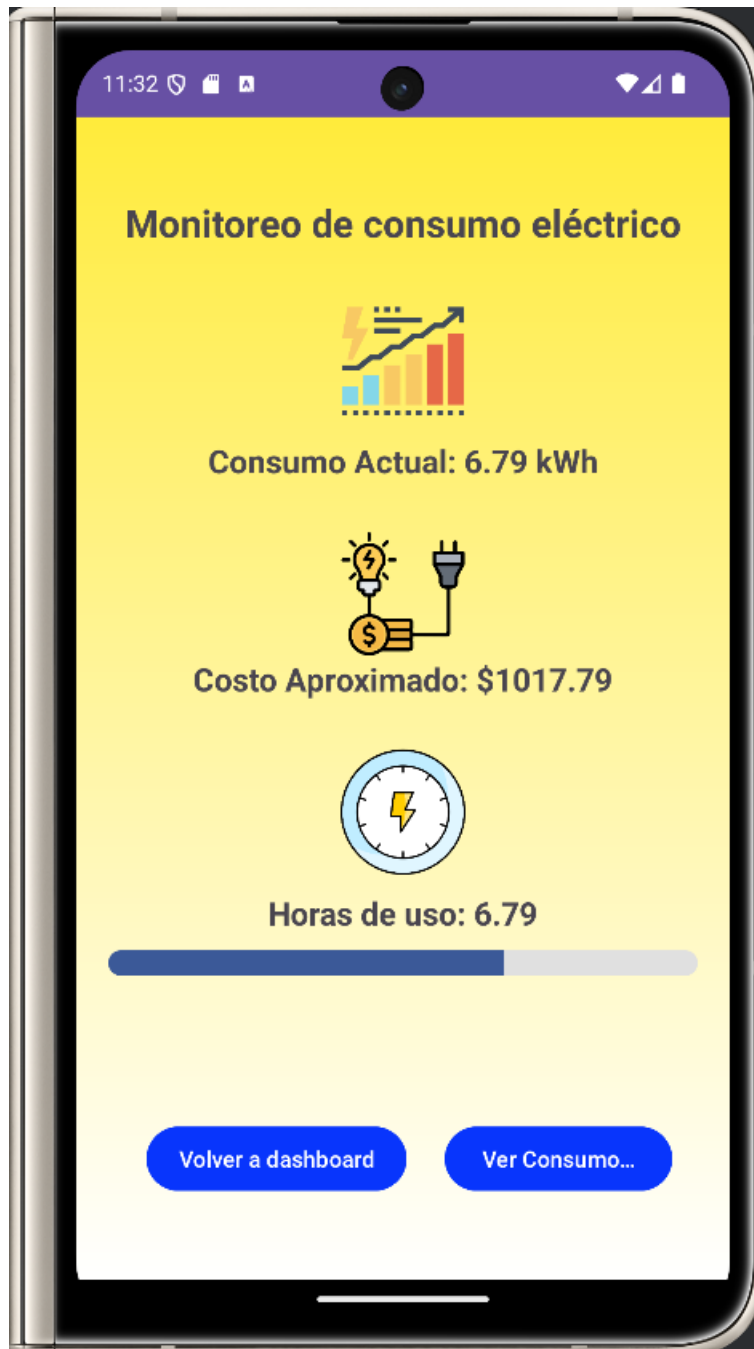
Si el usuario o la contraseña son incorrectos, se informa al usuario con un mensaje de error. Además, se incluye un bloque “try-catch” para manejar cualquier excepción que pueda ocurrir durante el proceso, registrando el error si se presenta uno. Finalmente, se cierra la actividad actual para regresar a la pantalla de inicio de sesión.

- ***Vista 7: Dashboard***



**Descripción:** la vista solo cuenta con botones que permiten la navegación entre la misma y las paginas con las que interactúa.

- ***Vista 8: Monitoreo de consumo eléctrico***



**Descripción:** Pantalla de monitoreo de consumo eléctrico, muestra los datos del consumo actual en kWh, el coste monetario que en que se traduce el gasto y las horas de uso que han derivado en dicho gasto.

## *Variables, funciones y obtenciones de id:*

```
class consumo : AppCompatActivity() {  
    private var consumoActual = 0.0f  
    private var consumoTextView: TextView? = null  
    private var costoTextView: TextView? = null  
    private var horasUsoTextView: TextView? = null  
    private var progressBar: ProgressBar? = null  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.consumo)  
  
        consumoTextView = findViewById(R.id.consumo_actual)  
        costoTextView = findViewById(R.id.costo_aproximado)  
        horasUsoTextView = findViewById(R.id.horas_uso)  
        progressBar = findViewById(R.id.progressBar)  
  
        val btnConsumo = findViewById<Button>(R.id.btnConsumo)  
        btnConsumo.setOnClickListener {  
            mostrarConsumo()  
        }  
  
        val btnHome = findViewById<Button>(R.id.btnHome)  
        btnHome.setOnClickListener {  
            finish()  
        }  
    }  
}
```

## ***Funciones clave en archivo consumo.kt:***

### **1\_fn “mostrarConsumo”**

```
private fun mostrarConsumo() {  
    consumoActual = (Math.random() * 10 + 1).toFloat()  
    consumoTextView!!.text = "Consumo Actual: " + String.format("%.2f", consumoActual) + " kWh"  
  
    val costoAproximado = calcularCosto(consumoActual)  
    costoTextView!!.text = "Costo Aproximado: $" + String.format("%.2f", costoAproximado)  
  
    val progreso = (consumoActual / 10 * 100).toInt()  
    progressBar!!.progress = progreso  
  
    mostrarMensajeConsumo(progreso)  
  
    calcularHorasUso()  
}
```

**Descripción:** La función “mostrarConsumo()” se encarga de actualizar los elementos de la interfaz relacionados con el consumo de energía. En esta función, se genera un valor aleatorio para el consumo actual de energía, que varía entre 1 y 10 kWh, utilizando “Math.random()”. Este valor se convierte a un “Float” y se muestra en un “TextView” como "Consumo Actual: X kWh", formateado a dos decimales.

Luego, se calcula el porcentaje de progreso basado en el consumo actual, donde el consumo máximo posible se establece en 10 kWh. Este porcentaje se utiliza para actualizar una “ProgressBar”, proporcionando una representación visual del consumo. Además, se llama a la función “mostrarMensajeConsumo(progreso)” para proporcionar retroalimentación sobre el nivel de consumo y se invoca “calcularHorasUso()” para calcular el tiempo de uso de los dispositivos.

## 2\_fn “mostrarMensajeConsumo”

```
private fun mostrarMensajeConsumo(progreso: Int) {  
    when {  
        progreso > 80 → {  
            Toast.makeText(context: this, text: "Consumo Alto", Toast.LENGTH_SHORT).show()  
        }  
        progreso > 50 → {  
            Toast.makeText(context: this, text: "Consumo Moderado", Toast.LENGTH_SHORT).show()  
        }  
        else → {  
            Toast.makeText(context: this, text: "Consumo Bajo", Toast.LENGTH_SHORT).show()  
        }  
    }  
}
```

**Descripción:** La función “mostrarMensajeConsumo(progreso: Int)” proporciona retroalimentación al usuario sobre el nivel de consumo energético. Utiliza una estructura “when” para evaluar el porcentaje de progreso. Si el progreso es mayor al 80%, se muestra un mensaje de "Consumo Alto". Si el progreso es mayor al 50%, se indica "Consumo Moderado". De lo contrario, se muestra un mensaje de "Consumo Bajo". Cada mensaje se presenta mediante un “Toast”, asegurando que el usuario reciba una notificación clara y breve sobre su uso de energía.

## 3\_fn “calcularCosto”

```
private fun calcularCosto(consumo: Float): Float {  
    val tarifaPorKWh = 150.0f  
    return consumo * tarifaPorKWh  
}
```

**Descripción:** La función “calcularCosto(consumo: Float)” calcula el costo asociado al consumo de energía. Se define una tarifa fija de 150.0 unidades monetarias por kWh. El costo se calcula multiplicando el consumo por esta tarifa, y se devuelve el valor resultante.

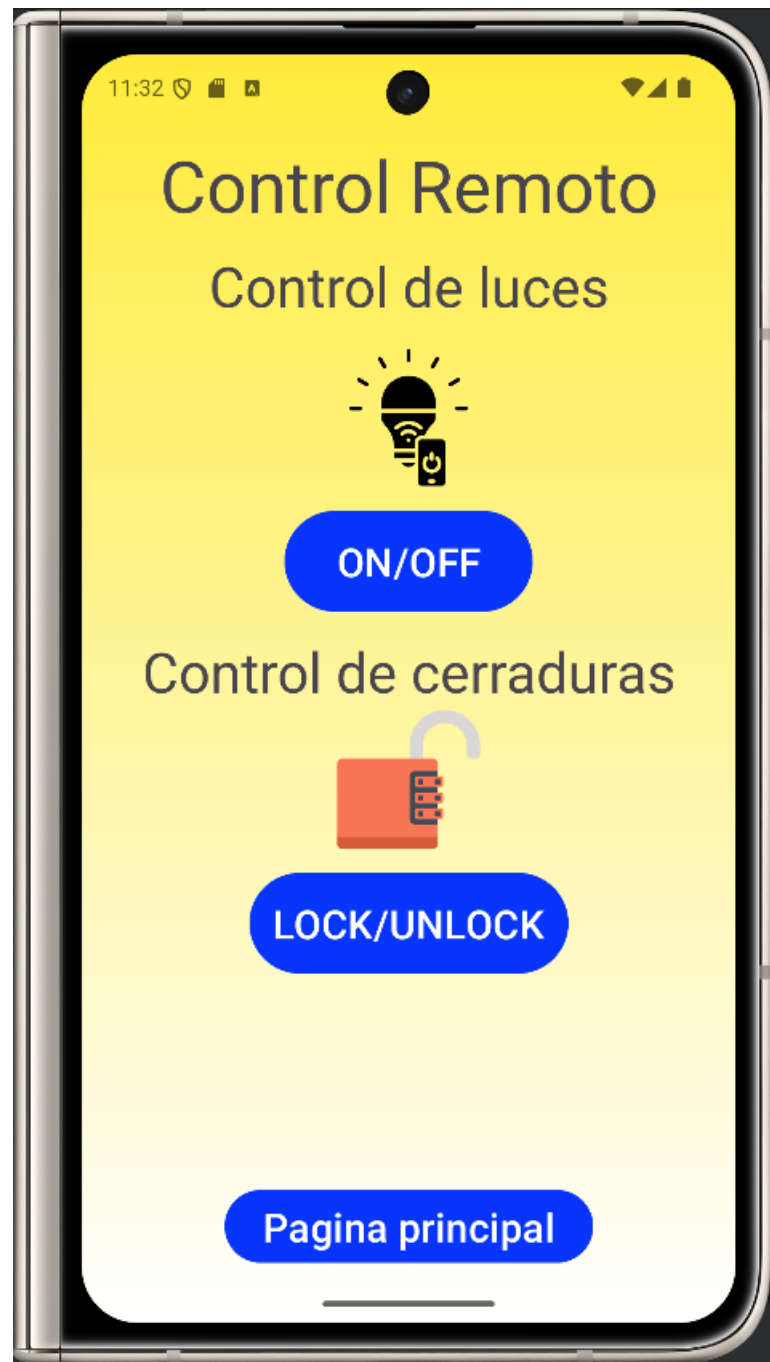
## 4\_fn “calcularHorasUso”

```
private fun calcularHorasUso() {  
    horasUsoTextView!!.text = "Horas de uso: " + String.format("%.2f", consumoActual)  
}
```

Descripción: la función “calcularHorasUso()” actualiza un “TextView” correspondiente a las horas de uso, mostrando el consumo actual como "Horas de uso: X". Esto proporciona al usuario información adicional sobre su consumo energético en términos de tiempo.



- ***Vista 9: Control de automatización Domótica***



**Descripción:** Pantalla de control remoto, cuenta con dos subsecciones principales que corresponden al control de las luces y al control de cerraduras, cada una se compone por una imagen o icono que representa el estado actual de dicho modulo junto un botón que se usa para encender o apagar el mismo dependiendo de su estado actual.

## ***Funciones clave en archivo controles.kt:***

### **1\_fn “fnOnOff”**

```
private fun fnOnOff() {

    try {
        //Toast.makeText(this,"Valor: $luzXdefecto",Toast.LENGTH_SHORT).show()
        val lgtState: ImageView= findViewById(R.id.imgLuz)
        val lgtMsg: Button= findViewById(R.id.btnLuces)

        when(luzXdefecto){
            0→{
                lgtState.setImageResource(R.drawable.lighton)
                lgtMsg.text = "On"
            }
            1→{
                lgtState.setImageResource(R.drawable.lightoff)
                lgtMsg.text = "Off"
            }
        }
        if (luzXdefecto=0){
            luzXdefecto=1
        }else{
            luzXdefecto=0
        }
    }catch (e:Exception){
        Log.e( tag: "Error", e.message.toString())
    }
}
```

**Descripción:** La función “fnOnOff()” se encarga de controlar el estado de las luces en la aplicación, permitiendo al usuario encender y apagar las luces de manera interactiva. Al ejecutarse, la función primero intenta acceder a los elementos de la interfaz de usuario necesarios: un “ImageView” que representa el estado de la luz y un “Button” que permite al usuario activar o desactivar las luces.

Utilizando una estructura “when”, la función verifica el estado actual de la variable “luzXdefecto”, que determina si la luz está encendida (valor 0) o apagada (valor 1). Si la luz está apagada, se actualiza la imagen del “ImageView” para mostrar una luz encendida (usando “R.drawable.lighton”) y se cambia el texto del botón a "On". Si la luz está encendida, se muestra una luz apagada (usando “R.drawable.lightoff”) y el texto del botón cambia a "Off".

Después de actualizar la interfaz, la función alterna el estado de “luzXdefecto”. Si la luz estaba apagada, se establece en 1 (encendida), y si estaba encendida, se cambia a 0 (apagada). Esto permite que cada clic en el botón cambie el estado de las luces de manera efectiva.

Es importante destacar que el control de los cerrojos sigue un funcionamiento prácticamente idéntico. La lógica de alternancia y la actualización de la interfaz serían similares, permitiendo al usuario gestionar tanto las luces como los cerrojos de forma coherente y sencilla.

- **Referencias:**

- Keywords and operators | Kotlin. (s. f.). Recuperado de <https://kotlinlang.org/docs/keyword-reference.html>
- GeeksforGeeks. (2024, 5 abril). Kotlin functions. Recuperado de <https://www.geeksforgeeks.org/kotlin-functions/>
- Aris. (2023, 15 enero). Capítulo 6 – Funciones en Kotlin. Recuperado de <https://cursokotlin.com/capitulo-6-funciones-kotlin/>
- Diseños en vistas. (s. f.). Recuperado de <https://developer.android.com/develop/ui/views/layout/declaring-layout?hl=es-419>
- Bustos, J. L. (2024, 24 abril). Vistas en Android: ¿qué son y cómo funcionan? Recuperado de <https://keepcoding.io/blog/vistas-en-android-que-son-y-como-funcionan/>
- Luis, J. (2023, 13 octubre). Crear un Botón y Lanzar una Nueva Actividad en Kotlin. Recuperado de <https://codea.app/blog/crear-un-boton-y-lanzar-una-nueva-actividad-en-kotlin>