Documentación Técnica – Sistema de Gestión de Incidencias del Metro CDMX
Hackathon AI Mobility 2025

---

1. Descripción General del Proyecto

El Sistema de Gestión de Incidencias del Metro CDMX es una aplicación móvil diseñada para optimizar la detección, reporte, asignación y resolución de incidencias operativas dentro de la red del Metro de la Ciudad de México.

Participan tres actores clave con roles bien definidos:

- Jefe de Estación: detecta y crea un reporte de incidencia desde su estación.
- Regulador: revisa el reporte, agrega información técnica y lo asigna.
- Técnico: recibe el reporte completo y se desplaza a la estación para resolver la incidencia.

La aplicación se desarrolla para Android, utilizando Firebase como plataforma backend para autenticación, base de datos en tiempo real (Firestore) y sincronización entre usuarios.

---

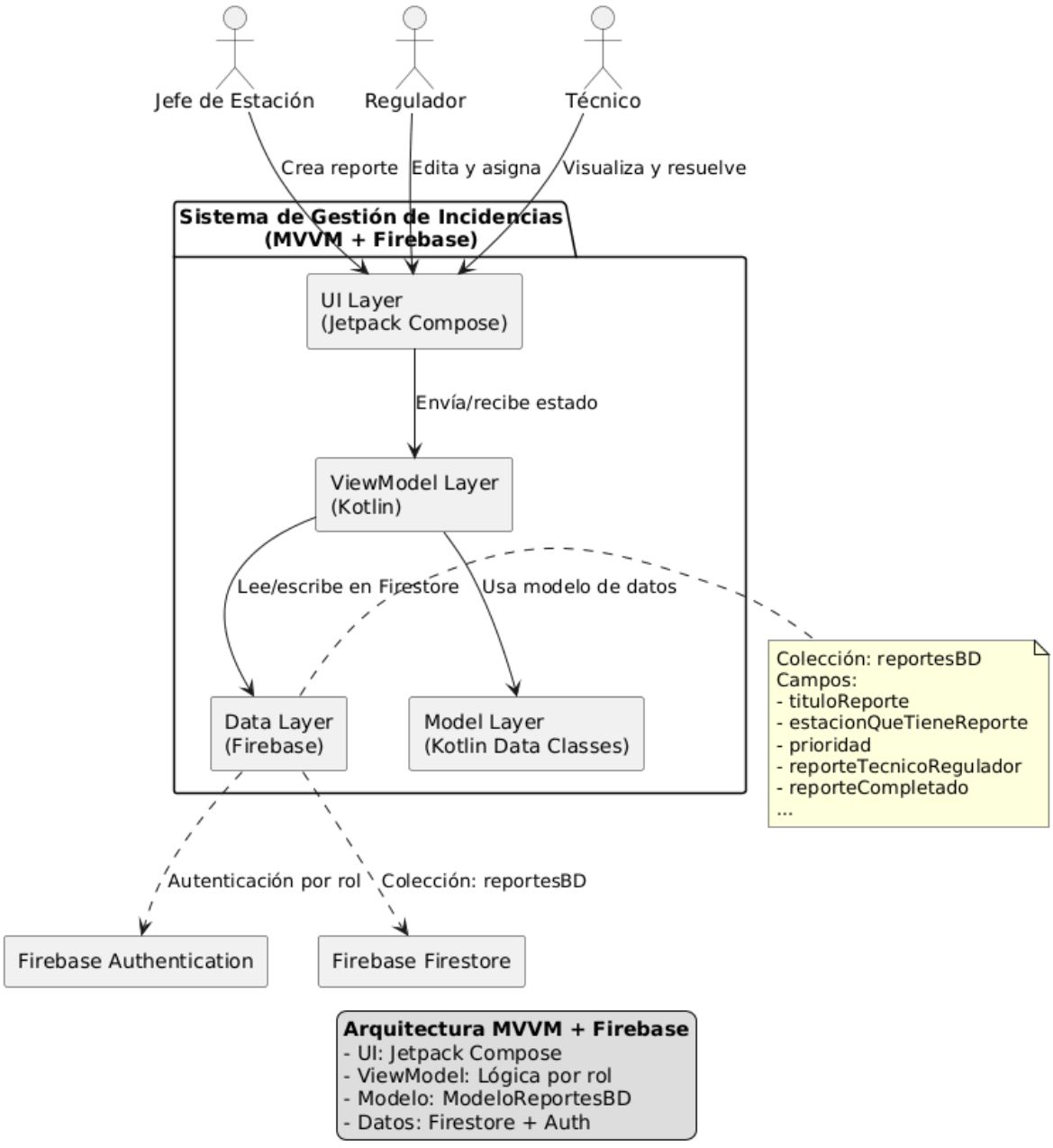
2. Arquitectura del Sistema

La aplicación sigue el patrón arquitectónico MVVM (Model-View-ViewModel) para lograr una clara separación entre lógica de negocio, presentación y datos. Está construida con Kotlin y Jetpack Compose para una interfaz moderna y reactiva.

2.1. Capas del Sistema

- UI Layer (Interfaz de Usuario)
  Implementada con Jetpack Compose. Cada rol tiene una interfaz específica:
    - Pantalla del Jefe de Estación (creación de reportes)
    - Pantalla del Regulador (edición y asignación)
    - Pantalla del Técnico (visualización y cierre de incidencias)
- ViewModel Layer
  Contiene la lógica de presentación:
    - JefeEstacionViewModel
    - ReguladorViewModel
    - TecnicoViewModel
      Estos componentes gestionan validaciones, llamadas al repositorio y el estado de los reportes.

- Model Layer
  Define la estructura de los datos mediante la clase ModeloReportesBD, que representa un documento en Firestore.
- Data Layer
  Usa Firebase Firestore como base de datos en tiempo real y Firebase Authentication para la gestión de usuarios por rol.



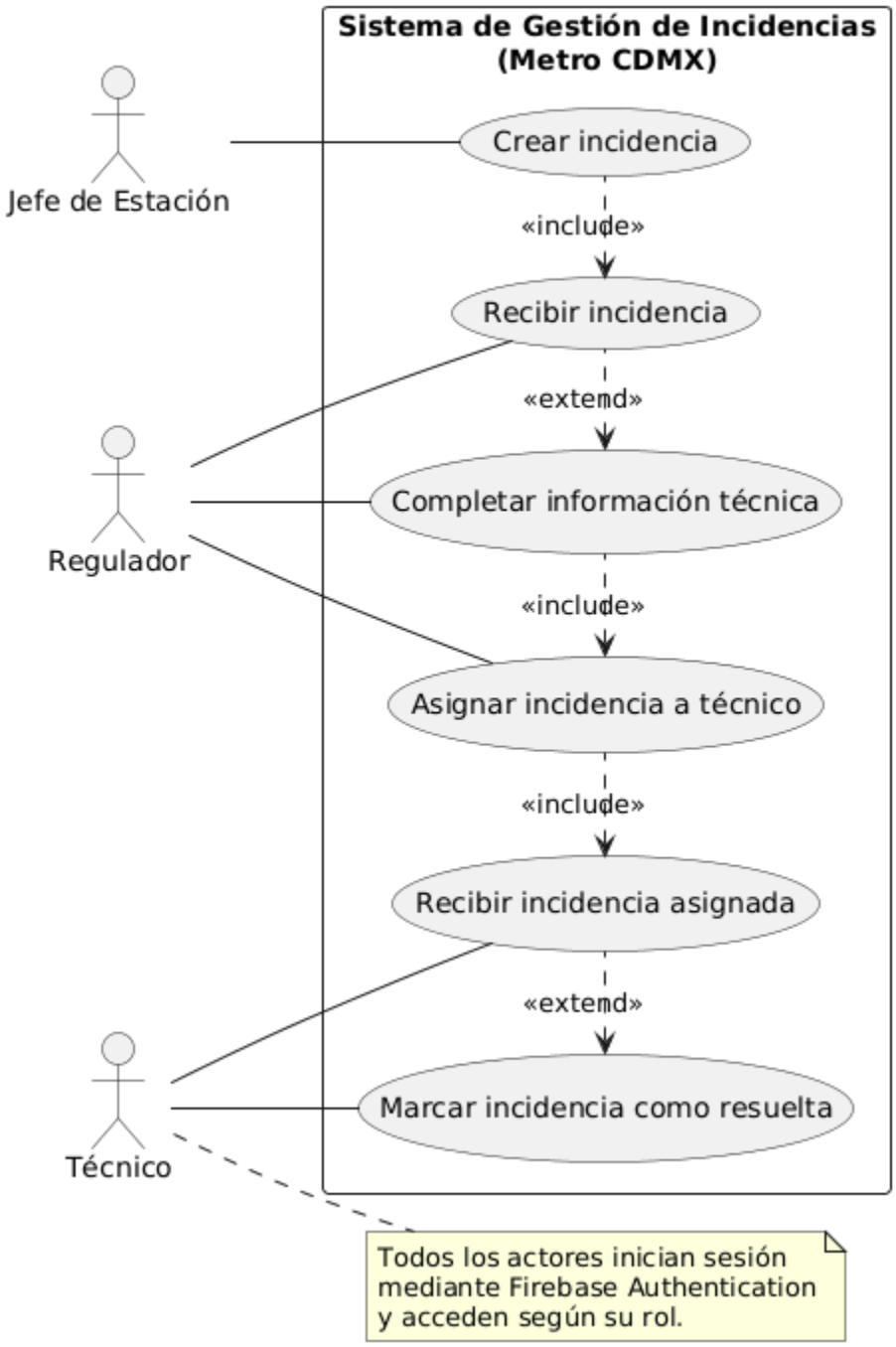
2.2. Comunicación entre Actores

La sincronización entre los tres roles se logra mediante escuchas en tiempo real a la colección reportesBD:

1. El Jefe de Estación crea un documento con estado implícito *"pendiente"*.
2. El Regulador filtra los reportes sin datos técnicos y los completa.
3. El Técnico recibe el reporte actualizado y lo marca como *"resuelto"* al finalizar.

Nota: El sistema está diseñado para permitir fácil integración futura de notificaciones push (Firebase Cloud Messaging).

Diagrama de Casos de Uso - Sistema de Incidencias Metro CDMX



---

3. Flujo del Sistema

1. Jefe de Estación
   Crea un reporte con los siguientes datos:
   - Título del reporte
   - Descripción del problema
   - Hora aproximada del incidente
   - Estación afectada
2. Regulador
   Recibe el reporte y agrega:
   - Descripción técnica detallada
   - Herramientas necesarias
   - Nivel de prioridad (1 = alta, 2 = media, 3 = baja)
     Al enviar, el reporte se actualiza en Firestore y se marca como listo para asignación.
3. Técnico
   Recibe el reporte completo y procede a atenderlo en el lugar indicado.

---

4. Estructura de Datos en Firebase Firestore

Colección: reportesBD

Campo	Tipo	Descripción
`tituloReporte`	String	Título del incidente
`descripcionReporteJefeDeEstacion`	String	Descripción inicial del Jefe de Estación
`estacionQueTieneReporte`	String	Nombre de la estación afectada
`horaProblema`	String	Hora aproximada del incidente (HH:mm)
`fechaHoraCreacionReporte`	Timestamp	Momento exacto de creación del reporte
`nombreDeJefeDeEstacionCreadorReporte`	String	Nombre del Jefe que reportó
`reporteTecnicoRegulador`	String	Descripción técnica agregada por el Regulador
`prioridad`	Int	1 = alta, 2 = media, 3 = baja
`herramientas`	String	Herramientas o materiales necesarios

`reporteCompletado`	Int	0 = pendiente, 1 = resuelto

5. Modelo de Datos (Kotlin)

```
data class ModeloReportesBD(
  val idDocumento: String? = null,
  val nombreDeJefeDeEstacionCreadorReporte: String? = null,
  val fechaHoraCreacionReporte: Timestamp? = null,
  val tituloReporte: String? = null,
  val estacionQueTieneReporte: String? = null,
  val descripcionReporteJefeDeEstacion: String? = null,
  val horaProblema: String? = null,
  val reporteTecnicoRegulador: String? = null,
  val prioridad: Int? = null,
  val herramientas: String? = null,
  val reporteCompletado: Int? = 0
)
```

6. Vista del Regulador

El Regulador accede a los reportes creados por los Jefes de Estación y los complementa con:

- Descripción técnica detallada del problema
- Lista de herramientas necesarias para la reparación
- Nivel de prioridad (1, 2 o 3)

Una vez completado, al hacer clic en “Enviar”, el sistema actualiza el documento en Firestore, permitiendo que el Técnico reciba el reporte completo.

---

7. Instalación y Configuración

Para ejecutar el proyecto localmente, seguir estos pasos:

1. Clonar el repositorio en Android Studio.
2. Asegurar que el dispositivo o emulador esté en modo desarrollador.
3. Configurar el proyecto con SDK 35 (compileSdk y targetSdk).
4. Colocar el archivo google-services.json (descargado desde Firebase Console) en la carpeta app/.

5. Sincronizar el proyecto con Gradle.

---

8. Manual de Usuario

- Jefe de Estación:
  Inicia sesión → Toca “Nuevo Reporte” → Ingresa datos → Envía.
  ✅ Tu incidencia será revisada por un Regulador.
- Regulador:
  Visualiza reportes pendientes → Completa información técnica → Asigna al técnico.
- Técnico:
  Recibe notificación → Abre el reporte → Atiende la incidencia → Marca como “Completado”.

---

9. Créditos

Desarrollado por el Equipo Hackathon AI Mobility 2025 como solución innovadora para la movilidad urbana y la operación eficiente del Metro CDMX.
Este sistema busca reducir tiempos de respuesta, optimizar recursos humanos y mejorar la experiencia del usuario dentro del transporte público.