Documentación Técnica del Proyecto de Aplicación Móvil y Web con Firebase y Flutter

# 1. Tecnologías Utilizadas

* **Flutter**: Framework para el desarrollo de aplicaciones móviles y web con un solo código base. Utiliza Dart como lenguaje de programación.
* **Firebase Authentication**: Para la autenticación de usuarios, soportando múltiples métodos como email/password.
* **Firebase Firestore**: Base de datos NoSQL que será utilizada para almacenar y gestionar datos de inventarios, productos y usuarios mediante operaciones CRUD.
* **Arquitectura Modular en Flutter**: El proyecto sigue un patrón de arquitectura modular, donde los directorios y componentes de la aplicación están organizados de manera lógica y funcional. Esto facilita la escalabilidad, el mantenimiento y la reutilización de código.

# 2. Arquitectura de Firebase

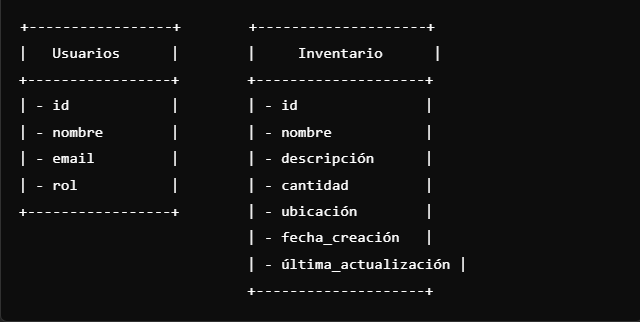
**2.1 Firebase Authentication**

* **Autenticación por Email/Contraseña**: El sistema permite a los usuarios autenticarse mediante credenciales tradicionales. Las credenciales se envían a Firebase Auth, que valida las mismas y genera un token de sesión para cada usuario.
* **Flujo de Autenticación**:
  1. El usuario se registra o inicia sesión desde la aplicación móvil o web.
  2. El token de sesión generado es almacenado localmente para validar las solicitudes futuras.
  3. Al cerrar sesión, el token se elimina, forzando la autenticación en el siguiente acceso.

# 2.2 Firebase Firestore

* **Base de Datos NoSQL**:
  + La estructura de la base de datos es flexible, lo que permite almacenar colecciones y documentos sin la rigidez de las bases de datos relacionales.
  + Los **inventarios** serán almacenados como documentos en una colección llamada Inventarios, donde cada documento representa un producto con sus detalles.
  + **Usuarios**: Se crea una colección llamada Usuarios donde se almacenan detalles como el ID de usuario, nombre y correo electrónico.
* **CRUD (Create, Read, Update, Delete)**:
  + **Crear**: Los usuarios pueden añadir nuevos productos al inventario desde la aplicación móvil o web.
  + **Leer**: Se pueden obtener todos los productos almacenados, filtrados o no, en tiempo real.
  + **Actualizar**: Los usuarios pueden modificar los detalles de los productos existentes.
  + **Eliminar**: Los productos pueden ser eliminados de la base de datos.
* **Seguridad**: Las reglas de seguridad de Firestore están configuradas para permitir acceso solo a usuarios autenticados.

# Modelo Lógico de la Base de Datos



# 4. Arquitectura Modular en Flutter

El proyecto está estructurado siguiendo una arquitectura modular, donde cada funcionalidad se organiza de forma independiente en módulos separados. Esto sigue las buenas prácticas de desarrollo para proyectos Flutter, donde el código es desacoplado y fácilmente mantenible.

# 4.1 Estructura de Directorios:

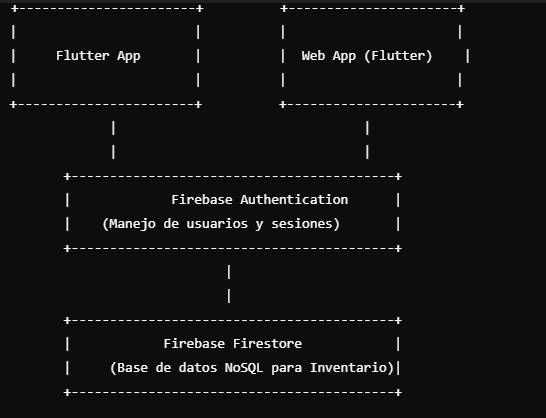
* **views/screen/**: Aquí se encuentran todas las pantallas principales de la aplicación, organizadas en subdirectorios según la funcionalidad (por ejemplo, inventario\_screen, scanner\_screen, etc.).
* **views/widgets/**: Componentes reutilizables como botones, formularios y barras de navegación que pueden ser utilizados en diferentes pantallas.
* **business\_logic/view\_model/**: Contiene los ViewModels que gestionan la lógica de negocio, siguiendo el patrón **MVVM** (Model-View-ViewModel).
* **business\_logic/service/**: Aquí se encuentran los servicios que interactúan con Firebase para la autenticación y operaciones de Firestore.
* **business\_logic/actions/**: Contiene las acciones y casos de uso (use cases) que implementan la lógica más compleja del negocio.

# 4.2 Patrón de Arquitectura: MVVM

Este proyecto utiliza el patrón **MVVM** (Modelo-Vista-ViewModel) en Flutter. Este patrón separa la lógica de la interfaz de usuario de la lógica del negocio, facilitando el testing y el mantenimiento.

* **Modelo (Model)**: Representa los datos y la lógica de acceso a la base de datos (Firestore).
* **Vista (View)**: Representa las pantallas o componentes visuales de la aplicación.
* **ViewModel**: Maneja la lógica de la interfaz de usuario y comunica los datos entre la vista y el modelo.

# Diagramas de Arquitectura



****

# 5. Casos de Uso

**Caso de Uso 1: Autenticación de Usuarios**

* **Descripción**: Los usuarios pueden autenticarse en la aplicación mediante email y contraseña.
* **Actores**: Usuario.
* **Precondiciones**: El usuario debe tener una cuenta registrada.
* **Postcondiciones**: El usuario obtiene acceso a las funcionalidades de la app tras la autenticación exitosa.

**Caso de Uso 2: Gestión de Inventarios**

* **Descripción**: Los usuarios pueden agregar, modificar y eliminar productos en el inventario.
* **Actores**: Usuario autenticado.
* **Precondiciones**: El usuario debe estar autenticado y tener los permisos correspondientes.
* **Postcondiciones**: Los cambios se reflejan en la base de datos Firestore.

**6. Requerimientos Funcionales**

1. El sistema debe permitir el registro y autenticación de usuarios mediante email y contraseña.
2. Los usuarios autenticados deben poder realizar operaciones CRUD en los productos del inventario.
3. La aplicación debe mostrar el estado en tiempo real de los inventarios almacenados en Firestore.

**7. Requerimientos No Funcionales**

1. **Seguridad**: La autenticación y autorización deben garantizar que solo los usuarios válidos accedan a los recursos de la aplicación.
2. **Escalabilidad**: El sistema debe ser capaz de soportar múltiples usuarios concurrentes sin degradación del rendimiento.
3. **Usabilidad**: La interfaz debe ser intuitiva y fácil de usar, tanto en la aplicación móvil como en la web.