FoodRadar: Desarrollo de una Aplicación Móvil para Exploración Gastronómica Basada en Geolocalización

Diego Ramírez, Gabriel Londoño, Santiago Restrepo, David Solis
Departamento de Ingeniería de Sistemas e Industrial
Facultad de Ingeniería, Universidad Nacional de Colombia
Bogotá, Colombia
Emails: {dieramirezma, glondonot, srestreporo, dsolis}@unal.edu.co

***Resumen*—FoodRadar es una aplicación móvil desarrollada para Android cuyo propósito es permitir a los usuarios descubrir restaurantes cercanos mediante geolocalización en tiempo real, mapas interactivos y filtrado según preferencias culinarias. La aplicación utiliza tecnologías como OSMDroid, Firebase y Google Play Services para proporcionar funcionalidades como visualización de rutas, estimación de tiempos de viaje, análisis de comentarios mediante técnicas de procesamiento de lenguaje natural y perfiles personalizados de usuario. Este documento presenta la arquitectura del sistema, el conjunto de tecnologías empleadas, los flujos principales de uso y las decisiones de diseño que guiaron la implementación final de FoodRadar.**

***Index Terms*—Aplicaciones móviles, geolocalización, Android, OSMDroid, Firebase, mapas interactivos, recomendación gastronómica.**

1. Introducción

El uso de aplicaciones móviles basadas en localización se ha consolidado como una herramienta fundamental para la búsqueda y recomendación de establecimientos gastronómicos. En este contexto, FoodRadar surge como una solución orientada a facilitar la exploración culinaria mediante la integración de tecnologías de geolocalización, representación cartográfica y análisis de datos.

A diferencia de plataformas consolidadas como Google Maps o TripAdvisor, que dependen de infraestructuras comerciales y marcos de representación cerrados, FoodRadar adopta un enfoque basado en software de código abierto y servicios en la nube flexibles. Esto permite un mayor control sobre la visualización del mapa, los marcadores, las rutas y los flujos de interacción con el usuario.

El presente informe describe el proceso de diseño e implementación de FoodRadar, abarcando la arquitectura del sistema, las tecnologías utilizadas, los flujos funcionales y las consideraciones de seguridad y almacenamiento de datos.

2. Trabajos Relacionados

Las aplicaciones de exploración gastronómica han sido ampliamente estudiadas y desarrolladas. Herramientas como Foursquare, Google Maps o Yelp proporcionan funcionalidades robustas para la búsqueda de establecimientos, pero dependen de datos altamente estructurados y bases propietarias [3]. En particular, investigaciones recientes han enfatizado la relevancia de integrar datos geoespaciales con análisis de opiniones, combinando mapas interactivos con técnicas de procesamiento del lenguaje natural para mejorar la calidad de las recomendaciones.

OSMDroid ha sido utilizado en múltiples trabajos debido a su naturaleza abierta, permitiendo personalización total de capas, estilos y marcadores [1]. De manera similar, Firebase se ha consolidado como una solución efectiva para autenticación y almacenamiento de datos en aplicaciones móviles, gracias a su escalabilidad y sincronización en tiempo real [2].

FoodRadar combina estas tecnologías con la API de análisis de texto Gemini, lo que permite ir más allá de la simple visualización de restaurantes, incorporando un análisis semántico de comentarios para ofrecer recomendaciones personalizadas.

3. Arquitectura Detallada del Sistema

La arquitectura de FoodRadar se basa en una estructura clara de tres capas en la aplicación móvil que interactúa con un conjunto de servicios externos y de nube, como se detalla en la Fig. 1.

3.1. Capas del Cliente Móvil (Android)

El lado del cliente (cajas azul, naranja y morada) implementa el patrón de tres capas para gestionar la presentación, la lógica de negocio y el acceso a datos.

- **Capa de Interfaz de Usuario (UI - Azul):** Contiene todas las Actividades (e.g., **Mapa**, **Opciones**, **Perfil**) y los Layouts XML. Es responsable de la interacción con el usuario, la captura de entradas (e.g., solicitud de ruta) y la representación visual de los datos, incluyendo el renderizado del mapa a través de **OSMDroid**.
- **Capa de Lógica de Negocio (Business Logic - Naranja):** Orquesta el flujo de la aplicación. Módulos como **Gestión de Mapas y Geolocalización**

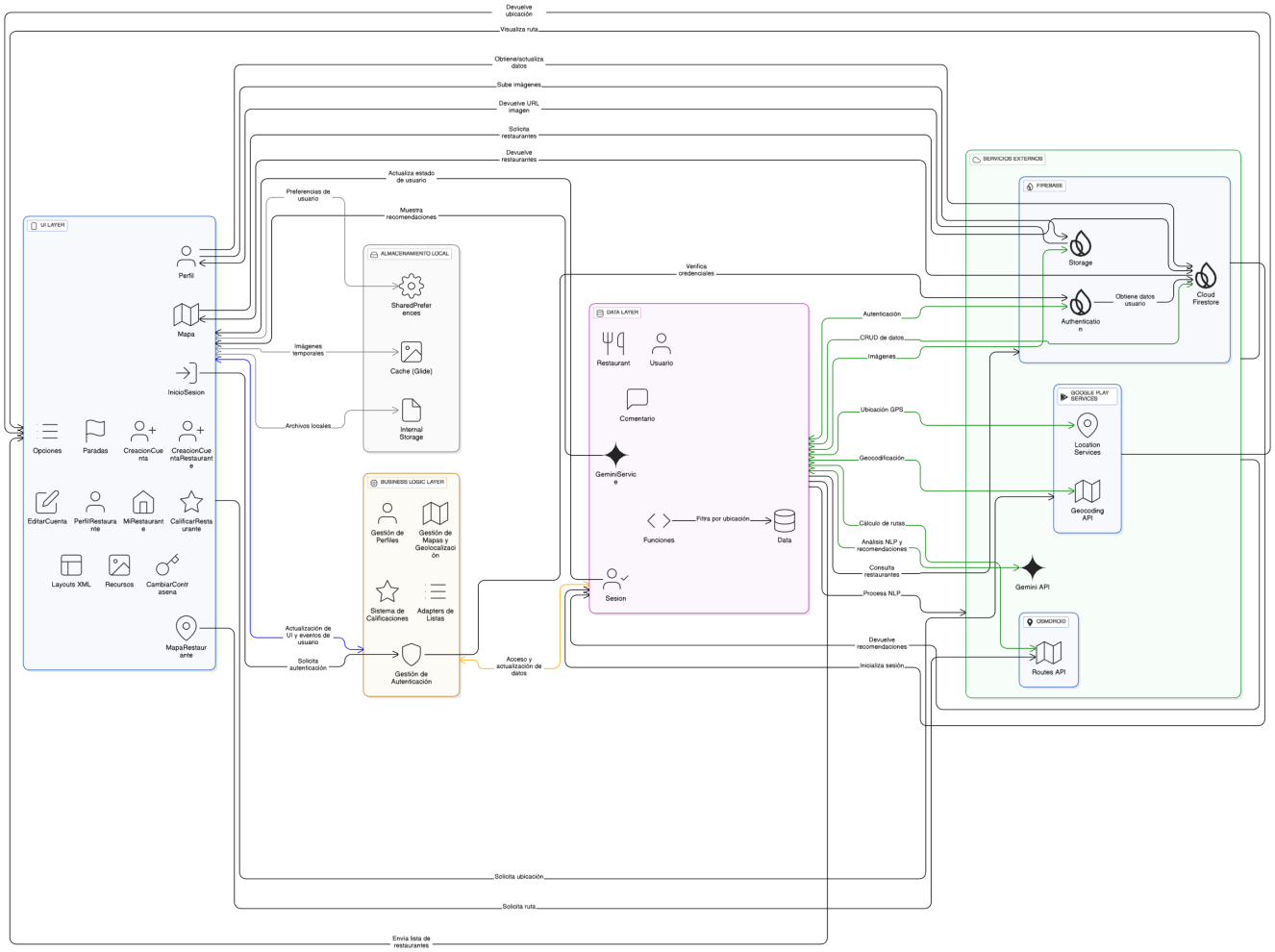

Figura 1. Diagrama de Arquitectura del Sistema FoodRadar.

procesan la ubicación del usuario y filtran la lista de restaurantes obtenida de la capa inferior. Adapta los datos crudos para que sean presentables en la UI.

- **Capa de Datos (Data Layer - Morada):** Encargada de la persistencia y la comunicación con el *backend*. Contiene los **Modelos** (*Restaurant, Usuario*) y los Servicios de acceso (e.g., **Funciones**), gestionando la sesión del usuario y el *caching* de imágenes y archivos.

3.2. Servicios de Backend y Externos

Los servicios externos (cajas verdes) son esenciales para la funcionalidad de nube, geolocalización avanzada y análisis inteligente.

- **Firebase (Authentication, Firestore, Storage):** Proporciona la infraestructura de nube. **Authentication** maneja el inicio de sesión. **Firestore** almacena la información estructurada de la aplicación (usuarios, restaurantes, comentarios) y **Storage** maneja los archivos binarios (imágenes).
- **Gemini API (Análisis NLP):** Se utiliza para procesar y analizar los comentarios de los usuarios (*Comentario*) mediante **Procesamiento de Lenguaje Natural (NLP)**. Esto permite generar un **Análisis NLP** y **Recomendaciones** que influyen en los datos presentados al usuario.
- **Servicios de Geolocalización:**
  - **Google Play Services Location:** Se usa exclusivamente para la obtención precisa de la ubicación GPS del dispositivo (*FusedLocationProviderClient*).
  - **OSRM (Open Source Routing Machine):** Integrado a través de **OSMBonusPack**, gestiona el **Cálculo de la Ruta** óptima y los tiempos de viaje, enviando el **overlay** de la ruta de vuelta a **OSMDroid** para su visualización.

La comunicación entre la aplicación y el *backend* es bidi-

reccional, permitiendo la solicitud de datos (*Obtiene datos de usuario*) y la actualización de información (*Actualización de datos*).

4. Tecnologías y Dependencias

El desarrollo de FoodRadar integra diversas tecnologías modernas, priorizando la arquitectura modular, el rendimiento adecuado y la escalabilidad mediante el uso estratégico de código abierto y servicios en la nube.

4.1. Lenguaje de Programación

El proyecto fue desarrollado completamente en **Kotlin**, el lenguaje de programación oficialmente recomendado por Google para Android. Las ventajas de Kotlin sobre Java incluyen:

- **Seguridad de Nulos (Null Safety):** Reduce la probabilidad de errores comunes como el *NullPointerException*.
- **Expresividad y Concisión:** Permite escribir código más limpio y con menos repeticion
- **Concurrencia con Coroutines:** Se utiliza la biblioteca **Kotlin Coroutines** para la gestión de operaciones asíncronas y de larga duración (como las consultas a Firestore y la comunicación con la Gemini API) sin bloquear el hilo principal (UI).

4.2. Bibliotecas de AndroidX y Arquitectura

Para garantizar una arquitectura robusta y mantener la compatibilidad con versiones modernas de Android, se empleó el conjunto de bibliotecas **AndroidX**, crucial para el uso del patrón **MVVM (Model-View-ViewModel)**.

- **Lifecycle y ViewModel:** Facilitan la gestión del ciclo de vida de los componentes, asegurando que los datos persistan durante los cambios de configuración y que la lógica de negocio se separe de la interfaz de usuario.
- **ConstraintLayout:** Se adoptó para la construcción de interfaces complejas y responsivas con jerarquías de vista planas, mejorando el rendimiento de renderizado.
- **Navigation Component:** Permite un control estructurado, seguro y visual de la navegación entre las diferentes pantallas (*Activities* y *Fragments*) de la aplicación.

4.3. Mapas y Servicios de Geolocalización (Open-Source)

La decisión de usar tecnologías de código abierto para la cartografía ofrece un control total sobre la personalización del mapa.

- **OSMDroid:** Es el motor principal para la visualización de mapas, utilizando datos de **OpenStreetMap**. Su naturaleza abierta elimina las restricciones de uso y los costos asociados a las soluciones propietarias.
- **OSMBonusPack:** Extiende las capacidades de OSMDroid, siendo fundamental para:
  - El manejo avanzado de **Marcadores** personalizados.
  - La integración con el motor de rutas **OSRM (Open Source Routing Machine)** para el cálculo eficiente y la visualización de rutas (*RoadManager* y *RoadOverlay*).
- **Google Play Services Location (FusedLocationProviderClient):** A pesar de no usar Google Maps para el renderizado, se utiliza este servicio de Google debido a su precisión y su optimización en el uso de la batería para obtener la ubicación GPS en tiempo real.
- **Android Geocoder Nativo:** Se emplea para la conversión de coordenadas geográficas a direcciones legibles por humanos (geocodificación inversa), aprovechando las funciones base del SDK de Android.

4.4. Servicios en la Nube (BaaS)

Se utilizó el ecosistema de **Firebase** como *Backend as a Service* (BaaS) para acelerar el desarrollo y garantizar la escalabilidad.

- **Firebase Authentication:** Provee un sistema de registro e inicio de sesión seguro y escalable, manejando la gestión de tokens y sesiones de usuario.
- **Cloud Firestore:** Actúa como la base de datos principal NoSQL. Su modelo de datos orientado a documentos y la sincronización en tiempo real son ideales para actualizar los mapas y las listas de restaurantes de forma dinámica.
- **Firebase Storage:** Se utiliza para el almacenamiento de objetos binarios, específicamente las imágenes de perfil de usuario y las fotografías de los restaurantes.

4.5. Integración de Inteligencia Artificial

- **API de Gemini:** Esta integración es crucial para la diferenciación del proyecto. Se utiliza el modelo multimodal de Gemini para realizar un **Análisis Semántico de Comentarios** mediante NLP. Permite ir más allá del promedio numérico, ofreciendo recomendaciones basadas en el sentimiento y el contenido textual de las opiniones de los usuarios.

4.6. Otras Dependencias Críticas

- **Glide:** Biblioteca líder en el manejo de imágenes para Android. Gestiona la carga, el *caching* y el cambio de tamaño de imágenes de manera eficiente, optimizando el uso de la memoria.

- **Gson/Jackson:** Bibliotecas de serialización/deserialización para el manejo de estructuras JSON, utilizadas para la comunicación con APIs externas (como la API de Gemini) y la gestión de datos locales temporales.

5. Flujos Principales de Uso

Los flujos funcionales de la aplicación se estructuraron para asegurar una experiencia fluida y coherente, centrándose en el descubrimiento, la recomendación asistida por IA y la navegación.

5.1. Descubrimiento de Restaurantes y Recomendación Inteligente

El proceso de exploración inicia con la selección de un tipo de comida en la pantalla de *Opciones* o mediante la selección manual de restaurantes en la vista principal, luego de haberse autenticado. La aplicación consulta la base de datos **Firestore** y filtra los establecimientos en función de la ubicación del usuario y sus preferencias.

Un flujo clave es la **Recomendación con IA**, donde el usuario puede solicitar una sugerencia personalizada. Como

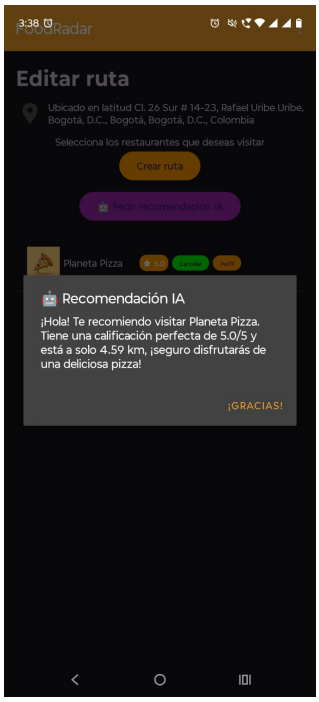

Figura 2. Mensaje de recomendación generado por Gemini.

se muestra en la Fig. 2, la aplicación interactúa con la **API de Gemini** para analizar comentarios de restaurantes, extraer el sentimiento y la calidad de la calificación, y generar una recomendación concisa e informativa. En este ejemplo, se recomienda visitar "Planeta Pizza"basándose en su calificación perfecta y la cercanía (4.59 km). Este flujo ilustra la integración entre la *Capa de Datos* (GeminiService) y la *Lógica de Negocio* (filtrado por distancia).

5.2. Navegación y Rutas Interactivas

Una vez que el usuario selecciona uno o varios destinos, el sistema calcula la ruta óptima.

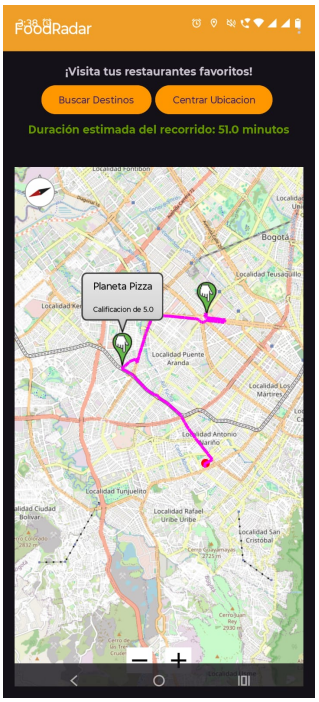

Figura 3. Visualización de ruta de navegación óptima calculada por OSRM.

- Cálculo de Ruta: La *Lógica de Negocio* utiliza el motor **OSRM** (integrado mediante *OSMBonusPack*) para calcular la ruta más eficiente desde la ubicación actual del usuario hasta el destino.
- Visualización: El mapa, renderizado por **OSMDroid** con datos de OpenStreetMap, muestra la ruta trazada en magenta y los marcadores de destino, como se observa en la Fig. 3. La interfaz proporciona inmediatamente información relevante para el viaje, como la **Duración estimada del recorrido (51.0 minutos)**. Este proceso permite al usuario planificar su desplazamiento de manera eficiente y visual.

5.3. Gestión de Perfil

El sistema permite el manejo completo de la experiencia del usuario:

- **Autenticación Segura:** Los flujos de *Registro* e *Inicio de Sesión* se manejan mediante **Firebase Authentication**.
- **Preferencias:** El sistema permite la edición de datos de usuario y el cambio de contraseña, almacenando las preferencias personales en **Firestore** y configuraciones simples en **SharedPreferences**.

- **Historial:** Se registra el historial de restaurantes visitados o calificados para futuras referencias y para alimentar el sistema de recomendación avanzado.

6. Seguridad y Privacidad

La seguridad del usuario y la gestión responsable de los permisos y datos son pilares fundamentales en el diseño de FoodRadar. Implementamos medidas de seguridad a nivel de aplicación, base de datos y transmisión de datos.

6.1. Autenticación y Gestión de Sesión

La autenticación se implementa mediante **Firebase Authentication**, aprovechando su robustez y cumplimiento de estándares de la industria.

- **Gestión de Tokens:** Firebase maneja automáticamente la creación, expiración y refresco de tokens de sesión, minimizando el riesgo de robo de identidad.
- **Cifrado de Credenciales:** Las contraseñas de los usuarios se almacenan mediante algoritmos de *hashing* seguros antes de su persistencia en el backend.
- **Diferenciación de Roles:** Se implementaron lógicas para diferenciar entre usuarios estándar y perfiles de restaurantes, asegurando que solo los usuarios autorizados puedan editar los datos de sus respectivos establecimientos.

6.2. Protección de Datos en Firestore

Para evitar el acceso no autorizado a los datos, se establecieron **Reglas de Seguridad en Firestore**.

- **Control Granular de Acceso:** Las reglas definen quién puede leer, escribir o eliminar documentos. Por ejemplo, los datos de los comentarios solo pueden ser escritos por un usuario autenticado y los datos del perfil solo pueden ser modificados por el propietario.
- **Validación de Datos:** Las reglas también validan la estructura y el tipo de datos antes de ser escritos en la base de datos, previniendo inyecciones de datos maliciosos o inconsistentes.

6.3. Permisos y Privacidad en Android

El manejo de la geolocalización es la principal preocupación de privacidad.

- **Permisos de Ubicación:** Se solicita el permiso de **ubicación en primer plano** solo cuando la aplicación está en uso activo, y se sigue el principio de mínimo privilegio, nunca solicitando acceso en segundo plano.
- **Almacenamiento en Caché:** El almacenamiento local y en caché se gestiona de acuerdo con las buenas prácticas de Android para evitar la exposición de datos sensibles.

7. Almacenamiento de Datos

La estrategia de almacenamiento de FoodRadar combina la escalabilidad de la nube con la eficiencia del almacenamiento local, optimizando tanto el rendimiento como la disponibilidad de los datos.

7.1. Cloud Firestore (Base de Datos NoSQL en Tiempo Real)

Firestore es la columna vertebral del almacenamiento de datos estructurados, elegida por su escalabilidad horizontal y su capacidad de sincronización en tiempo real.

- **Datos Clave:** Almacena datos de *restaurantes* (ubicación, categoría, valoración), *usuarios* (preferencias, historial) y *comentarios/calificaciones*.
- **Consultas Optimizadas:** El diseño de la colección se optimizó para las consultas basadas en geolocalización (mediante el uso de campos de latitud/longitud) y el filtrado por categoría, esenciales para el flujo de descubrimiento.
- **Sincronización en Vivo:** La capacidad de escuchar cambios en tiempo real permite que la lista de restaurantes y el mapa se actualicen automáticamente cuando un nuevo restaurante es añadido o calificado, sin necesidad de que el usuario recargue manualmente la actividad.

7.2. Firebase Storage (Almacenamiento de Archivos)

Se utiliza para el manejo de objetos binarios, desacoplando el almacenamiento de imágenes de la base de datos de documentos.

- **Imágenes de Perfil y Establecimientos:** Almacena las fotos de los restaurantes y las imágenes de perfil de usuario.
- **URL Seguras:** Proporciona URLs seguras y temporales para acceder a los archivos, integrándose perfectamente con la biblioteca **Glide** para la carga eficiente en la interfaz.

7.3. Almacenamiento Local y Persistencia (Caché y Preferencias)

El almacenamiento local es clave para mejorar la experiencia del usuario y reducir la latencia.

- **SharedPreferences:** Se emplea para la persistencia liviana y no sensible, incluyendo configuraciones de usuario (e.g., unidades de distancia preferidas) y los filtros gastronómicos seleccionados.
- **Almacenamiento en Caché de Glide: Glide** gestiona automáticamente el caché de imágenes descargadas de Firebase Storage, lo que reduce drásticamente el uso de ancho de banda y el tiempo de carga en visitas posteriores.

- **Persistencia de Datos Estáticos:** Ciertos datos estáticos o resultados de análisis se almacenan localmente para garantizar la disponibilidad instantánea y reducir las llamadas redundantes a las APIs de nube.

8. Integraciones y Servicios Externos

FoodRadar se distingue por su integración inteligente de servicios externos de código abierto y APIs de inteligencia artificial, garantizando funcionalidad avanzada sin comprometer la flexibilidad.

8.1. Integración de Inteligencia Artificial: Gemini API

La integración con la **Gemini API** (como parte de la plataforma Google AI) es el diferenciador clave en el sistema de recomendación.

- **Análisis Semántico de Comentarios:** El modelo de lenguaje es utilizado para realizar **Procesamiento de Lenguaje Natural (NLP)** sobre los comentarios de los usuarios. Esto permite extraer no solo la calificación numérica, sino también el sentimiento, la mención de platos específicos y las críticas recurrentes.
- **Generación de Recomendaciones Personalizadas:** Basándose en el análisis semántico y el historial de preferencias del usuario, la API genera recomendaciones concisas e hiper-personalizadas, mejorando la calidad de las sugerencias automáticas.

8.2. Infraestructura Cartográfica Abierta: OSMDroid y OSRM

En contraste con las soluciones propietarias, se optó por una infraestructura de mapas y rutas de código abierto.

- **OSMDroid y OpenStreetMap:** Proporcionan la visualización del mapa base, permitiendo una personalización completa del estilo, la capa de mapa y los marcadores sin incurrir en costos de licencia por volumen.
- **OSRM (Open Source Routing Machine):** Integrado mediante la librería **OSMBonusPack**, OSRM es el motor de cálculo de rutas. Proporciona un servicio de enrutamiento rápido y flexible para calcular la ruta óptima, distancia y tiempo estimado de viaje entre el usuario y los restaurantes.

8.3. Servicios de Geolocalización (Google Play Services)

Aunque se evita la plataforma de mapas completa de Google, se utiliza el servicio más básico y eficiente para la obtención de ubicación.

- **FusedLocationProviderClient:** Se emplea para obtener la ubicación precisa del dispositivo de manera eficiente en términos de batería, asegurando una experiencia de usuario fluida sin depender de servicios de mapas de pago.

9. Conclusiones y Trabajo Futuro

FoodRadar representa un proyecto exitoso en la intersección de aplicaciones móviles, sistemas de información geográfica (SIG) e inteligencia artificial. El sistema demuestra la viabilidad de integrar tecnologías abiertas (**OSMDroid, OSRM**), servicios escalables en la nube (**Firebase**) y capacidades de análisis semántico (**Gemini**) para construir una plataforma de exploración gastronómica robusta y modular. La arquitectura en capas garantiza la mantenibilidad y facilita la evolución del producto.

9.1. Contribuciones Clave del Proyecto

- **Arquitectura Híbrida:** Demostración de un modelo de arquitectura que combina la eficiencia de las herramientas open-source para mapas y rutas con la robustez de un BaaS líder para la gestión de datos.
- **Recomendación Inteligente:** Superación del sistema básico de promedios de calificación mediante la integración de NLP para una comprensión profunda de las opiniones de los usuarios.
- **Experiencia Localizada y Eficiente:** Uso optimizado de *FusedLocationProviderClient* y *Kotlin Coroutines* para garantizar la precisión de la geolocalización y el rendimiento de las operaciones asíncronas.

Referencias

[1] OSMDroid, "OpenStreetMap Android Library," 2025. [Online]. Available: https://osmdroid.github.io/

[2] Firebase, "Firebase Documentation," Google, 2025. [Online]. Available: https://firebase.google.com/

[3] Google Maps Platform, "Routes API Documentation," 2025. [Online]. Available: https://developers.google.com/maps

[4] Android Developers, "AndroidX Libraries," 2025. [Online]. Available: https://developer.android.com/jetpack