

**UNIVERSIDAD NACIONAL DE COLOMBIA**  
**FACULTAD DE INGENIERÍA**  
**CURSO: Programación Orientada a Objetos**

---

**DOCUMENTACIÓN DEL SISTEMA – MYTIENDITA**

**Sistema de Gestión para la Tienda Deportiva MaxGear**

*“Tu cuerpo lo pide, tu mente lo exige”*

---

**INTEGRANTES DEL EQUIPO:**

- Juan José Ríos Trujillo
- Javier Esteban Moreno Garzón
- Samuel Eduardo García Corredor

**DOCENTE:**

Néstor Germán Bolívar Pulgarín

**FECHA DE ENTREGA:**

15 de julio de 2025

**VERSIÓN DEL PROYECTO:**

Sprint 4 – Proyecto final de curso

## Índice

### 1. Introducción

- 1.1. Contexto del proyecto
- 1.2. Objetivo general

### 2. Propósito del sistema

- 2.1. Justificación
- 2.2. Público objetivo
- 2.3. Rol del sistema en MaxGear

### 3. Alcance del producto

- 3.1. Funcionalidades principales
- 3.2. Limitaciones actuales
- 3.3. Análisis de requisitos
- 3.4. Diseño y arquitectura
- 3.5. Programación y tecnologías utilizadas
- 3.6. Pruebas funcionales realizadas
- 3.7. Documentación del código fuente
- 3.8. Consideraciones de mantenimiento

### 4. Entorno operativo

- 4.1. Plataforma y herramientas utilizadas
- 4.2. Requisitos de ejecución
- 4.3. Compatibilidad
- 4.4. Infraestructura del proyecto
- 4.5. Arquitectura física y dependencias
- 4.6. Observaciones de despliegue

### 5. Requerimientos funcionales

- 5.1. Módulo de autenticación y gestión de usuarios
- 5.2. Módulo de gestión de productos (catálogo)
- 5.3. Módulo de carrito de compras
- 5.4. Módulo de recarga de saldo
- 5.5. Módulo de confirmación de compra (checkout)
- 5.6. Módulo de estadísticas y descuentos

### 6. Requerimientos no funcionales

- 6.1. Seguridad
- 6.2. Rendimiento



UNIVERSIDAD  
NACIONAL  
DE COLOMBIA

- 6.3. Disponibilidad
- 6.4. Compatibilidad
- 6.5. Usabilidad
- 6.6. Mantenibilidad y extensibilidad

## 7. Manual de usuario

- 7.1. Pantalla de login
- 7.2. Pantalla de registro
- 7.3. Menú principal
- 7.4. Vista de productos
- 7.5. Carrito de compras
- 7.6. Confirmación de compra (checkout)
- 7.7. Recarga de saldo
- 7.8. Resumen de ventas y estadísticas

---

### ***Historial de versiones – MyTiendita***

---

#### **Sprint 1 – Estructura Base y Autenticación**

**Fecha de entrega:** 13 de mayo de 2025

- Se creó el proyecto Android en Java con Firebase como backend.
  - Se implementó el sistema de **registro** y **login** usando Firebase Authentication.
  - Se construyó la base de datos en Firebase Realtime Database con estructura inicial para usuarios.
  - Se añadió navegación básica entre pantallas (login, registro, menú principal).
  - Se incluyó asignación automática del rol "cliente" al momento de registrarse.
- 

#### **Sprint 2 – Módulo de Productos y Roles**

**Fecha de entrega:** 27 de mayo de 2025



- Se implementó la vista del **catálogo de productos** usando **RecyclerView** conectado a Firebase.
  - Se introdujo la lógica de **roles** (cliente/admin) para mostrar u ocultar botones desde **MainActivity**.
  - El administrador puede **agregar, editar y eliminar productos**.
  - El cliente puede ver los productos y agregarlos a su carrito.
  - Se integró diseño básico para las tarjetas de producto (**item\_producto\_card.xml**).
- 

### Sprint 3 – Carrito de Compras y Recarga de Saldo

**Fecha de entrega:** 10 de junio de 2025

- Se creó el módulo completo de **carrito de compras**, con suma y resta de cantidades y eliminación de ítems.
  - Se implementó la pantalla de **recarga de saldo**, que simula una recarga y actualiza el campo **saldo** en Firebase.
  - Se habilitó el botón "Comprar" desde el carrito para acceder al **checkout**.
  - Se añadieron validaciones básicas (mínimo 1 unidad por producto, saldo suficiente).
  - Se realizaron pruebas funcionales de flujo completo cliente → compra.
- 

### Sprint 4 – Ventas, Estadísticas y Cierre del Sistema

**Fecha de entrega:** 15 de julio de 2025

- Se desarrolló la pantalla de **checkout** con cálculo del total, validación de saldo y registro de la venta.

- Las ventas se almacenan en la base de datos bajo `ventas/{id}` con productos, total, fecha y usuario.
- Se implementó el módulo de **estadísticas**, incluyendo gráficos de barras y pastel con MPAndroidChart.
- Se añadieron **sugerencias de descuento** y **alertas de reabastecimiento** según el análisis de ventas.
- Se finalizó la documentación, pruebas finales y diseño de la interfaz definitiva.

## **1. Introducción**

En un entorno comercial cada vez más competitivo, la eficiencia operativa se ha convertido en un factor crítico para el éxito de los negocios físicos. Las tiendas deportivas, como **MaxGear**, no son la excepción. En la actualidad, muchas de estas tiendas siguen operando con métodos manuales para controlar su inventario y registrar ventas, lo que genera errores frecuentes, baja trazabilidad, pérdida de oportunidades y una escasa capacidad de análisis estratégico.

**MyTiendita** es una aplicación móvil nativa, desarrollada como proyecto final para el curso de Programación Orientada a Objetos. Fue creada con el objetivo de servir como sistema digital de gestión integral para la tienda deportiva **MaxGear**, empresa cuyo lema “Tu cuerpo lo pide, tu mente lo exige” resalta la importancia del rendimiento físico y mental, lo cual se refleja también en su operatividad interna.

Esta aplicación busca optimizar y modernizar los procesos de la tienda, permitiendo al personal encargado controlar el stock, registrar compras, validar pagos simulados por parte del cliente, aplicar descuentos estratégicos y analizar las ventas realizadas a través de estadísticas visuales. Todo esto en un entorno conectado, flexible y respaldado por Firebase, una plataforma robusta para servicios móviles.

## **2. Propósito**

El propósito central de **MyTiendita** es convertirse en el **sistema de gestión oficial de MaxGear**, reemplazando procesos manuales por una solución digital eficiente, escalable y

segura. Su diseño está pensado específicamente para las operaciones internas de esta tienda deportiva, y busca brindar a los encargados:

- Control completo sobre el catálogo de productos deportivos.
- Registro confiable de compras realizadas en el punto de venta.
- Simulación de pagos mediante una función de recarga de saldo.
- Validación de disponibilidad de stock y fondos antes de registrar una venta.
- Generación de reportes y sugerencias de descuentos con base en los productos más vendidos.

Todo el sistema está centrado en el flujo real de atención al cliente en tienda física. El cliente nunca interactúa directamente con la app: es el vendedor quien ejecuta las acciones según las decisiones tomadas en tiempo real (ej. cuánto paga el cliente, qué productos elige, cuánto queda de saldo).

El sistema opera con los siguientes componentes:

- **Firebase Authentication:** Para el acceso seguro del personal.
- **Firebase Realtime Database:** Para almacenamiento en la nube, sincronización y análisis de productos, ventas y usuarios.
- **Arquitectura basada en MVC**, lo que permite mantener un orden claro entre datos, lógica de negocio y vistas gráficas.

### ***3. Alcance del producto***

El alcance funcional de **MyTiendita** cubre los procesos operativos esenciales de **MaxGear**, pensados para su uso exclusivo por parte de empleados autorizados (con rol de "admin") o vendedores (rol de "cliente"). No se trata de una tienda online ni de una app orientada al **3.**

### **3.1. Funcionalidades principales**

El sistema **MyTiendita** está diseñado para cubrir las siguientes funcionalidades principales:

- Registro y autenticación de usuarios mediante Firebase.
- Asignación automática del rol "cliente" en el registro.
- Diferenciación visual de opciones según el rol (cliente o administrador).
- Gestión de productos: agregar, editar y eliminar productos (solo administrador).
- Visualización del catálogo por parte de los clientes.
- Agregado de productos al carrito y gestión de cantidades.
- Cálculo automático del total del carrito.
- Recarga simulada de saldo por parte del cliente.
- Proceso de compra con validación de saldo.
- Registro de ventas en la base de datos.
- Visualización de estadísticas de productos vendidos.
- Sugerencias de descuentos y reabastecimiento basadas en el historial de ventas.

---

### **3.2. Limitaciones actuales**

El sistema, en su versión actual (Sprint 4), presenta las siguientes limitaciones:

- No se actualiza automáticamente el stock de productos después de la venta.
- No existe un sistema de roles múltiples más allá de "cliente" y "admin".

- La validación de entradas es básica (no se validan formatos de correo o campos numéricos).
  - No se incluye un historial de compras visible para el cliente.
  - El sistema depende completamente de la conectividad con Firebase (no tiene modo offline).
- 

### 3.3. Análisis de requisitos

Se recopilaron los requisitos funcionales y no funcionales esperados para una aplicación de gestión básica de tienda, ajustados al contexto de una tienda física que desea digitalizar su inventario y ventas.

Se priorizó la separación de responsabilidades entre el cliente y el administrador, y la interacción fluida con la base de datos en tiempo real, aprovechando la estructura de Firebase Realtime Database.

---

### 3.4. Diseño y arquitectura

El sistema está desarrollado en Java para Android, utilizando una arquitectura basada en:

- Actividades (**Activities**) para representar cada pantalla.
- Clases modelo (**Producto**, **CarritoItem**, **Usuario**) para mapear datos de Firebase.
- **RecyclerView** y adaptadores personalizados para listas dinámicas.
- Firebase Authentication y Firebase Realtime Database como backend central.

El diseño es modular y fácilmente ampliable para futuras funcionalidades.

---

### 3.5. Programación y tecnologías utilizadas



- Android Studio con Java
  - Firebase Authentication (login y registro)
  - Firebase Realtime Database (almacenamiento de productos, usuarios, carritos y ventas)
  - Firebase UI (sinc. con adaptadores RecyclerView)
  - Librería MPAndroidChart (para estadísticas gráficas)
- 

### **3.6. Pruebas funcionales realizadas**

Se realizaron pruebas manuales para verificar:

- Registro e inicio de sesión.
  - Diferenciación de roles en **MainActivity**.
  - Flujo completo de compra desde el catálogo hasta el checkout.
  - Registro correcto de ventas en la base de datos.
  - Visualización coherente de productos en el carrito.
  - Cálculo correcto de saldos y total del carrito.
  - Visualización de estadísticas con datos simulados.
- 

### **3.7. Documentación del código fuente**

Cada clase incluye comentarios que explican la lógica principal de sus métodos. Las actividades están separadas por funcionalidad y se nombran de manera semántica.

La estructura de carpetas y clases facilita la comprensión y mantenimiento del sistema.



---

### **3.8. Consideraciones de mantenimiento**

La arquitectura utilizada permite:

- Agregar nuevos roles o permisos con modificaciones mínimas.
- Añadir una pantalla de historial de compras por usuario.
- Implementar control de stock restando unidades tras cada venta.
- Exportar reportes o generar recibos electrónicos.

---

## **4. Entorno Operativo (*versión ampliada y profesional*)**

---

### **4.1. Plataforma y herramientas utilizadas**

La aplicación **MyTiendita** fue desarrollada como una solución nativa para dispositivos Android. Se utilizó **Android Studio** como entorno principal, haciendo uso de librerías de Firebase para gestionar la autenticación y el almacenamiento de datos en la nube.

**Herramientas y tecnologías:**

Tecnología / Herramienta	Función dentro del proyecto
<b>Android Studio</b>	IDE de desarrollo principal (versión recomendada: 2023.1+)
<b>Java</b>	Lenguaje de programación utilizado

<b>Firebase Authentication</b>	Control de acceso y login con email/contraseña
<b>Firebase Realtime Database</b>	Almacenamiento de usuarios, productos, carritos y ventas
<b>Firebase (RecyclerViewAdapter)</b>	UI Sincronización automática de listas (catálogo y carrito)
<b>Gradle (KTS)</b>	Sistema de compilación y gestión de dependencias
<b>XML Layouts</b>	Diseño de interfaces gráficas (layouts individuales por pantalla)
<b>Google Services Plugin</b>	Integración general con servicios de Firebase

---

## 4.2. Requisitos de ejecución en el dispositivo

Para ejecutar correctamente la aplicación **MyTiendita** en un dispositivo Android, se requiere cumplir con las siguientes condiciones:

<b>Requisito técnico</b>	<b>Valor mínimo recomendado</b>
<b>Sistema operativo</b>	Android 5.0 Lollipop (API 21) o superior
<b>Conectividad</b>	Obligatoria (uso en línea, requiere conexión a Internet)

<b>RAM del dispositivo</b>	Mínimo 1 GB
<b>Espacio de almacenamiento</b>	~50 MB (dependiendo del tamaño del caché y base de datos local)
<b>Permisos de Android</b>	Acceso a Internet
<b>Cuenta de usuario</b>	Email válido registrado en Firebase

**Nota:** La app no requiere permisos de cámara, ubicación ni almacenamiento externo. Toda la lógica se ejecuta localmente con sincronización en la nube.

---

#### 4.3. Compatibilidad

La aplicación **MyTiendita** está diseñada exclusivamente para dispositivos Android. A continuación, se describen los aspectos de compatibilidad:

- Compatible con dispositivos que usen Android 5.0 (API 21) o versiones superiores.
  - Diseñada para funcionar correctamente en smartphones y tabletas.
  - Adaptada principalmente para orientación vertical.
  - No compatible con iOS, sistemas operativos de escritorio o navegadores web.
  - No se encuentra publicada en la Google Play Store, ya que su distribución es local/académica.
- 

#### 4.4. Infraestructura lógica del proyecto (estructura de paquetes)

La estructura del proyecto en Android Studio se organizó en paquetes y clases siguiendo un esquema limpio y semántico, como se evidencia en la captura:



```

com.example.mytiendita
|
+-- Actividades principales:
|   +-- MainActivity.java           ← Menú principal según rol
|   +-- Database.java              ← Vista del catálogo de productos
|   +-- CarritoActivity.java       ← Visualización y gestión del carrito
|   +-- CheckoutActivity.java     ← Validación de saldo y finalización de venta
|   +-- AddProductActivity.java   ← Agregar nuevos productos
|   +-- EditProductActivity.java  ← Editar productos existentes
|   +-- DeleteProductActivity.java ← (Opcional, si no se maneja desde Database)
|   +-- RecargarActivity.java    ← Simulación de saldo ingresado por el cliente
|   +-- ResumenVentasActivity.java ← Módulo de descuentos y estadísticas
|   +-- Login.java / Register.java ← Pantallas de autenticación
|
+-- Modelos de datos:
|   +-- Producto.java             ← Contiene nombre, precio y stock
|   +-- CarritoItem.java          ← Elemento individual del carrito
|   +-- Usuario.java              ← Información asociada al login
|
+-- Adaptadores y visualización:
|   +-- ProductoAdapter.java / ProductoViewHolder.java
|   +-- CarritoViewHolder.java
|
+-- Recursos XML:
|   +-- Layouts principales:
|       +-- activity_main.xml
|       +-- activity_login.xml / register.xml
|       +-- activity_carrito.xml / checkout.xml / database.xml
|       +-- activity_resumen_ventas.xml / recargar.xml
|       +-- edit_producto.xml / add_producto.xml
|
|   +-- Items visuales (listas):
|       +-- item_producto_card.xml
|       +-- item_carrito_card.xml
|

```

## 4.5. Arquitectura física y dependencias en tiempo de ejecución

- Toda la lógica de persistencia y lectura de datos depende de la disponibilidad de **Firebase**.
- Las dependencias se resuelven en tiempo de compilación mediante **Gradle (Kotlin DSL)**.
- Los datos están centralizados en Firebase Realtime Database con acceso controlado por roles.

## 4.6. Observaciones sobre instalación y despliegue

- La aplicación fue construida como **proyecto académico**, por lo que el despliegue se realiza por instalación directa del **.apk**.
- No utiliza Google Play Services para notificaciones push ni almacenamiento externo.
- Al estar conectada a Firebase, requiere que el archivo **google-services.json** esté correctamente configurado.

## 5. Requerimientos Funcionales

A continuación se describen los requerimientos funcionales del sistema **MyTiendita**, organizados en seis módulos clave que reflejan los flujos reales de uso de la aplicación. Esta sección detalla las funcionalidades, reglas de negocio, validaciones internas y comportamientos esperados en cada parte del sistema.

---

### 5.1. Módulo de Autenticación y Gestión de Usuarios

**Objetivo:** Permitir la creación y validación de usuarios mediante Firebase Authentication, asignarles automáticamente un rol y saldo inicial, y controlar su acceso a las funcionalidades correspondientes.

**Requisitos funcionales:**

ID	Requerimiento
----	---------------

RF01 El sistema debe permitir el registro de nuevos usuarios mediante email y contraseña.

RF02 Al registrarse, se debe crear automáticamente el nodo **usuarios/{uid}** en Firebase.

RF03 El nodo del usuario debe contener los campos: `correo`, `rol = cliente`, `saldo = 0`.

RF04 El login debe validar email y contraseña, y permitir el acceso solo si son correctos.

RF05 Si el usuario ya está autenticado, debe redirigirse directamente al `MainActivity`.

RF06 El sistema debe leer el campo `rol` desde Firebase para mostrar botones según rol.

RF07 Si no se encuentra el campo `rol`, debe mostrarse un mensaje de error ("Rol desconocido").

RF08 Si los campos están vacíos, se debe mostrar advertencia (sin continuar con el flujo).

RF09 Actualmente no se valida el formato del correo ni longitud mínima de contraseña (*limitación conocida*).

---

## 5.2. Módulo de Gestión de Productos (Catálogo – Solo Admin)

**Objetivo:** Permitir al usuario con rol de “admin” agregar, editar o eliminar productos del catálogo en Firebase. Los productos se visualizan en tiempo real y están disponibles para los clientes.

**Requisitos funcionales:**

ID	Requerimiento
----	---------------

- RF10 El admin debe poder agregar productos indicando: nombre, precio y stock.
- RF11 El sistema debe guardar los productos en la rama `productos/{id}`.
- RF12 El admin debe poder editar cualquiera de los tres campos de un producto existente.
- RF13 El admin debe poder eliminar productos del catálogo desde la vista `Database`.
- RF14 Al editar o eliminar un producto, los cambios deben reflejarse en tiempo real.
- RF15 Si se intenta ingresar stock negativo o campos vacíos, se debe evitar el guardado (*se asume como validación futura necesaria*).
- RF16 Productos con nombres duplicados pueden ser agregados sin restricciones (*limitación técnica actual*).

---

### 5.3. Módulo de Carrito de Compras

**Objetivo:** Permitir a los usuarios con rol "cliente" agregar productos al carrito, ajustar cantidades y calcular el total dinámicamente. Los datos se almacenan bajo `carritos/{uid}`.

**Requisitos funcionales:**

ID	Requerimiento
----	---------------

- RF17 El cliente debe poder agregar productos desde el catálogo a su carrito personal.



- RF18 El carrito se guarda en la ruta `carritos/{uid}`.
- RF19 Si un producto ya existe en el carrito, al agregarlo de nuevo se **reemplaza** (no suma).
- RF20 El usuario puede aumentar o disminuir la cantidad con botones desde `CarritoActivity`.
- RF21 El total del carrito debe actualizarse automáticamente en pantalla.
- RF22 El sistema no impone límites máximos de cantidad (*puede agregarse como mejora*).
- RF23 Si la cantidad llega a 1 y se intenta restar más, se muestra un mensaje de advertencia.
- RF24 El usuario puede eliminar productos individualmente del carrito.

---

#### 5.4. Módulo de Recarga de Saldo

**Objetivo:** Simular la entrega de dinero por parte del cliente, ingresando un monto que se sumará al saldo del usuario autenticado.

**Requisitos funcionales:**

ID	Requerimiento
----	---------------

- RF25 El cliente puede ingresar un monto a través de un campo de texto.
- RF26 El sistema debe validar que el campo no esté vacío.
- RF27 El saldo actual se obtiene desde `usuarios/{uid}/saldo`.
- RF28 El monto ingresado se **suma** al saldo actual, y el nuevo saldo se actualiza en Firebase.
- RF29 No se imponen restricciones en la cantidad máxima ni verificación de tipo (*mejora posible: solo números positivos*).
- RF30 Despues de una recarga exitosa, se muestra un mensaje y se retorna automáticamente.

---

## 5.5. Módulo de Checkout y Registro de Ventas

**Objetivo:** Validar que el usuario tenga saldo suficiente para completar su compra. Si se cumple la condición, registrar la venta en Firebase y descontar el saldo.

**Requisitos funcionales:**

ID	Requerimiento
----	---------------

- RF31 El sistema debe calcular el total del carrito (`precio × cantidad` por producto).

RF32 El sistema debe obtener el saldo del usuario desde Firebase.

RF33 Si el saldo es insuficiente, se muestra un mensaje y no se permite la compra.

RF34 Si el saldo es suficiente, se debe registrar una venta en `ventas/{idVenta}` con:

→ UID del usuario, fecha (timestamp), total de la venta y lista de productos.

RF35 El saldo del usuario debe actualizarse descontando el valor total de la compra.

RF36 El carrito debe vaciarse tras una compra exitosa.

RF37 Actualmente el stock de los productos **no se actualiza automáticamente** (*limitación actual con potencial de mejora*).

---

## 5.6. Módulo de Estadísticas y Descuentos

**Objetivo:** Analizar las ventas registradas y generar sugerencias visuales para reabastecimiento o descuento, además de mostrar gráficos con comportamiento de productos vendidos.

**Requisitos funcionales:**

ID	Requerimiento
----	---------------

RF38 El sistema debe consultar todas las ventas registradas en la rama `ventas`.



- RF39 Debe contar la cantidad total de unidades vendidas por producto.
- RF40 Se deben mostrar los 3 productos más vendidos y los 3 menos vendidos.
- RF41 Para los menos vendidos, debe sugerirse aplicar descuentos entre 10% y 20%.
- RF42 Para los más vendidos, debe sugerirse considerar reabastecimiento.
- RF43 El sistema debe mostrar dos gráficos: uno de barras (BarChart) y uno de pastel (PieChart).
- RF44 El análisis debe mostrarse a través de cuadros de texto y **AlertDialog**.
- RF45 El módulo está disponible tanto para admins como para clientes.
- RF46 Los descuentos sugeridos **no se aplican automáticamente al precio del producto** (*solo sugerencia visual*).

## **6. Requerimientos No Funcionales**

---

### **6.1. Seguridad**

<b>ID</b>	<b>Requerimiento</b>
-----------	----------------------

- RNF0 El sistema debe utilizar **Firebase Authentication** para proteger el acceso  
1 mediante email y contraseña.
- RNF0 La lógica de acceso debe garantizar que solo usuarios autenticados puedan operar  
2 el sistema.
- RNF0 Los datos del usuario deben almacenarse en nodos personales  
3 (**usuarios/{uid}**) separados por ID único.
- RNF0 El sistema debe ocultar u ofrecer funciones según el rol ("admin" o "cliente")  
4 obtenido desde Firebase.
- RNF0 El sistema no debe almacenar localmente credenciales ni datos sensibles.  
5

---

## 6.2. Rendimiento

### ID Requerimiento

- RNF0 El tiempo de respuesta para operaciones comunes (login, cargar productos,  
6 carrito) debe ser menor a 2 segundos en red estable.
- RNF0 El sistema debe actualizar el contenido del carrito, productos y ventas en **tiempo**  
7 **real** utilizando Firebase.

RNF0 Las gráficas y estadísticas deben generarse dinámicamente en menos de 2  
8 segundos tras la carga de datos.

---

### 6.3. Disponibilidad

#### ID Requerimiento

RNF0 La aplicación requiere conexión a Internet para funcionar correctamente (acceso a  
9 Firebase).

RNF1 Si se pierde la conexión, el sistema debe notificar visualmente que no puede  
0 operar momentáneamente (*mejora pendiente*).

RNF11 Las funcionalidades dependen de la disponibilidad de los servicios de Firebase.

---

### 6.4. Compatibilidad

#### ID Requerimiento

RNF1 La aplicación debe funcionar en dispositivos Android con versión **API 21**  
2 (**Lollipop**) o superior.

RNF1 La interfaz debe estar adaptada a pantallas de celulares y tablets en orientación  
3 vertical.

RNF1 La app no es compatible con sistemas iOS, Windows ni plataformas web (*versión solo Android*).  
4

---

## 6.5. Usabilidad

### ID Requerimiento

RNF1 La aplicación debe mostrar mensajes claros ante errores de usuario (por ejemplo, 5 campos vacíos).

RNF1 La navegación debe ser intuitiva, con botones grandes y etiquetas 6 autoexplicativas.

RNF1 La distinción entre roles debe estar clara desde el **MainActivity**, mostrando 7 solo lo necesario.

RNF1 El flujo de compra debe ser guiado paso a paso (catálogo → carrito → saldo → 8 compra).

---

## 6.6. Mantenibilidad y extensibilidad

### ID Requerimiento

RNF1 El código fuente debe estar organizado en paquetes por funcionalidad (**modelos**, 9 **vistas**, etc.).

- RNF2 Las clases deben estar divididas por responsabilidad: activities, modelos y controladores.
- RNF2 El sistema debe ser fácilmente ampliable para futuras funciones como: control de stock automático, lector de código de barras, reportes PDF, etc.
- RNF2 El uso de Firebase permite escalar el sistema sin modificar la arquitectura base.

## **7. Manual de Usuario (basado en layouts existentes)**

---

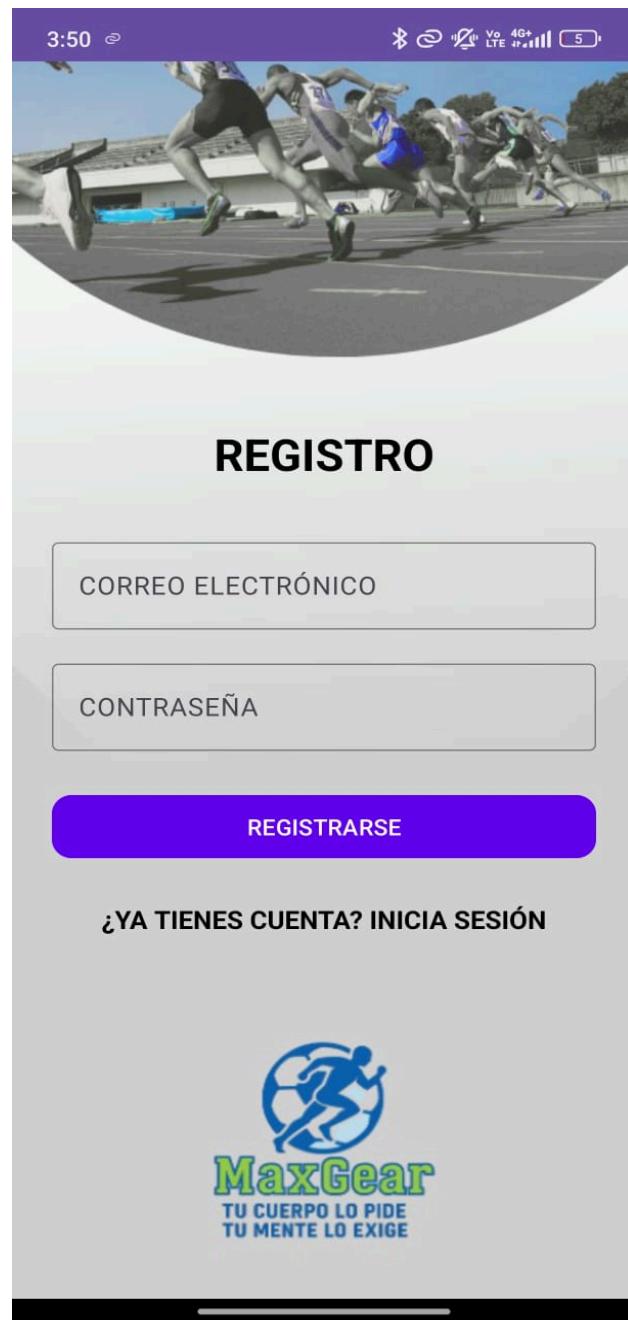
### **7.1. Pantalla de Login**



**Descripción esperada:**

Pantalla de autenticación de usuario. Permite ingresar con un correo electrónico y contraseña previamente registrados. Si el usuario ya está autenticado, se redirige automáticamente al menú principal.

## 7.2. Pantalla de Registro



### Descripción esperada:

Formulario de registro para nuevos usuarios. Solicita un correo electrónico y una contraseña. Al crear la cuenta, se asigna automáticamente el rol "cliente" y se inicializa el saldo en \$0.

### 7.3. Menú Principal (MainActivity)



#### Descripción esperada:

Interfaz principal de la aplicación. Muestra distintos botones según el rol del usuario

autenticado (cliente o administrador). Desde aquí se accede a las funcionalidades principales: catálogo, carrito, saldo, descuentos y cierre de sesión.

---

## 7.4. Vista de Productos (Database.java)



UNIVERSIDAD  
**NACIONAL**  
DE COLOMBIA

### **Descripción esperada:**

Lista de productos obtenidos desde Firebase. Los administradores pueden editar y eliminar productos. Los clientes pueden agregarlos al carrito. Cada tarjeta muestra nombre, precio y stock disponible.

---

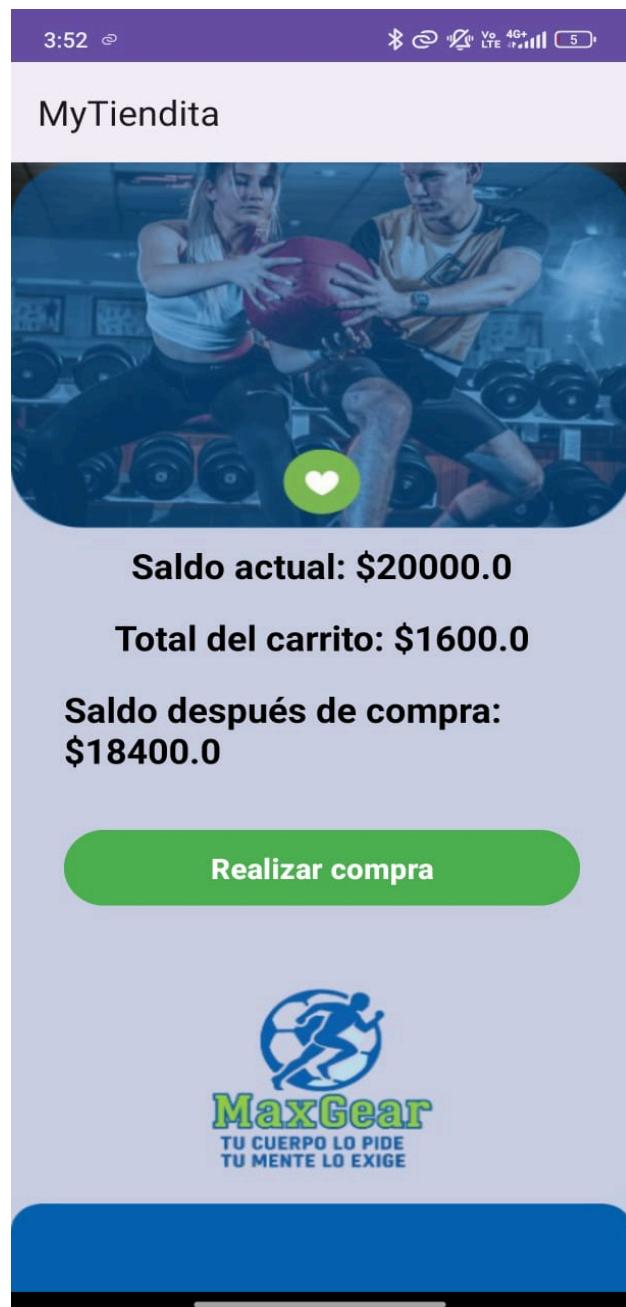
## **7.5. Carrito de Compras**



**Descripción esperada:**

Muestra los productos actualmente en el carrito del usuario. Permite aumentar o disminuir la cantidad de cada producto, eliminar elementos y visualizar el total actualizado automáticamente.

---

**7.6. Confirmación de Compra (Checkout)**

**Descripción esperada:**

Pantalla de resumen previo a la compra. Presenta el saldo actual, el total del carrito y el saldo restante. Valida si el usuario tiene fondos suficientes antes de registrar la venta.

---

## 7.7. Recarga de Saldo

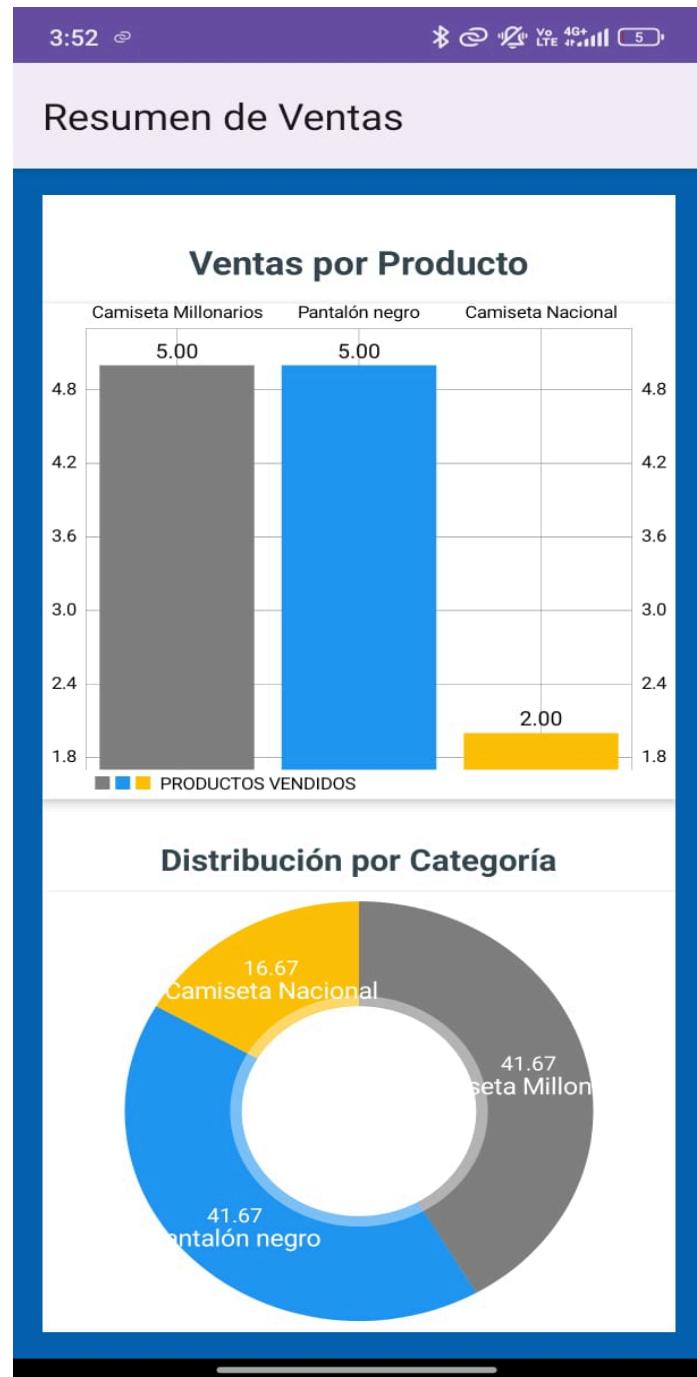


### **Descripción esperada:**

Permite ingresar un monto para simular el pago del cliente. El valor ingresado se suma al saldo actual del usuario y se guarda en Firebase.

---

## **7.8. Resumen de Ventas y Estadísticas**



**Descripción esperada:**

Pantalla de estadísticas visuales. Se muestran gráficas de barras y pastel con base en la cantidad de productos vendidos. También se generan sugerencias de descuento para productos con baja rotación y recomendaciones de reabastecimiento para los más vendidos.

