INSTITUTO PROFESIONAL SANTO TOMÁS

SEDE CHILLÁN

Evaluación Final Unidad II

Nombre Alumno : Macarena Antonia Leiva Sobarzo
Nombre Profesor : Manuel Moisés Merino Piceros
Nombre Asignatura : Programación Android
Nombre Carrera : Ingeniería en Informática

05/11/2024

Índice

Introducción

Contexto:

Este proyecto consiste en el desarrollo de una aplicación móvil que facilita a los usuarios la realización de encargos sin requerir el pago inmediato a través de la plataforma. La app permite a los clientes seleccionar productos y realizar pedidos, pero el pago se efectúa únicamente en el momento en que el usuario recoge su pedido en un punto de venta físico. Esto garantiza un plazo de hasta 3 horas desde la solicitud para realizar el retiro y pago.

Para que los usuarios tengan acceso a los productos disponibles, el sistema incluye una funcionalidad CRUD, que solo puede tener acceso el administrador.

Importancia y relevancia del proyecto:

La aplicación se basa en la necesidad practica para poder realizar pedidos con comodidad y evitar las largas filas.

El sistema CRUD permite la mejora y eficiencia del inventario, lo cual es crucial para mantener actualizada la información y garantizar que el usuario vea los productos de manera precisa.

Optimiza la interacción del cliente con el comercio, promueve un modelo accesible y competente con las preferencias del usuario.

Objetivos de la aplicación

Que problema resuelve la aplicación:

Resuelve la problemática de hacer compras de manera rápida y conveniente, sin la necesidad de hacer pagos en línea. Hay usuarios que por motivos de seguridad prefieren no pagar en línea. Este sistema les permite hacer reservas sin comprometerse económicamente, aparte evita pérdidas de tiempo y permite que el usuario reserve y recoja cuando le convenga dentro del plazo establecido.

Beneficios esperados para los usuarios:

El usuario puede hacer un pedido en cualquier momento (dentro de los horarios establecidos).

No es necesario pagar en línea, solo al momento de recoger la compra.

Los usuarios tienen acceso al catálogo actualizado de los productos.

Metas concretas que se esperan alcanzar con la aplicación:

- Facilitar la reserva de productos sin pago en línea.
- Usar el sistema CRUD para mantener los productos actualizados en tiempo real.
- Minimizar los tiempos de espera para el usuario.
- Mejorar la experiencia de compra del usuario, ofreciéndole flexibilidad en el tiempo y el método de pago.

Descripción de la aplicación

Funcionalidades clave de la aplicación:

La aplicación permite al administrador ingresar nuevos productos, detallando el nombre y la cantidad de este, e ID es auto incremental y primary key.
Puede leer los productos, actualizar los productos y también eliminar en caso de ser necesario.

Descripción del diseño y navegación de la aplicación:

La interfaz de la sección CRUD está diseñada para ser intuitiva y sencilla.
Agregar: al iniciar el proyecto, nos lleva automáticamente a la ventana donde salen los productos. Podemos ingresar los campos y luego seleccionar el botón Agregar y el producto se agrega correctamente.

Editar y eliminar: al seleccionar un producto, automáticamente salen los detalles y los botones “Guardar cambios” y “Eliminar”, en caso de querer editar, cambiamos los datos necesarios y seleccionamos el botón correspondiente. En caso de eliminar, solo seleccionamos el botón con el mismo nombre.

Breve explicación de las herramientas y tecnologías empleadas en el desarrollo:

SQLite: Base de datos integrada en la aplicación, donde se almacenan los productos y sus detalles. SQLite permite realizar operaciones CRUD de manera rápida y sencilla, ideal para aplicaciones móviles.

RecyclerView: Componente visual utilizado para mostrar el catálogo de productos en una lista.

Explicación de las nuevas funcionalidades que la aplicación contiene

Nuevas características añadidas durante el desarrollo:

Una de las características principales que se añade, es que, al momento de ingresar las credenciales para iniciar sesión, sistema recoge el rol del usuario y en caso de que este sea “Administrador” nos lleva al activity para hacer el CRUD en los productos, de lo contrario solo nos permite seleccionar productos y agregarlos al carro de compra.

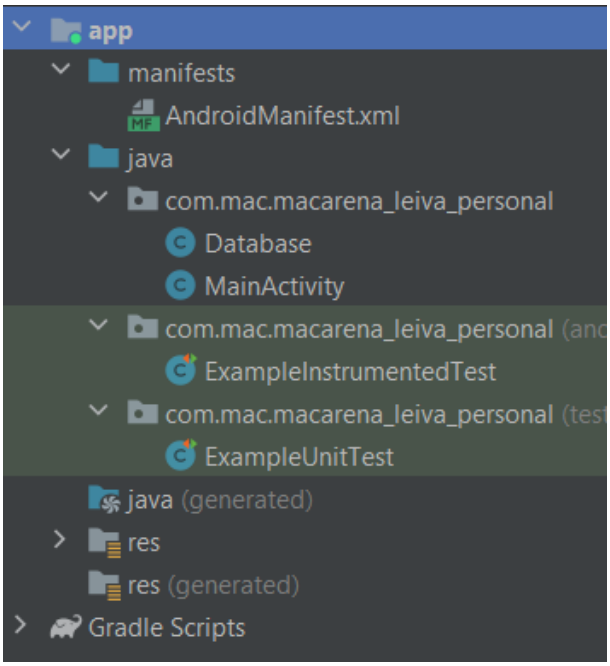
Como estas funcionalidades mejoran la experiencia del usuario:

Esto es demasiado importante, ya que al permitir que el usuario “normal” pueda hacer un CRUD puede desatarse el caos, ya que cualquiera podría ingresar datos y estos no ser acorde a lo que de verdad se tiene.

Demostración de conexión a la base de datos

Explicación como la aplicación se conecta a la base de datos:

Primero que todo, para poder hacer la base de datos, creé una clase: “Database”



Dentro de la clase está el siguiente código:

El package y las importaciones:

```
package com.mac.macarena_leiva_personal;
import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import java.util.ArrayList;
```

```
public class Database extends SQLiteOpenHelper {  2 usages

    private static final String Database_Name = "Mac_Lei.db";  1 usage
    private static final String Table_name = "productos";  6 usages
    private static final String Col_1 = "ID";  1 usage
    private static final String Col_2 = "NOMBRE";  3 usages
    private static final String Col_3 = "CANTIDAD";  3 usages
```

Esta primera sección define la clase Database, que extiende SQLiteOpenHelper, lo cual permite crear y gestionar una base de datos SQLite. Aquí se definen constantes importantes:

- Database_Name es el nombre de la base de datos ("Mac_Lei.db").
- Table_name es el nombre de la tabla ("productos").
- Col_1, Col_2, Col_3 son los nombres de las columnas de la tabla (ID, NOMBRE, CANTIDAD).

```
@Override
public void onCreate(SQLiteDatabase db) {
    db.execSQL(
            "CREATE TABLE " + Table_name + " (ID INTEGER PRIMARY KEY AUTOINCREMENT, NOMBRE TEXT, CANTIDAD INTEGER)"
    );
}
```

El método onCreate se ejecuta la primera vez que se crea la base de datos. Aquí, la tabla productos es creada usando un comando SQL (CREATE TABLE). Define tres columnas:

- ID: tipo INTEGER y clave primaria, con autoincremento.
- NOMBRE: tipo TEXT para almacenar el nombre del producto.
- CANTIDAD: tipo INTEGER para almacenar la cantidad del producto.

```java
@Override  no usages
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    db.execSQL("DROP TABLE IF EXISTS " + Table_name);
    onCreate(db);
}
```

El método onUpgrade se llama cuando se actualiza la versión de la base de datos. En este caso, elimina la tabla productos si ya existe y luego llama a onCreate para recrearla. Esto permite que se hagan cambios en la estructura de la base de datos si es necesario.

```java
public boolean insertProducto(String nombre, int cantidad) {  1 usage
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues contentValues = new ContentValues();
    contentValues.put(Col_2, nombre);
    contentValues.put(Col_3, cantidad);
    long result = db.insert(Table_name,  nullColumnHack: null, contentValues);
    return result != -1;
}
```

El método insertProducto permite agregar un nuevo producto a la tabla. Recibe el nombre y cantidad del producto, abre la base de datos en modo escritura, y usa ContentValues para insertar los datos en la tabla productos. Retorna true si la inserción fue exitosa.

```java
public ArrayList<String> getAllProductos() {  2 usages
    ArrayList<String> productos = new ArrayList<>();
    SQLiteDatabase db = this.getReadableDatabase();
    Cursor res = db.rawQuery( sql: "SELECT * FROM " + Table_name,  selectionArgs: null);
    if (res.moveToFirst()) {
        do {
            int id = res.getInt(res.getColumnIndexOrThrow(Col_1));
            String nombre = res.getString(res.getColumnIndexOrThrow(Col_2));
            int cantidad = res.getInt(res.getColumnIndexOrThrow(Col_3));
            productos.add("ID: " + id + ", Producto: " + nombre + ", Cantidad: " + cantidad);
        } while (res.moveToNext());
    }
    res.close();
    return productos;
}
```

Este método devuelve una lista de todos los productos en la base de datos. Usa un Cursor para recorrer cada fila y extraer los valores de ID, NOMBRE y CANTIDAD, formateándolos en una cadena y agregándolos a un ArrayList. Este ArrayList contiene la información de todos los productos.

```java
public void deleteProducto(String nombre) {  1 usage
    SQLiteDatabase db = this.getWritableDatabase();
    db.delete(Table_name,  whereClause: "NOMBRE = ?", new String[]{nombre});
}
```

El método deleteProducto elimina un producto de la base de datos. Recibe el nombre del producto, abre la base en modo escritura y usa el método delete de SQLiteDatabase para eliminar la fila donde NOMBRE coincide con el nombre dado.

```
    public void updateProducto(String oldNombre, String newNombre, int newCantidad) {  1 usage
        SQLiteDatabase db = this.getWritableDatabase();
        ContentValues contentValues = new ContentValues();
        contentValues.put(Col_2, newNombre);
        contentValues.put(Col_3, newCantidad);
        db.update(Table_name, contentValues,  whereClause: "NOMBRE = ?", new String[]{oldNombre});
    }
}
```

El método updateProducto permite modificar la información de un producto existente. Recibe el nombre antiguo, el nuevo nombre y la nueva cantidad, abre la base en modo escritura y usa ContentValues para definir los valores actualizados. Luego, usa update para cambiar la fila donde NOMBRE coincide con el nombre antiguo.

El MainActivity también fue modificado:

```
public class MainActivity extends AppCompatActivity {

    Database database;  6 usages
    EditText ingresarNombre, ingresarCantidad, editarNombre, editarCantidad;  3 usages
    Button btnAgregar, btnEditar, btnEliminar;  2 usages
    ListView listarProductos;  3 usages
    ArrayList<String> productos;  5 usages
    ArrayAdapter<String> adapter;  3 usages
    String productoSeleccionado;  4 usages
```

Aquí declare las variables.

```java
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    database = new Database( context: this);
    ingresarNombre = findViewById(R.id.editTextTask);
    ingresarCantidad = findViewById(R.id.editTextCantidad);
    editarNombre = findViewById(R.id.editTextEditTask);
    editarCantidad = findViewById(R.id.editTextEditCantidad);
    btnAgregar = findViewById(R.id.buttonAdd);
    btnEditar = findViewById(R.id.buttonEdit);
    btnEliminar = findViewById(R.id.buttonDelete);
    listarProductos = findViewById(R.id.listViewTasks);
```

En onCreate, se inicializan los elementos de la interfaz de usuario asociándolos con sus IDs en el archivo de diseño XML (activity_main.xml). También se crea una instancia de Database.

```java
productos = database.getAllProductos();
adapter = new ArrayAdapter<>( context: this, android.R.layout.simple_list_item_1, productos);
listarProductos.setAdapter(adapter);
```

Se recuperan los productos almacenados en la base de datos (getAllProductos) y se agregan a productos, que es un ArrayList<String>. Se crea un adaptador ArrayAdapter para mostrar los productos en listarProductos (un ListView).

```
btnAgregar.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        String nombre = ingresarNombre.getText().toString();
        String cantidades = ingresarCantidad.getText().toString();
        if (!nombre.isEmpty() && !cantidades.isEmpty()) {
            int cantidad = Integer.parseInt(cantidades);
            database.insertProducto(nombre, cantidad);
            ActualizarListadoProductos();
            ingresarNombre.setText("");
            ingresarCantidad.setText("");
        }
    }
});
```

Agrega un producto a la base de datos:

- Obtiene los valores de ingresarNombre y ingresarCantidad.
- Verifica que ambos no estén vacíos.
- Convierte cantidades a un número entero y llama a insertProducto para guardar el producto en la base de datos.
- Llama a ActualizarListadoProductos para refrescar la lista.
- Limpia los campos de entrada de texto.

```java
listarProductos.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> adapterView, View v, int position, long id) {
        productoSeleccionado = productos.get(position);
        String[] partes = productoSeleccionado.split( regex: ", ");
        editarNombre.setText(partes[1].split( regex: ": ")[1]);
        editarCantidad.setText(partes[2].split( regex: ": ")[1]);
        editarNombre.setVisibility(View.VISIBLE);
        editarCantidad.setVisibility(View.VISIBLE);
        btnEditar.setVisibility(View.VISIBLE);
        btnEliminar.setVisibility(View.VISIBLE);
    }
});
```

Permite seleccionar un producto:

- Cuando el usuario hace clic en un producto en la lista, se guarda en productoSeleccionado.
- Divide productoSeleccionado en partes para obtener el nombre y cantidad.
- Muestra los campos de edición (editarNombre, editarCantidad) y los botones (btnEditar, btnEliminar) para modificar o eliminar el producto.

```java
btnEditar.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        String nuevoNombre = editarNombre.getText().toString();
        String nuevaCantidades = editarCantidad.getText().toString();
        if (!nuevoNombre.isEmpty() && !nuevaCantidades.isEmpty()) {
            int nuevaCantidad = Integer.parseInt(nuevaCantidades);
            database.updateProducto(productoSeleccionado.split( regex: ", ")[1].split( regex: ": ")[1], nuevoNombre, nuevaCantidad);
            ActualizarListadoProductos();
            LimpiarCampos();
        }
    }
});
```

Permite editar un producto:

- Obtiene los valores de editarNombre y editarCantidad.
- Verifica que ambos no estén vacíos.
- Convierte nuevaCantidades en un entero.
- Llama a updateProducto para actualizar el producto seleccionado en la base de datos, usando el nombre antiguo (obtenido de productoSeleccionado).
- Actualiza la lista de productos y limpia los campos de edición.

```
btnEliminar.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        database.deleteProducto(productoSeleccionado.split( regex: ", ")[1].split( regex: ": ")[1]);
        ActualizarListadoProductos();
        LimpiarCampos();
    }
});
```

Elimina un producto:

- Llama a deleteProducto para borrar el producto seleccionado, extrayendo su nombre de productoSeleccionado.
- Actualiza la lista de productos.
- Limpia los campos de edición y oculta los controles de edición.

```
public void ActualizarListadoProductos() {  3 usages
    productos.clear();
    productos.addAll(database.getAllProductos());
    adapter.notifyDataSetChanged();
}
```

Actualiza la lista de productos:

- Limpia el ArrayList productos.
- Recupera todos los productos de la base de datos y los añade a productos.
- Llama a notifyDataSetChanged() en el adaptador para que el ListView se refresque y muestre los datos actualizados.

```
public void LimpiarCampos() {  2 usages
    editarNombre.setText("");
    editarCantidad.setText("");
    editarNombre.setVisibility(View.GONE);
    editarCantidad.setVisibility(View.GONE);
    btnEditar.setVisibility(View.GONE);
    btnEliminar.setVisibility(View.GONE);
}
```

Limpia y oculta los campos de edición después de editar o eliminar un producto:

- Borra el texto de editarNombre y editarCantidad.
- Oculta editarNombre, editarCantidad, btnEditar y btnEliminar para que no interfieran en otras acciones.

Por último, también modifique el activity_main:

Solo dejaré captura de lo que ocupé dentro de la implementación del CRUD, creo que no necesita mayor explicación.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/padding"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <!-- Campo para ingresar un nuevo producto -->
    <EditText
        android:id="@+id/editTextTask"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Nombre del producto"
        android:paddingTop="20dp" />
```

```
<!-- Campo para ingresar la cantidad del producto -->
<EditText
    android:id="@+id/editTextCantidad"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_below="@id/editTextTask"
    android:hint="Cantidad"
    android:inputType="number" />
```

```
<!-- Botón para agregar el producto -->
<Button
    android:id="@+id/buttonAdd"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@id/editTextCantidad"
    android:text="Agregar" />

<!-- Lista para mostrar los productos -->
<ListView
    android:id="@+id/listViewTasks"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_below="@id/buttonAdd" />

<!-- Campo para editar el producto seleccionado -->
<EditText
    android:id="@+id/editTextEditTask"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_below="@id/listViewTasks"
    android:hint="Editar nombre"
    android:visibility="gone" />
```

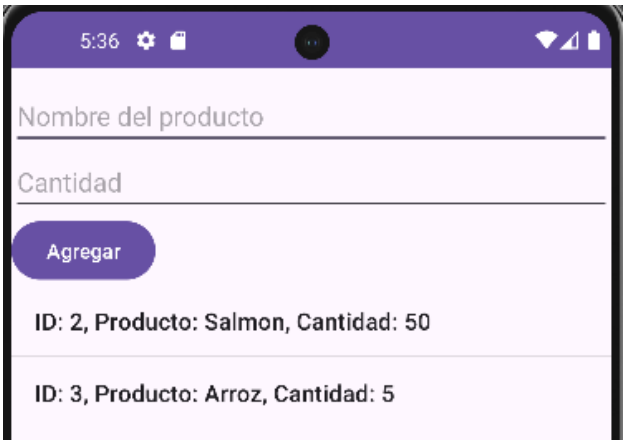
```
<!-- Campo para editar la cantidad del producto seleccionado -->
<EditText
    android:id="@+id/editTextEditCantidad"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_below="@id/editTextEditTask"
    android:hint="Editar cantidad"
    android:inputType="number"
    android:visibility="gone" />

<!-- Botón para guardar cambios -->
<Button
    android:id="@+id/buttonEdit"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@id/editTextEditCantidad"
    android:text="Guardar Cambios"
    android:visibility="gone" />
```

```
    <!-- Botón para eliminar el producto -->
    <Button
        android:id="@+id/buttonDelete"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_toEndOf="@id/buttonEdit"
        android:layout_below="@id/editTextEditCantidad"
        android:text="Eliminar"
        android:visibility="gone" />
</RelativeLayout>
```

Ejemplos de operaciones CRUD:

En este caso, el CRUD consiste en que el usuario (Administrador) pueda ingresar productos.

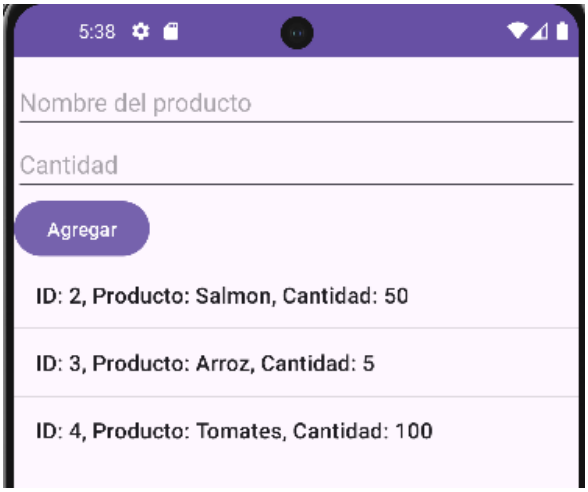
Ejecutamos la aplicación y nos aparece esta ventana. Podemos verificar que la aplicación nos entrega los datos ingresados sin necesidad de ingresar datos antes para ver la lista (Leer)



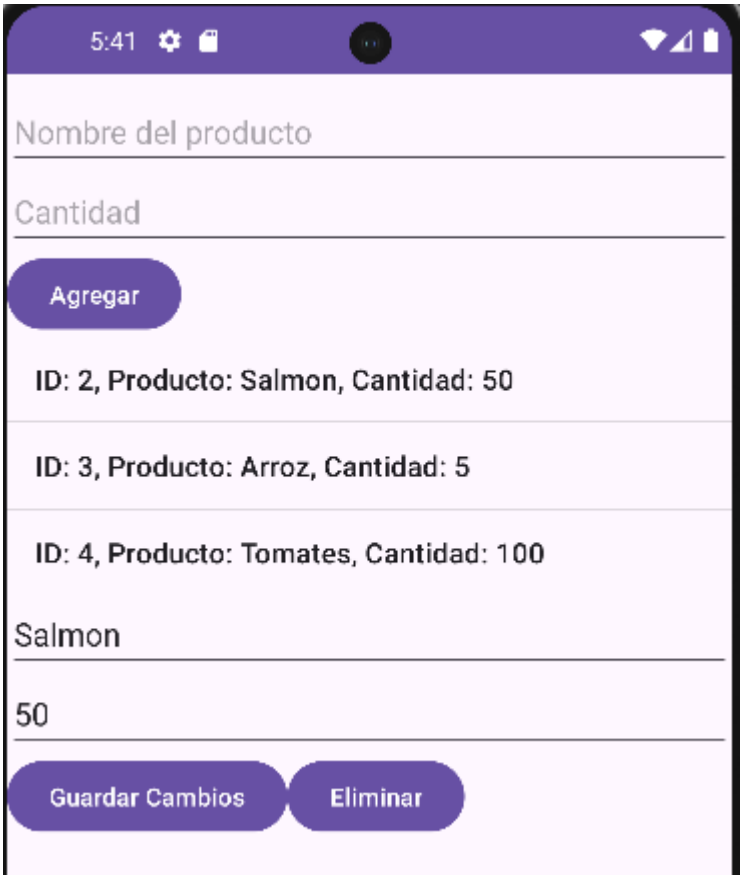
Aquí yo ya había ingresado datos.

De igual manera mostrare que se pueden ingresar datos

Ingresar: ingresé “Tomates”, una vez que los campos fueron llenados, seleccionamos el botón “Agregar” y se agrega el producto.



Actualizar: Seleccioné “Salmon”



Al seleccionar un producto, automáticamente aparecen los botones “Guardar Cambios” y “Eliminar”. En este caso vamos a cambiar la cantidad de los salmones.

Se puede verificar que el dato cambió correctamente.

Eliminar: Seleccionamos un producto (“Salmon”) y pinchamos “Eliminar”

A continuación, está la captura que verifica que el producto fue eliminado.

Descripción de como se manejan posibles errores de conexión:

Hasta el momento no he implementado posibles errores de conexión, pero espero que al finalizar mi proyecto esto quede completado.

Algunas cosas que podría hacer son:

- **Verificación de Conectividad Inicial**: Antes de realizar operaciones que dependen de una conexión a Internet, la aplicación verifica el estado de la conectividad del dispositivo. Esto se logra utilizando servicios de conectividad que permiten comprobar si el dispositivo está conectado a una red activa.

- **Notificación de Error en Tiempo Real**: Cuando se detecta un problema de conexión (por ejemplo, una desconexión repentina), la aplicación notifica al usuario de manera inmediata. Esto suele hacerse mediante un mensaje emergente (toast) o una alerta que indica la falta de conexión. La notificación puede incluir recomendaciones, como activar los datos móviles o conectarse a una red Wi-Fi.

Elementos finales para que la aplicación quede terminada

Descripción de las pruebas que aún deben realizarse:

Fuera de la realización del CRUD, necesito hacer las pruebas para identificar el rol del usuario, para que me lleve a la ventana correspondiente según si es administrador o cliente.

Además, tengo que verificar que el cliente pueda agregar productos al carro de compra y que, una vez finalizado el pedido, las cantidades seleccionadas se descuenten del inventario de cada producto.

Aspectos por optimizar o corregir antes de la finalización:

Puedo optimizar el proyecto para que me diga si estoy o no conectada a internet, para que no haya ninguna falla en caso de que la respuesta sea negativa.

Planificación de las últimas etapas del desarrollo:

En este caso es bueno hacer una tabla:

Actividad	Descripción
Pruebas de usabilidad	Realizar pruebas para identificar mejoras
Corrección de errores	Abordar los errores que pueden haberse encontrado.
Documentación	Hacer un informe detallado con las funcionalidades de la aplicación
Entrega	Hacer entrega de la aplicación al docente para ser evaluada.

Conclusión

Reflexión sobre el desarrollo y los aprendizajes obtenidos:

Uno de los aprendizajes más significativos ha sido la importancia de la planificación y la organización en cada etapa del proyecto. Una buena planificación puede facilitar el camino hacia el éxito. Aprender a priorizar tareas y responsabilidades ha sido clave para mantener el enfoque y la motivación.

Aunque el proyecto es individual, a sido de gran ayuda compartir ideas con compañeros y ayudarnos entre nosotros.

Desde un punto de vista técnico, el desarrollo del CRUD para los productos ha proporcionado una comprensión más profunda de las bases de datos y su integración con aplicaciones móviles.

Reconocimientos y agradecimientos:

Agradecer al profesor por subir clase a clase el material que utilizamos y dejarlo a disposición del curso.