Programación de Aplicaciones Móviles Nativas

Diseño de la Base de Datos

Raúl Cruz Ortega
Laura González Suárez
Ismael Ramos Alonso

Introducción

En el contexto del desarrollo de aplicaciones móviles, la elección y diseño de la base de datos son factores determinantes para el éxito de un proyecto. En este informe, abordaremos la tarea de seleccionar la tecnología de base de datos más apropiada para nuestra aplicación de compra y venta de artesanía. Consideraremos los modelos y componentes relacionados con la capa de persistencia, justificaremos la elección de la tecnología y definiremos cómo integrar la lógica de la base de datos en nuestra aplicación.
Además, también se anticipan posibles cambios futuros, examinando la posibilidad de cambiar el motor de la base de datos, identificando las partes de la aplicación que requerirían modificaciones, considerando las dificultades que podrían surgir y proponiendo estrategias para minimizar el impacto de tales cambios.

1. Revisión de la arquitectura del proyecto final

En nuestra arquitectura MVVM (Model-View-ViewModel) diseñada para la aplicación de venta de artesanía, la capa de persistencia desempeña un papel crítico en la gestión de datos y en la interacción con la base de datos. Esta capa se encarga de asegurar la disponibilidad y el almacenamiento eficiente de los datos, constituyendo un pilar esencial para el correcto funcionamiento de la aplicación. Por ello, hemos identificado y resaltado los siguientes modelos y componentes relacionados con la capa de persistencia:

Modelos de Datos (Model): En esta capa, destacamos la presencia de modelos de datos que representan las estructuras fundamentales de la información utilizada en la aplicación. Estos modelos incluyen elementos como "Producto", "Cliente", "Pedido", entre otros. Cada modelo define las características específicas de los datos, como nombre, precio, descripción y otros atributos relevantes.

Servicios de Base de Datos: La capa de persistencia también incorpora servicios de base de datos que facilitan la comunicación con el sistema de almacenamiento subyacente. Estos servicios son esenciales para gestionar la conexión con la base de datos, ejecutar operaciones de lectura y escritura de datos, y proporcionar métodos que permiten acceder y manipular la información de manera eficiente. A través de este enfoque, estamos aislando el modelo de la base de datos, de modo que si en el futuro deseamos cambiar de plataforma de base de datos, no será necesario modificar el modelo de datos ni los servicios de la capa de persistencia.

2. Selección de tecnología para la base de datos: Firebase y SQLite

En el proceso de elección de la tecnología para la base de datos de nuestra aplicación, hemos adoptado una estrategia que combina dos tecnologías esenciales: Firebase y SQLite. Esta selección se fundamenta en la necesidad de abordar distintos aspectos de almacenamiento de datos de manera eficaz y eficiente.

a. Firebase para Datos Comunes y Actualización en Tiempo Real:

Hemos decidido aprovechar Firebase, una plataforma integral de desarrollo de aplicaciones creada por Google, por varias razones fundamentales:

- Bases de Datos en Tiempo Real: Firebase proporciona una base de datos en tiempo real que se adapta perfectamente a las necesidades de nuestra aplicación de compra y venta de productos. Esto nos permite mostrar información actualizada de productos y ofertas a todos los usuarios en tiempo real.
- Herramientas de autenticación: Firebase ofrece servicios de autenticación que simplifican el proceso de registro e inicio de sesión de los usuarios.
- Almacenamiento en la Nube: Utilizaremos el almacenamiento en la nube de Firebase para guardar datos compartidos y recursos multimedia, lo que garantiza la disponibilidad y accesibilidad de estos elementos para todos los usuarios.
- Escalabilidad automática: Firebase escala automáticamente, lo que significa que si tu aplicación de compra-venta crece, no tendrás que preocuparte por configuraciones adicionales.
- Desarrollo ágil: Firebase proporciona una solución de back-end sin servidor, permitiendo centrarse más en las características del front-end.
- Integración con otras herramientas de Google: Facilita el análisis del comportamiento del usuario, monetización y otros aspectos mediante herramientas como Google Analytics.

b. SQLite para Datos Personales y Almacenamiento Local:

En complemento a Firebase, hemos optado por integrar SQLite como nuestra base de datos local. Esta elección se basa en la necesidad de gestionar los datos personales del usuario, como el almacenamiento de sus artículos favoritos y preferencias. Esto garantiza la privacidad y disponibilidad de estos datos sin depender de la conectividad en línea.

La combinación de Firebase y SQLite nos permite ofrecer una experiencia de usuario completa. Firebase se encargará de manejar datos compartidos y actualizaciones en tiempo real, mientras que SQLite gestionará de manera eficiente la persistencia de datos personales de cada usuario de forma local.

3. Integración de la lógica que gestionará la capa de persistencia

Módulo de Persistencia:
Dentro de este módulo, se incluirán las clases y componentes relacionados con la capa de persistencia, garantizando que la gestión de datos se mantenga organizada y aislada del resto de la aplicación.

Entidades y atributos en Firebase:
En Firebase, las entidades servirán como representaciones de los objetos de alto nivel en nuestra base de datos en tiempo real. En nuestro proyecto, se han definido las siguientes entidades que se sincronizarán en tiempo real con la base de datos de Firebase:

- **Usuario:** Representa la entidad de usuario, con atributos como nombre, dirección de correo electrónico, contraseña, etc.
- **Artesano:** Representa la entidad de artesano, con atributos como nombre, especialidad, correo electrónico, descripción, etc.
- **Producto:** Representa la entidad de producto, con atributos como nombre del producto, precio, imagen, descripción, y una referencia al artesano que lo creó.
- **Encargo:** Representa la entidad de encargo, con atributos como detalles del encargo, estado, fecha, y una referencia al usuario que realizó el encargo y al producto relacionado.

Entidades y atributos en SQLite:
En SQLite, se utilizarán entidades para modelar los datos personales del usuario, centrándose en aspectos clave como "Artículos Favoritos." Estos datos serán almacenados localmente en el dispositivo del usuario, garantizando la privacidad y el acceso a la información personal.

El atributo de “artículos favoritos” del usuario será “Id del producto”

Relaciones en Firebase y SQLite:
Las relaciones en Firebase se establecen entre las siguientes entidades:

- **Usuario -> Artesano:** Cada usuario puede también ser un artesano, y cada artesano está vinculado a un usuario. Esto refleja la capacidad de los usuarios para asumir roles duales en la plataforma.
- **Artesano -> Producto:** Los artesanos pueden crear múltiples productos, y cada producto está asociado a un artesano. Esto permite la atribución de productos a sus creadores.
- **Usuario -> Encargo:** Un usuario puede realizar varios encargos, y cada encargo está relacionado con un usuario.
- **Artesano -> Encargo:** Un artesano puede aceptar varios encargos, y cada encargo puede tener o no asignado un artesano.

En la capa de SQLite, no se requerirán relaciones complejas, ya que su enfoque principal es el almacenamiento de datos personales del usuario. La estructura de datos se simplifica, centrándose en la gestión de artículos favoritos del usuario.

- **Usuario -> Producto**: Un usuario puede guardar productos en su lista de favoritos.

DAOs (Data Access Objects):

Firebase no utiliza DAOs debido a que las operaciones CRUD (Crear, Leer, Actualizar y Eliminar) se gestionan directamente a través de Firebase Realtime Database y Firestore.
En cambio, en el caso de SQLite, se utilizarán DAOs para acceder y manipular los datos locales.de SQLite:

- **FavoritosDAO:** Un DAO para gestionar las operaciones relacionadas con los artículos favoritos del usuario, como añadir y eliminar favoritos.

Repositorios:

En cuanto a la base de datos de Firebase, se crearán repositorios para gestionar la lógica relacionada con las operaciones de Firebase. En concreto, se utilizará un repositorio para la autenticación y otro para la gestión de datos en tiempo real.
Estos repositorios actuarían como intermediarios entre las referencias de Firebase y el ViewModel.

En la capa de SQLite, se implementarán repositorios para encapsular la lógica relacionada con la persistencia local. Esto incluirá la implementación de métodos para obtener, insertar, actualizar y eliminar datos en SQLite.

- **FavoritosRepository**: Facilita la interacción con los productos favoritos del usuario, utilizando el FavoritosDAO para operaciones de base de datos.

4. Reflexión sobre cambios futuros

¿Qué pasaría si en un futuro se quisiera cambiar el motor de base de datos?
Sería necesario realizar una migración de todos los datos de Firebase o SQLite a la nueva base de datos, además de revisar y adaptar la capa de persistencia de la aplicación para que sea compatible con el nuevo motor.

¿Qué partes de tu aplicación tendrías que modificar?
En el caso de cambiar de Firebase a otro motor de base de datos, tendríamos que modificar el repositorio, la lógica de autenticación y cualquier otra parte que interactúe directamente con Firebase. Además, habría que revisar las consultas y operaciones específicas que pueden variar entre motores de base de datos.
En el caso de SQLite, tendríamos que modificar las entidades y tablas, además de adaptar los DAOs para que interactúen con este nuevo motor.

¿Qué dificultades anticipas?
Las principales dificultades serían: garantizar la integridad y consistencia de los datos durante la migración, enfrentar posibles incompatibilidades entre Firebase y el nuevo sistema, y la curva de aprendizaje asociada al nuevo sistema.

¿Cómo podrías diseñar tu aplicación para minimizar el impacto de tal cambio?
Una clara abstracción entre la capa de datos y la lógica de la aplicación es esencial. Esto se puede lograr mediante la implementación de interfaces o patrones de diseño que separen la lógica de la aplicación de la lógica de la base de datos. Así, si en el futuro se decide cambiar la base de datos, solo habría que implementar de nuevo estas interfaces sin alterar el resto del código.