GOBIERNO DEL
ESTADO DE MÉXICO

UNIVERSIDAD TECNOLÓGICA
DEL VALLE DE TOLUCA

UNIVERSIDAD TECNOLÓGICA DEL VALLE DE TOLUCA

DIRECCIÓN DE CARRERA DE TECNOLOGÍAS DE LA INFORMACIÓN Y COMUNICACIÓN

MATERIA:

DESARROLLO PARA DISPOSITIVOS INTELIGENTES

PROYECTO:

STOCKMASTER

MANUAL TECNICO STOCKMASTER

PRESENTA:

Alvarez Hernández José Saúl
Ricardo Dominguez Osvaldo
Gomez Morales Erick David

JULIO 2023

SANTA MARÍA ATARASQUILLO, LERMA ESTADO DE MÉXICO. 27 DE ENERO DE 2024

Contenido

Introducción

Propósito del Manual

El propósito de este manual técnico es proporcionar una guía detallada para el desarrollo, configuración y mantenimiento de la aplicación **STOCKMASTER**. Está destinado a desarrolladores, administradores del sistema y otros profesionales técnicos que trabajen con la aplicación o necesiten comprender su funcionamiento y estructura. Este documento incluye información sobre la funcionalidad de la aplicación, los componentes clave, y cómo interactuar con los diferentes módulos de la misma.

Descripción General del Proyecto

STOCKMASTER es una aplicación de administración de inventarios diseñada para simplificar la gestión de usuarios y productos. Su objetivo principal es proporcionar una plataforma eficiente para que los administradores gestionen los usuarios de la aplicación y mantengan un control detallado sobre el inventario disponible.

Funcionalidades Principales

1. **Gestión de Usuarios:**
    - **Implementación CRUD:** Los administradores pueden agregar, visualizar, editar y eliminar usuarios en la aplicación mediante múltiples diálogos específicos para estos propósito. Se deben ingresar detalles como nombre, correo electrónico, nombre de usuario, contraseña y rol, claro según el rol que le corresponde es enviado a vistas diferentes que a los administradores les permite visualizar usuarios y productos mientras que a los usuarios solo los productos.

2. **Búsqueda de Usuarios y Productos:**
    - **Filtrado Dinámico:** Los administradores pueden buscar usuarios y productos específicos mediante un campo de búsqueda que filtra la lista en tiempo real según el término ingresado.
3. **Gestión de Productos:**

- **Implementación CRUD:** Los administradores y usuarios pueden agregar, visualizar, editar y eliminar productos en la aplicación mediante múltiples diálogos específicos para estos propósito. Se deben ingresar detalles como imagen del producto, nombre, cantidad de piezas en inventario, el precio de adquisición y precio de venta

4. **Contador de Usuarios y Productos:**
    - **Número de Usuarios y Productos:** Muestra el número total de usuarios y Productos en la base de datos, permitiendo a los administradores tener una visión clara de la base de usuarios y a los usuarios sin permisos de administrados visualizar los productos y la cantidad almacenada en la base de datos.
5. **Cierre de Sesión:**
    - **Logout:** Permite a los usuarios cerrar sesión y redirige al usuario a la pantalla de inicio de sesión.

Tecnologías Utilizadas

- **Android Studio:** Entorno de desarrollo utilizado para construir y mantener la aplicación.
- **Firebase Realtime Database:** Servicio de base de datos en tiempo real utilizado para almacenar y gestionar datos de usuarios y productos.
- **Firebase Storage:** Servicio que se encarga de almacenar las imágenes de los productos y se conecta a su vex con realtime database
- **Dialogos Personalizados:** Para la adición y edición de usuarios y productos.

STOCKMASTER está diseñada para ser intuitiva y fácil de usar, proporcionando a los administradores herramientas efectivas para gestionar tanto los usuarios como el inventario a través de una interfaz amigable y funcionalidades robustas.

Requisitos del Sistema

Hardware

Para ejecutar y desarrollar la aplicación **STOCKMASTER**, se recomienda tener las siguientes especificaciones mínimas de hardware:

- **Procesador:** Intel Core i5 o equivalente
- **Memoria RAM:** 8 GB
- **Espacio en Disco:** 20 GB de espacio libre para almacenamiento de proyecto y archivos temporales
- **Resolución de Pantalla:** Resolución mínima de 1280x800 píxeles
- **Conexión a Internet:** Requerida para la sincronización con Firebase y descarga de dependencias

Requisitos de Software

STOCKMASTER es una aplicación para dispositivos Android y requiere las siguientes versiones de software:

- **Sistema Operativo para Desarrollo:**
  - **Windows:** Windows 10 o superior
  - **macOS:** macOS 10.14 Mojave o superior
  - **Linux:** Distribuciones modernas como Ubuntu 18.04 o superior
- **Entorno de Desarrollo:**
  - **Android Studio:** Versión 4.2 o superior
- **SDK de Android:**
  - **Android SDK:** Versiones 30 (Android 11) o superior
  - **Gradle:** Versión compatible con el Android Studio utilizado
- **Otros Requisitos de Software:**
  - **Java Development Kit (JDK):** Versión 8 o superior (recomendado JDK 11)
  - **Google Services Plugin:** Compatible con la versión de Gradle utilizada

Dependencias

La aplicación **STOCKMASTER** utiliza las siguientes bibliotecas y frameworks para su funcionalidad:

- **Firebase:**
  - **Firebase BOM (Bill of Materials):** com.google.firebase:firebase-bom:33.1.2
    - Permite gestionar versiones de dependencias de Firebase y asegurarse de que todas las bibliotecas de Firebase sean compatibles.
  - **Firebase Realtime Database:** com.google.firebase:firebase-database-ktx
    - Utilizado para la gestión de datos en tiempo real y sincronización de información.
  - **Firebase Storage:** com.google.firebase:firebase-storage
    - Utilizado para el almacenamiento y gestión de archivos como imágenes y documentos en la nube.
- **Bibliotecas para la carga de imágenes:**
  - **Picasso:** com.squareup.picasso:picasso:2.71828
    - Una biblioteca para la carga y visualización eficiente de imágenes en la interfaz de usuario.
  - **Glide:** com.github.bumptech.glide:glide:4.11.0 y com.github.bumptech.glide:glide:4.16.0
    - Otra biblioteca popular para la carga y visualización de imágenes, utilizada en lugar de o junto con Picasso para una integración eficiente de imágenes.
- **Bibliotecas de UI:**
  - **CircleImageView:** de.hdodenhof:circleimageview:3.1.0
    - Utilizada para mostrar imágenes de perfil en forma de círculo, mejorando la estética de la interfaz de usuario.
- **Dependencias de Kotlin:**
  - **Kotlin Standard Library:** org.jetbrains.kotlin:kotlin-stdlib

        - Proporciona funciones básicas de Kotlin necesarias para el desarrollo en este lenguaje.

Estas dependencias deben ser incluidas en el archivo build.gradle del proyecto y sincronizadas adecuadamente para garantizar el correcto funcionamiento de la aplicación.

Configuración del Entorno de Desarrollo

Instalación de Herramientas

1. **Instalación de Android Studio:**
    - **Verificar Instalación:** Asegúrate de tener Android Studio instalado en tu máquina. Si no está instalado, descárgalo desde el sitio oficial de Android Studio e instálalo siguiendo las instrucciones proporcionadas.
    - **Actualizar Android Studio:** Si ya tienes Android Studio, verifica si hay actualizaciones disponibles. Abre Android Studio, ve a Help > Check for Updates (en Windows) o Android Studio > Check for Updates (en macOS), e instala las actualizaciones necesarias.
2. **Configuración de Firebase:**
    - **Verificar Configuración Actual:** Asegúrate de que el proyecto esté correctamente configurado con Firebase. Esto incluye tener el archivo google-services.json en el directorio app del proyecto.
    - **Configuración de Firebase en el Proyecto:** Si Firebase ya está configurado, revisa que todos los servicios necesarios (como Realtime Database, Storage, etc.) estén habilitados en el Firebase Console.

Configuración del Proyecto

Configuración del Proyecto en Android Studio:

- **Revisar Dependencias:**
    - Abre el archivo build.gradle (nivel de aplicación) y asegúrate de que todas las dependencias necesarias estén incluidas. A continuación se muestra un ejemplo de dependencias que deberían estar presentes:

implementation(platform("com.google.firebase:firebase-bom:33.1.2"))
implementation("com.google.firebase:firebase-database-ktx")
implementation("com.google.firebase:firebase-storage")
implementation ("com.squareup.picasso:picasso:2.71828")
implementation("de.hdodenhof:circleimageview:3.1.0")
implementation("com.github.bumptech.glide:glide:4.11.0")
implementation(kotlin("stdlib"))
implementation ("com.github.bumptech.glide:glide:4.16.0")

- Sincroniza el proyecto para asegurarte de que todas las dependencias estén correctamente descargadas y configuradas. Esto se puede hacer desde Android Studio a través de File > Sync Project with Gradle Files.

buildscript {
dependencies {
classpath "com.google.gms:google-services:4.3.15"
}
}

- Asegúrate de que el archivo build.gradle (nivel de aplicación) aplique el plugin de Google Services:

apply plugin: 'com.google.gms.google-services'

- **Configuración de Recursos y Archivos:**
  - **Revisar Archivos de Diseño XML:**
    - Asegúrate de que todos los archivos de diseño XML estén presentes y correctamente configurados en el directorio res/layout. Esto incluye archivos como activity_main.xml, dialog_add_user.xml, etc.
  - **Revisar Recursos de Imágenes:**

- Verifica que todas las imágenes y otros recursos estén correctamente ubicados en el directorio res/drawable. Asegúrate de que no falten archivos importantes.

Verificación de Configuración de Firebase:

- **Inicialización de Firebase:**
  - Verifica que Firebase esté correctamente inicializado en tu código, usualmente en la clase Application o en el onCreate de la actividad principal:

    FirebaseApp.initializeApp(this)

Arquitectura de la Aplicación

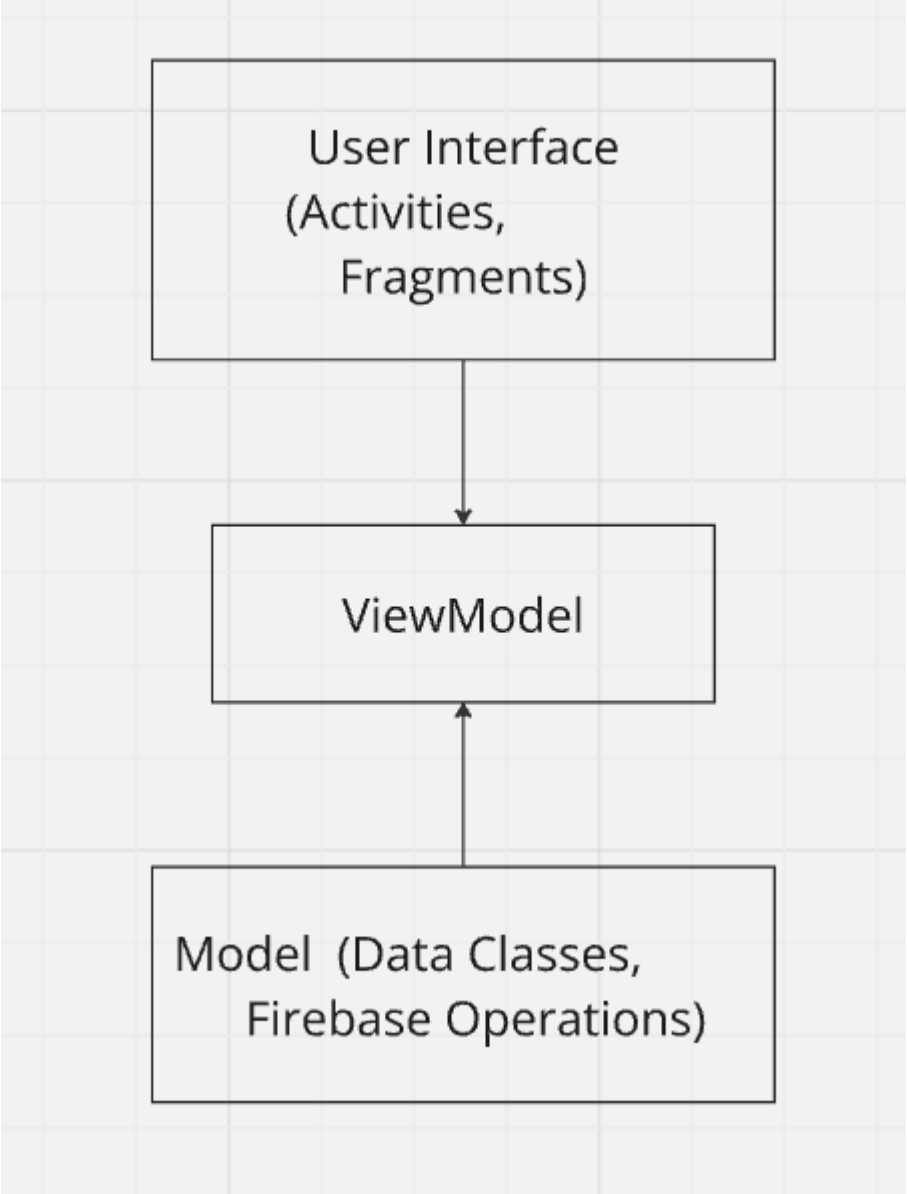
Descripción General

STOCKMASTER utiliza una arquitectura basada en el patrón **MVVM (Model-View-ViewModel)**, que es adecuado para aplicaciones Android debido a su separación clara entre la lógica de negocio, la presentación y la interfaz de usuario.

- **Modelo (Model):** Representa los datos y la lógica de negocio de la aplicación. En STOCKMASTER, el modelo incluye las clases de datos como User, que representan la estructura de los datos en la base de datos, y las operaciones para manipular estos datos (por ejemplo, lectura y escritura en Firebase).
- **Vista (View):** Es la interfaz de usuario que muestra los datos y permite la interacción del usuario. En STOCKMASTER, las vistas están representadas por las actividades (Activity) y fragmentos (Fragment) de la aplicación, que muestran los datos y proporcionan interfaces para que los usuarios interactúen con la aplicación.
- **ViewModel:** Actúa como un intermediario entre el modelo y la vista. En STOCKMASTER, los ViewModel gestionan la lógica de presentación y actualizan la vista con los datos del modelo. Se encargan de la comunicación

con Firebase y de la transformación de los datos del modelo en un formato adecuado para la vista.

Diagrama



User Interface (UI):

- **Activities y Fragments:** Manejan la interacción del usuario y muestran datos. Por ejemplo, UsersAdminActivity, MainActivity, etc.
- **Adapters:** Facilitan la visualización de listas de datos, como UserAdapter, que se utiliza para mostrar la lista de usuarios en un RecyclerView.

ViewModel:

- **UserViewModel:** Gestiona los datos y la lógica de presentación para usuarios. Se comunica con Firebase para obtener y actualizar datos, y notifica a la vista sobre los cambios.

Model:

- **Data Classes:** Representan la estructura de los datos. Por ejemplo, User define los campos del usuario como id, name, email, username, y role.
- **Firebase Operations:** Contienen las operaciones necesarias para interactuar con Firebase, como leer y escribir datos en la base de datos y en el almacenamiento.

Descripción de Componentes

Activities:

1. **ProductsAdminActivity:**
    - **Descripción:** Permite a los administradores gestionar productos en la aplicación. Incluye funcionalidades para agregar, editar, eliminar y buscar productos. Muestra una lista de productos utilizando un RecyclerView.
    - **Funcionalidades:**
        - **Agregar Producto:** Abre un formulario para ingresar detalles del nuevo producto, incluyendo nombre, descripción, precio y una imagen opcional.
        - **Editar Producto:** Permite modificar los detalles de un producto existente, incluyendo la actualización de la imagen del producto.

        - **Eliminar Producto:** Permite eliminar un producto seleccionado de la lista y de la base de datos.
        - **Buscar Producto:** Funcionalidad de búsqueda para filtrar productos según diferentes criterios.
2. **ProductDetailActivity:**
    - **Descripción:** Muestra los detalles completos de un producto seleccionado. Incluye la visualización de la imagen del producto, descripción, precio y otros detalles relevantes.
    - **Funcionalidades:**
        - **Ver Detalles:** Muestra toda la información del producto seleccionado.
        - **Editar o Eliminar:** Opciones para editar o eliminar el producto desde la vista de detalles.
3. **ProductSearchActivity:**
    - **Descripción:** Permite a los usuarios buscar productos específicos en la aplicación. La búsqueda puede realizarse por nombre, categoría u otros filtros.
    - **Funcionalidades:**
        - **Buscar Productos:** Interfaz para ingresar términos de búsqueda y mostrar resultados en una lista.
4. **ProductAddActivity:**
    - **Descripción:** Pantalla para agregar un nuevo producto a la base de datos. Incluye formularios para ingresar todos los detalles del producto.
    - **Funcionalidades:**
        - **Formulario de Adición:** Campos para ingresar nombre, descripción, precio, y cargar una imagen del producto.

Adapters:

1. **ProductAdapter:**

- **Descripción:** Adaptador para el RecyclerView en ProductsAdminActivity y ProductSearchActivity. Gestiona la visualización de la lista de productos.
- **Funcionalidades:**
  - **Visualización de Productos:** Muestra productos en una lista con detalles breves.
  - **Acciones de Edición y Eliminación:** Maneja la interacción del usuario para editar o eliminar productos directamente desde la lista.

Data Classes:

1. **Product:**
   - **Descripción:** Clase que define los datos del producto y su estructura. Incluye campos para almacenar la información del producto en la base de datos.
   - **Campos:**
     - id: Identificador único del producto.
     - name: Nombre del producto.
     - description: Descripción detallada del producto.
     - price: Precio del producto.
     - imageUrl: URL de la imagen del producto (almacenada en Firebase Storage).
     - category: Categoría del producto (opcional).
2. **ProductViewModel:**
   - **Descripción:** Gestiona la lógica de presentación relacionada con los productos. Se encarga de la comunicación con Firebase para obtener y actualizar datos de productos.
   - **Funcionalidades:**
     - **Cargar Productos:** Recupera la lista de productos desde Firebase.
     - **Actualizar Producto:** Actualiza los detalles de un producto en Firebase.

- **Eliminar Producto:** Elimina un producto de la base de datos.

Estos componentes forman la base de la gestión de productos en STOCKMASTER, facilitando la administración y visualización de los productos en la aplicación. Si necesitas ajustes o detalles adicionales, avísame.

Funcionalidades Principales

Administración de Usuarios:

1. **Agregar Usuario:**
    - **Proceso y Lógica:**
        - Los administradores pueden agregar nuevos usuarios a la aplicación a través de la UsersAdminActivity.
        - **Formulario de Registro:** El administrador ingresa los datos del usuario, incluyendo nombre, correo electrónico, nombre de usuario, contraseña y rol (opcional). La contraseña se encripta antes de ser almacenada en Firebase.
        - **Validación:** Se verifica que el correo electrónico no esté ya registrado y que el nombre de usuario sea único. Además, se aseguran de que las contraseñas cumplan con los requisitos de seguridad.
        - **Almacenamiento:** Los datos del nuevo usuario se almacenan en la base de datos de Firebase Realtime Database en la colección users.
2. **Editar Usuario:**
    - **Proceso y Lógica:**
        - La edición de usuarios se realiza en la UsersAdminActivity.
        - **Formulario de Edición:** El administrador selecciona un usuario y puede modificar su nombre, correo electrónico, nombre de usuario y rol. Las contraseñas no se pueden editar directamente; en su lugar, se debe realizar una actualización completa del perfil para cambiarla.

        - **Actualización:** Los cambios se aplican en la base de datos de Firebase Realtime Database. Se valida que el nombre de usuario y el correo electrónico sean únicos y actualizados correctamente.

3. **Eliminar Usuario:**
    - **Proceso y Lógica:**
        - La eliminación de usuarios también se maneja desde la UsersAdminActivity.
        - **Selección y Confirmación:** El administrador selecciona un usuario para eliminar y confirma la acción. La eliminación se realiza mediante un diálogo de confirmación para evitar errores accidentales.
        - **Eliminación:** El usuario seleccionado se elimina de la base de datos de Firebase Realtime Database. Los datos del usuario se eliminan permanentemente.
4. **Búsqueda de Usuarios:**
    - **Proceso y Lógica:**
        - La búsqueda de usuarios se realiza utilizando la funcionalidad de búsqueda en la UsersAdminActivity.
        - **Criterios de Búsqueda:** Los administradores pueden buscar usuarios por nombre, correo electrónico o nombre de usuario. La búsqueda se realiza en tiempo real a medida que el administrador escribe en el campo de búsqueda.
        - **Filtrado:** La lista de usuarios se filtra dinámicamente en función de los criterios de búsqueda, y se actualiza en el RecyclerView para mostrar los resultados relevantes.
5. **Cierre de Sesión:**
    - **Proceso y Lógica:**
        - El cierre de sesión se gestiona desde la LoginActivity o desde una opción de menú en otras actividades.

- **Desconexión:** El usuario actual se desconecta de la sesión activa y se elimina la información de sesión local.
- **Redirección:** Después de cerrar sesión, el usuario es redirigido a la pantalla de inicio de sesión (LoginActivity), donde debe autenticarse nuevamente para acceder a la aplicación.

Administración de Productos:

1. **Agregar Producto:**
    - **Proceso y Lógica:**
        - Los administradores agregan nuevos productos a través de la ProductsAdminActivity.
        - **Formulario de Adición:** El administrador ingresa los detalles del producto, incluyendo nombre, descripción, precio y una imagen opcional.
        - **Validación:** Se asegura que todos los campos requeridos estén completos y que el precio sea válido.
        - **Almacenamiento:** Los detalles del producto se almacenan en la base de datos de Firebase Realtime Database y la imagen en Firebase Storage.
2. **Editar Producto:**
    - **Proceso y Lógica:**
        - La edición de productos se maneja desde la ProductsAdminActivity y ProductDetailActivity.
        - **Formulario de Edición:** El administrador selecciona un producto y puede actualizar sus detalles, incluyendo el nombre, descripción, precio y la imagen del producto.
        - **Actualización:** Los cambios se aplican en la base de datos de Firebase Realtime Database. La imagen se actualiza en Firebase Storage si es necesario.
3. **Eliminar Producto:**
    - **Proceso y Lógica:**

        - La eliminación de productos se realiza desde la ProductsAdminActivity.
        - **Selección y Confirmación:** El administrador selecciona un producto para eliminar y confirma la acción. Se proporciona un diálogo de confirmación para evitar eliminaciones accidentales.
        - **Eliminación:** El producto seleccionado se elimina de la base de datos de Firebase Realtime Database y la imagen asociada se elimina de Firebase Storage.
4. **Búsqueda de Productos:**
    - **Proceso y Lógica:**
        - La búsqueda de productos se realiza en la ProductSearchActivity.
        - **Criterios de Búsqueda:** Los usuarios pueden buscar productos por nombre, categoría u otros filtros.
        - **Filtrado:** La lista de productos se filtra en función de los términos de búsqueda, y los resultados se muestran en un RecyclerView para facilitar la visualización.
5. **Visualización de Productos:**
    - **Proceso y Lógica:**
        - La visualización de productos se realiza en la ProductDetailActivity y otras actividades relacionadas.
        - **Detalles del Producto:** Muestra la información completa del producto seleccionado, incluyendo la imagen, descripción y precio.

Estas funcionalidades permiten una gestión eficaz tanto de usuarios como de productos en STOCKMASTER, asegurando una experiencia de usuario fluida y una administración eficaz.

Interfaz de Usuario

Diseño de la Actividad

1. **MainActivity:**
    - **Descripción de la Interfaz:**
        - **Pantalla Principal:** Muestra la información relevante del usuario actual y las opciones disponibles. Puede incluir un saludo al usuario, el número total de productos en inventario, y botones para acceder a la administración de productos o usuarios, así como una opción para cerrar sesión.
        - **Elementos Clave:**
            - **Texto de Bienvenida:** Un saludo personalizado al usuario.
            - **Número de Productos:** Muestra el conteo actual de productos en el inventario.
            - **Botones de Navegación:** Botones para acceder a ProductsAdminActivity, UsersAdminActivity, y otras funcionalidades.
            - **Botón de Cierre de Sesión:** Permite al usuario cerrar sesión y redirigirse a LoginActivity.
2. **UsersAdminActivity:**
    - **Descripción de la Interfaz:**
        - **Gestión de Usuarios:** Permite a los administradores gestionar usuarios mediante una lista y opciones para agregar, editar y eliminar usuarios.
        - **Elementos Clave:**
            - **Lista de Usuarios:** Muestra los usuarios en un RecyclerView, con opciones de edición y eliminación.
            - **Botón de Agregar Usuario:** Permite abrir un diálogo para agregar un nuevo usuario.

        - **Campo de Búsqueda:** Facilita la búsqueda de usuarios por nombre, correo electrónico o nombre de usuario.
        - **Botones de Acción:** Botones para editar o eliminar usuarios seleccionados.

3. **ProductAdminActivity:**
    - **Descripción de la Interfaz:**
        - **Gestión de Productos:** Permite a los administradores agregar, editar y eliminar productos del inventario.
        - **Elementos Clave:**
            - **Lista de Productos:** Muestra los productos en un RecyclerView, con opciones de edición y eliminación.
            - **Botón de Agregar Producto:** Permite abrir un diálogo para agregar un nuevo producto.
            - **Campo de Búsqueda:** Facilita la búsqueda de productos por nombre, categoría u otros filtros.
            - **Botones de Acción:** Botones para editar o eliminar productos seleccionados.
4. **EditProfileActivity:**
    - **Descripción de la Interfaz:**
        - **Gestión de Perfil del Usuario:** Permite a los usuarios modificar su perfil personal, agregar una imagen, cambiar contraseña, correo y nombre de usuario.
        - **Elementos Clave:**
            - **Imagen de perfil:** Muestra una foto de perfil que el usuario asigno, en caso de que no quiera se muestra una imagen por default.
            - **Email, Name y Username:** Permite visualizar los datos del usuario que inicio sesion.
            - **Password y Repeat Password:** Permite cambiar la constraseña para iniciar sesión y la envia hasheada de nuevo a firebase

Diálogos

1. **Agregar/Editar Usuario:**
    - **Descripción:**
        - **Diálogo de Agregar Usuario:** Permite ingresar los datos necesarios para registrar un nuevo usuario, incluyendo nombre, correo electrónico, nombre de usuario, contraseña y rol.
        - **Diálogo de Editar Usuario:** Permite modificar los detalles de un usuario existente. Incluye campos similares al diálogo de agregar, pero con datos precargados.
    - **Elementos Clave:**
        - **Campos de Entrada:** Para ingresar y editar nombre, correo electrónico, nombre de usuario, y rol.
        - **Botones de Confirmación y Cancelación:** Botones para guardar los cambios o cancelar la acción.
2. **Agregar/Editar Producto:**
    - **Descripción:**
        - **Diálogo de Agregar Producto:** Permite ingresar los detalles del producto, incluyendo nombre, descripción, precio y una imagen opcional.
        - **Diálogo de Editar Producto:** Permite modificar los detalles de un producto existente. Incluye campos precargados con los datos actuales del producto.
    - **Elementos Clave:**
        - **Campos de Entrada:** Para ingresar y editar nombre, descripción, precio y cargar una imagen del producto.
        - **Botones de Confirmación y Cancelación:** Botones para guardar los cambios o cancelar la acción.

Layout XML

1. **activity_main.xml:**

- **Descripción:**
  - **Diseño de Pantalla Principal:** Incluye un saludo al usuario, el número total de productos, y botones de navegación.
- **Elementos Clave:**
  - **TextView:** Para mostrar el saludo y el número de productos.
  - **Button:** Para navegar a ProductsAdminActivity, UsersAdminActivity, y cerrar sesión.

2. **activity_users_admin.xml:**
   - **Descripción:**
     - **Diseño de Gestión de Usuarios:** Incluye una lista de usuarios, campo de búsqueda, y botones para agregar usuarios.
   - **Elementos Clave:**
     - **RecyclerView:** Para mostrar la lista de usuarios.
     - **SearchView:** Para la búsqueda de usuarios.
     - **Button:** Para agregar un nuevo usuario.
3. **activity_product_admin.xml:**
   - **Descripción:**
     - **Diseño de Gestión de Productos:** Incluye una lista de productos, campo de búsqueda, y botones para agregar productos.
   - **Elementos Clave:**
     - **RecyclerView:** Para mostrar la lista de productos.
     - **SearchView:** Para la búsqueda de productos.
     - **Button:** Para agregar un nuevo producto.
4. **Diálogos (XML):**
   - **dialog_add_user.xml:**
     - **Diseño del Diálogo para Agregar/Edit Usuario:** Incluye campos de entrada para nombre, correo electrónico, nombre de usuario, contraseña y rol.
   - **dialog_add_product.xml:**

- **Diseño del Diálogo para Agregar/Edit Producto:** Incluye campos de entrada para nombre, descripción, precio y una opción para cargar una imagen.

Integración con Firebase

STOCKMASTER utiliza Firebase para gestionar usuarios y productos. A continuación, se describen los detalles de la estructura de la base de datos, las operaciones básicas y las reglas de seguridad implementadas.

Estructura de la Base de Datos

La estructura de la base de datos en Firebase está organizada en dos nodos principales: users y products.

- **Nodo users**: Almacena la información de los usuarios registrados. Cada usuario tiene un identificador único (userId), que se genera automáticamente por Firebase. La estructura del nodo users es la siguiente:

Nodo products: Contiene los datos de los productos en inventario. Cada producto también tiene un identificador único (productId). La estructura del nodo products es la siguiente:

Operaciones con Firebase

1. Lectura de Datos

Para leer datos de Firebase, se utiliza la referencia a los nodos de la base de datos y se manejan las respuestas mediante listeners.

- **Lectura de Usuarios:**

```
val database = FirebaseDatabase.getInstance().reference
val usersRef = database.child("users")

// Leer un usuario específico por ID
usersRef.child(userId).get().addOnSuccessListener { snapshot ->
    val user = snapshot.getValue(User::class.java)
    // Procesar los datos del usuario
}.addOnFailureListener {
    // Manejar error de lectura
}
```

En este ejemplo, se accede al nodo users y se obtiene un usuario específico por su userId. La respuesta se maneja en el listener addOnSuccessListener.

- **Lectura de Productos:**

```
val productsRef = database.child("products")

// Leer todos los productos
productsRef.get().addOnSuccessListener { snapshot ->
    val products = snapshot.children.mapNotNull { it.getValue(Product::class.java) }
    // Procesar la lista de productos
}.addOnFailureListener {
    // Manejar error de lectura
}
```

Este código lee todos los productos en el nodo products y convierte los datos en una lista de objetos Product.

2. Escritura de Datos

Para escribir o actualizar datos en Firebase, se utilizan métodos de escritura en la referencia correspondiente.

- **Agregar Usuario:**

```
val newUser = User(name = "John Doe", email = "john@example.com", username = "johndoe"
val userId = database.child("users").push().key
userId?.let {
    database.child("users").child(it).setValue(newUser).addOnSuccessListener {
        // Usuario agregado exitosamente
    }.addOnFailureListener {
        // Manejar error de escritura
    }
}
```

Aquí, se crea un nuevo usuario y se agrega al nodo users. El identificador del usuario se genera automáticamente mediante push().key.

- **Actualizar Producto**:

```
val updatedProduct = Product(name = "New Product", description = "Updated description"
val productId = "productId123" // ID del producto que se va a actualizar
database.child("products").child(productId).setValue(updatedProduct).addOnSuccessListe
    // Producto actualizado exitosamente
}.addOnFailureListener {
    // Manejar error de actualización
}
```

3. Seguridad y Reglas

Para proteger los datos en Firebase, se implementan reglas de seguridad que definen quién puede leer y escribir en la base de datos.

- **Reglas de Seguridad:**

```
{
  "rules": {
    ".read": "now < 1723788000000",  // 2024-8-16
    ".write": "now < 1723788000000",  // 2024-8-16
  }
}
```

- **Reglas Generales**: La lectura y escritura en la base de datos están permitidas solo para usuarios autenticados (auth != null).
- **Reglas para users**: Los usuarios pueden leer y escribir solo su propio nodo de usuario ($userId === auth.uid).
- **Reglas para products**: Los productos pueden ser leídos y escritos por cualquier usuario autenticado.

Estas reglas aseguran que solo los usuarios autorizados puedan acceder y modificar los datos de acuerdo con sus permisos.

Seguridad

La seguridad es un aspecto crucial en **STOCKMASTER**. A continuación, se detallan los métodos utilizados para asegurar las contraseñas y las validaciones de entrada para los datos del usuario.

Hashing de Contraseña

Para proteger las contraseñas de los usuarios, **STOCKMASTER** utiliza el hashing antes de almacenarlas en la base de datos. Esto asegura que incluso si la base de datos es comprometida, las contraseñas de los usuarios no se exponen en texto claro.

```
fun hashPassword(password: String): String {
    return try {
        val digest = MessageDigest.getInstance("SHA-256")
        val hashBytes = digest.digest(password.toByteArray(StandardCharsets.UTF_8))
        Base64.encodeToString(hashBytes, Base64.DEFAULT).trim()
    } catch (e: NoSuchAlgorithmException) {
        e.printStackTrace()
        ""
    }
}
```

Este método utiliza el algoritmo SHA-256 para generar un hash seguro de la contraseña. El hash se codifica en Base64 para facilitar su almacenamiento.

Al crear un nuevo usuario, la contraseña se hashea utilizando el método hashPassword antes de ser almacenada en Firebase.

Validaciones de Entrada

Para asegurar la integridad de los datos ingresados por el usuario, **STOCKMASTER** implementa varias validaciones en el lado del cliente. Estas validaciones ayudan a prevenir errores y a asegurar que los datos sean correctos y seguros antes de ser procesados o almacenados.

```
fun isValidEmail(email: String): Boolean {
    return email.contains("@") && Patterns.EMAIL_ADDRESS.matcher(email).matches()
}

fun isValidPassword(password: String): Boolean {
    return password.length >= 8
}

fun passwordsMatch(password: String, confirmPassword: String): Boolean {
    return password == confirmPassword
}

fun isUniqueUsername(username: String, existingUsernames: List<String>): Boolean {
    return !existingUsernames.contains(username)
}
```

Uso de las validaciones en el proceso de registro:

```
val email = "john@example.com"
val password = "userpassword"
val confirmPassword = "userpassword"
val username = "johndoe"

if (isValidEmail(email) && isValidPassword(password) && passwordsMatch(password, confirmPassword)) {
    // Verificar si el nombre de usuario es único
    val existingUsernames = listOf("existingUser1", "existingUser2") // Simulación de nombres de usuario existentes
    if (isUniqueUsername(username, existingUsernames)) {
        val newUser = User(
            name = "John Doe",
            email = email,
            username = username,
            password = hashPassword(password),
            role = "user"
        )
        // Proceder con el registro del usuario
    } else {
        // Mostrar error de nombre de usuario duplicado
    }
} else {
    // Mostrar mensajes de error de validación
}
```

En este ejemplo, se validan el correo electrónico, la contraseña y la confirmación de la contraseña. También se verifica que el nombre de usuario sea único antes de proceder con el registro.

Estas validaciones ayudan a garantizar que los datos ingresados por el usuario cumplan con los criterios esperados, mejorando la seguridad y la usabilidad de la aplicación.

Manejo de Errores

El manejo de errores es una parte crucial en el desarrollo de cualquier aplicación, ya que ayuda a mejorar la experiencia del usuario y facilita la depuración y el mantenimiento del código. A continuación, se detallan los errores comunes y sus soluciones, así como el manejo de excepciones en **STOCKMASTER**.

Errores Comunes

1. **Error de autenticación**: Cuando las credenciales del usuario son incorrectas o no existen en la base de datos.
    - **Solución**: Mostrar un mensaje de error utilizando un Toast.
2. **Error de conexión**: Problemas de conectividad con Firebase.
    - **Solución**: Implementar un manejo de excepciones y mostrar mensajes apropiados al usuario.

3. **Error de validación de entrada**: Datos ingresados por el usuario que no cumplen con los criterios de validación.
    - **Solución**: Mostrar mensajes de error claros y específicos utilizando Toast o validaciones en el lado del cliente.
4. **Error al agregar productos**: Problemas al intentar agregar un producto a la base de datos.
    - **Solución**: Manejar excepciones y proporcionar mensajes de retroalimentación al usuario.
5. **Error al actualizar productos**: Problemas al intentar actualizar la información de un producto.
    - **Solución**: Manejar excepciones y proporcionar mensajes de retroalimentación al usuario.
6. **Error al eliminar productos**: Problemas al intentar eliminar un producto de la base de datos.
    - **Solución**: Manejar excepciones y proporcionar mensajes de retroalimentación al usuario.

Manejo de Excepciones

Autenticación de Usuario:

```
private fun loginUser(email: String, password: String) {
    firebaseAuth.signInWithEmailAndPassword(email, password)
        .addOnCompleteListener(this) { task ->
            if (task.isSuccessful) {
                // Login successful
                val user = firebaseAuth.currentUser
                updateUI(user)
            } else {
                // Login failed
                Log.e("LoginActivity", "signInWithEmail:failure", task.exception)
                Toast.makeText(this, "Authentication failed.", Toast.LENGTH_SHORT).show()
                updateUI(null)
            }
        }
}
```

En este ejemplo, si el inicio de sesión falla, se registra el error utilizando Log.e y se muestra un Toast con un mensaje de error.

Lectura de Datos desde Firebase:

```
private fun fetchDataFromFirebase() {
    val databaseReference = FirebaseDatabase.getInstance().getReference("users")
    databaseReference.addValueEventListener(object : ValueEventListener {
        override fun onDataChange(dataSnapshot: DataSnapshot) {
            if (dataSnapshot.exists()) {
                // Data fetch successful
                val users = mutableListOf<User>()
                for (userSnapshot in dataSnapshot.children) {
                    val user = userSnapshot.getValue(User::class.java)
                    users.add(user!!)
                }
                updateUserList(users)
            } else {
                // No data available
                Log.e("MainActivity", "No data available")
                Toast.makeText(this@MainActivity, "No data available.", Toast.LENGTH_SHORT).show()
            }
        }

        override fun onCancelled(databaseError: DatabaseError) {
            // Failed to read data
            Log.e("MainActivity", "Failed to read data", databaseError.toException())
            Toast.makeText(this@MainActivity, "Failed to read data.", Toast.LENGTH_SHORT).show()
        }
    })
}
```

Aquí, si la lectura de datos falla, se maneja el error utilizando Log.e y Toast para informar al usuario.

Escritura de Datos en Firebase:

```
private fun addUserToFirebase(user: User) {
    val databaseReference = FirebaseDatabase.getInstance().getReference("users")
    val userId = databaseReference.push().key ?: return
    databaseReference.child(userId).setValue(user)
        .addOnCompleteListener { task ->
            if (task.isSuccessful) {
                Toast.makeText(this, "User added successfully.", Toast.LENGTH_SHORT).show()
            } else {
                Log.e("UsersAdminActivity", "Failed to add user", task.exception)
                Toast.makeText(this, "Failed to add user.", Toast.LENGTH_SHORT).show()
            }
        }
}
```

En este caso, si la escritura de datos falla, se registra el error utilizando Log.e y se muestra un mensaje de error con Toast.

Actualización de Datos en Firebase:

```
private fun updateUserInFirebase(user: User) {
    val databaseReference = FirebaseDatabase.getInstance().getReference("users").child(user.id)
    databaseReference.setValue(user)
        .addOnCompleteListener { task ->
            if (task.isSuccessful) {
                Toast.makeText(this, "User updated successfully.", Toast.LENGTH_SHORT).show()
            } else {
                Log.e("UsersAdminActivity", "Failed to update user", task.exception)
                Toast.makeText(this, "Failed to update user.", Toast.LENGTH_SHORT).show()
            }
        }
}
```

Este código maneja la actualización de datos de un usuario. Si la operación falla, se registra el error y se muestra un mensaje de error.

Eliminación de Datos en Firebase:

```
private fun deleteUserFromFirebase(userId: String) {
    val databaseReference = FirebaseDatabase.getInstance().getReference("users").child(userId)
    databaseReference.removeValue()
        .addOnCompleteListener { task ->
            if (task.isSuccessful) {
                Toast.makeText(this, "User deleted successfully.", Toast.LENGTH_SHORT).show()
            } else {
                Log.e("UsersAdminActivity", "Failed to delete user", task.exception)
                Toast.makeText(this, "Failed to delete user.", Toast.LENGTH_SHORT).show()
            }
        }
}
```

En este ejemplo, se maneja la eliminación de un usuario. Si la operación falla, se registra el error y se muestra un mensaje de error.

Validaciones de Entrada:

```
private fun validateUserData(email: String, password: String, confirmPassword: String): Boolean {
    return when {
        email.isBlank() -> {
            Toast.makeText(this, "Email is required.", Toast.LENGTH_SHORT).show()
            false
        }
        !Patterns.EMAIL_ADDRESS.matcher(email).matches() -> {
            Toast.makeText(this, "Enter a valid email.", Toast.LENGTH_SHORT).show()
            false
        }
        password.isBlank() -> {
            Toast.makeText(this, "Password is required.", Toast.LENGTH_SHORT).show()
            false
        }
        password.length < 8 -> {
            Toast.makeText(this, "Password must be at least 8 characters long.", Toast.LENGTH_SHORT).show()
            false
        }
        password != confirmPassword -> {
            Toast.makeText(this, "Passwords do not match.", Toast.LENGTH_SHORT).show()
            false
        }
        else -> true
    }
}
```

Aquí se implementan varias validaciones para los datos de entrada del usuario y se muestran mensajes de error específicos utilizando Toast.

Manejo de Excepciones en el Registro de Usuarios:

```
private fun registerUser(email: String, password: String, username: String) {
    if (!validateUserData(email, password, password)) {
        return
    }
    firebaseAuth.createUserWithEmailAndPassword(email, password)
        .addOnCompleteListener(this) { task ->
            if (task.isSuccessful) {
                // Registration successful
                val firebaseUser = firebaseAuth.currentUser
                val user = User(firebaseUser!!.uid, username, email, password, "user")
                addUserToFirebase(user)
                updateUI(firebaseUser)
            } else {
                // Registration failed
                Log.e("SignUpActivity", "createUserWithEmail:failure", task.exception)
                Toast.makeText(this, "Registration failed: ${task.exception?.message}", Toast.LENGTH_SHORT).show()
            }
        }
}
```

Este código maneja el registro de un nuevo usuario. Si el registro falla, se registra el error y se muestra un mensaje de error específico.

Manejo de Errores en la Carga de Imágenes:

```
private fun uploadImageToFirebase(uri: Uri) {
    val storageReference = FirebaseStorage.getInstance().reference.child("images/${UUID.randomUUID()}")
    storageReference.putFile(uri)
        .addOnSuccessListener {
            Toast.makeText(this, "Image uploaded successfully.", Toast.LENGTH_SHORT).show()
        }
        .addOnFailureListener { exception ->
            Log.e("ProductAdminActivity", "Failed to upload image", exception)
            Toast.makeText(this, "Failed to upload image: ${exception.message}", Toast.LENGTH_SHORT).show()
        }
}
```

Aquí, si la carga de la imagen falla, se registra el error y se muestra un mensaje de error específico utilizando Toast.

Anexos

Referencias

A continuación se listan las referencias y recursos utilizados durante el desarrollo y documentación de **STOCKMASTER**:

- **Firebase Documentation**: Proporciona guías y referencias detalladas sobre el uso de Firebase Realtime Database, Authentication, y Storage.
    - Firebase Realtime Database Documentation

    - Firebase Storage Documentation
- **Android Developers Documentation**: Incluye guías y referencias sobre el desarrollo de aplicaciones Android, uso de RecyclerView, y otros componentes de la interfaz de usuario.
    - Android Developers
    - RecyclerView
- **Picasso Documentation**: Biblioteca utilizada para la carga y manejo de imágenes.
    - Picasso GitHub
- **Glide Documentation**: Biblioteca utilizada para la carga y manejo de imágenes.
    - Glide Documentation

Glosario

- **Activity**: Componente principal de una aplicación Android que proporciona una pantalla con la cual los usuarios pueden interactuar para realizar una acción.
- **Adapter**: Componente que actúa como un puente entre una fuente de datos y la vista de la interfaz de usuario, como un RecyclerView.
- **Firebase**: Plataforma de desarrollo de aplicaciones móviles y web de Google que incluye servicios como bases de datos en tiempo real, autenticación, almacenamiento, entre otros.
- **RecyclerView**: Componente de la interfaz de usuario en Android que permite mostrar una gran cantidad de datos de manera eficiente y personalizable.
- **SDK (Software Development Kit)**: Conjunto de herramientas de desarrollo de software que permite a los desarrolladores crear aplicaciones para una plataforma específica.
- **Hashing**: Proceso de transformar una entrada (como una contraseña) en una cadena fija de caracteres, típicamente para fines de seguridad.
- **Toast**: Mensaje breve que aparece en la pantalla para proporcionar retroalimentación al usuario.

- **Unit Test (Prueba Unitaria)**: Prueba que verifica el correcto funcionamiento de una unidad de código de manera aislada.
- **Integration Test (Prueba de Integración)**: Prueba que verifica la interacción correcta entre diferentes módulos o componentes del sistema.
- **MVVM (Model-View-ViewModel)**: Patrón arquitectónico que separa el desarrollo de la interfaz de usuario (View) del desarrollo de la lógica de negocio o lógica de la aplicación (Model) mediante la introducción de un componente intermedio: el ViewModel.
- **Lombok**: Biblioteca para Java que permite reducir el código boilerplate, facilitando la generación automática de getters, setters, constructores, entre otros.
- **Glide**: Biblioteca de carga de imágenes para Android, que maneja eficientemente la descarga y el cache de imágenes.
- **Picasso**: Biblioteca de manejo de imágenes para Android, que facilita la carga y manipulación de imágenes.