Technical Documentation – Mexico City Metro Incident Management System
Hackathon AI Mobility 2025

1. Project Overview

The Mexico City Metro Incident Management System is a mobile application designed to optimize the detection, reporting, assignment, and resolution of operational incidents across the Mexico City Metro network.

Three key user roles are clearly defined:

- Station Manager: detects and creates an incident report from their station.
- Coordinator: reviews the report, adds technical details, and assigns it.
- Technician: receives the complete report and travels to the station to resolve the issue.

The app is developed for Android, using Firebase as the backend platform for authentication, real-time database (Firestore), and user synchronization.

2. System Architecture

The application follows the MVVM (Model–View–ViewModel) architectural pattern to achieve a clear separation between business logic, presentation, and data. It is built with Kotlin and Jetpack Compose for a modern and reactive user interface.

2.1 System Layers

- UI Layer (User Interface)
  Implemented with Jetpack Compose. Each role has a dedicated interface:
    - Station Manager screen (for creating reports)
    - Coordinator screen (for editing and assigning reports)
    - Technician screen (for viewing and closing incidents)
- ViewModel Layer
  Contains presentation logic:
    - StationManagerViewModel
    - CoordinatorViewModel
    - TechnicianViewModel
      These components handle validations, repository calls, and report status management.
- Model Layer
  Defines the data structure through the ReportModelBD class, representing a document in Firestore.

- Data Layer
  Uses Firebase Firestore as the real-time database and Firebase Authentication for role-based user management.

2.2 Communication Between Roles

Synchronization among the three roles is achieved through real-time listeners on the reportesBD (reports) Firestore collection:

1. The Station Manager creates a document with an implicit status of *"pending"*.
2. The Coordinator filters reports lacking technical data and completes them.
3. The Technician receives the updated report and marks it as *"resolved"* upon completion.

Note: The system is designed to easily integrate push notifications in the future (via Firebase Cloud Messaging).

3. System Workflow

1. Station Manager
   Creates a report with the following data:
   - Report title
   - Problem description
   - Approximate time of the incident
   - Affected station
2. Coordinator
   Receives the report and adds:
   - Detailed technical description
   - Required tools
   - Priority level (1 = high, 2 = medium, 3 = low)
     Upon submission, the report is updated in Firestore and marked as ready for assignment.
3. Technician
   Receives the complete report and proceeds to address it at the specified location.

4. Firebase Firestore Data Structure

Collection: reportesBD

Field	Type	Description
`tituloReporte`	String	Incident title

`descripcionReporteJefeDeEstacion`	String	Initial description by the Station Manager
`estacionQueTieneReporte`	String	Name of the affected station
`horaProblema`	String	Approximate incident time (`HH:mm`)
`fechaHoraCreacionReporte`	Timestamp	Exact moment the report was created
`nombreDeJefeDeEstacionCreadorReporte`	String	Name of the reporting Station Manager
`reporteTecnicoRegulador`	String	Technical description added by the Coordinator
`prioridad`	Int	1 = high, 2 = medium, 3 = low
`herramientas`	String	Required tools or materials
`reporteCompletado`	Int	0 = pending, 1 = resolved

5. Data Model (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. Coordinator View

The Coordinator accesses reports created by Station Managers and enhances them with:

- A detailed technical description of the problem
- A list of required tools for repair
- Priority level (1, 2, or 3)

Once completed, clicking “Submit” updates the document in Firestore, allowing the Technician to receive the full report.

7. Installation and Setup

To run the project locally, follow these steps:

1. Clone the repository in Android Studio.
2. Ensure the device or emulator is in developer mode.
3. Configure the project to use SDK 35 (compileSdk and targetSdk).
4. Place the google-services.json file (downloaded from Firebase Console) in the app/ folder.
5. Sync the project with Gradle.

8. User Manual

- Station Manager:
  Log in → Tap “New Report” → Enter data → Submit. ✅ Your incident will be reviewed by a Coordinator.
- Coordinator:
  View pending reports → Complete technical information → Assign to a technician.
- Technician:
  Receive notification → Open report → Address the incident → Mark as “Completed”.

9. Credits

Developed by the Hackathon AI Mobility 2025 team as an innovative solution for urban mobility and efficient operation of the Mexico City Metro.
This system aims to reduce response times, optimize human resources, and improve the user experience within public transportation.