Manual técnico – Depov Alerta Inmediata

Andres Monsalve Perez

Dayan Berrio Toro

Juan David Suarez

Santiago Gallego Gutierrez

Sena - centro de la manufactura avanzada.

Proyecto Final

Desarrollo y Análisis de Software.

2025

Contenido

Resumen.

Este manual técnico documenta el funcionamiento y estructura del sistema compuesto por una aplicación móvil y una plataforma web. Se detallan los requisitos de software, la arquitectura general del sistema, la configuración del backend (API), la estructura de la base de datos, así como el proceso de instalación y ejecución en un entorno local. También se describen los métodos principales, el flujo de información entre los componentes y se proponen posibles mejoras futuras para optimizar su funcionamiento y usabilidad.

Introducción.

El sistema de alerta temprana **Depov** (Despliegue de Emergencia para Operaciones de Verificación) surge como respuesta a la creciente sensación de inseguridad que experimentan muchas mujeres en Medellín, según lo evidenciado en encuestas y opiniones recopiladas. Con el objetivo de mejorar su seguridad, se plantea una solución tecnológica que permita reportar situaciones de emergencia en tiempo real a las autoridades competentes, facilitando así una respuesta rápida y eficiente. Este manual acompaña el desarrollo del sistema, compuesto por una aplicación móvil para usuarios y una plataforma web para la gestión administrativa de alertas.

Requisitos del sistema.

1. Versión web.

- **Navegador compatible:** Google Chrome, Mozilla Firefox, Microsoft Edge o Safari (últimas dos versiones recomendadas).
- **Resolución mínima recomendada:** 1280x720 píxeles.
- Conexión a internet estable.
- Permisos del navegador:
    - Acceso a la ubicación geográfica.
- Requisitos de hardware (modo local):
    - Memoria RAM: 8GB o más.
    - Espacio en el disco: Al menos de 500MG disponibles.

2. Aplicación móvil.

- Dispositivo Android 5.0 o superior.
- Memoria RAM: 2GB o más.
- Conexión a internet: Conexión estable a WIFI o datos móviles (para su versión local preferiblemente WIFI).
- Permitir las peticiones para Servicios concedidos a la aplicación.
    - Acceso a ubicación.
    - Acceso a ubicación en segundo plano.
    - Permiso para mostrar notificaciones.

3. Requisitos del servidor.

- **Sistema operativo recomendado:** Windows 10, macOS, o distribuciones Linux (Ubuntu 20.04+)
- **Node.js:** versión 18.18.0 o superior.
- **Servidor local:** XAMPP, WAMP, o similar (para desarrollo)
- **Entorno de ejecución adicional (opcional):**
    - PM2 para gestión de procesos de Node.js en producción.
- **Requisitos de hardware:**
    - Memoria RAM: mínimo 4 GB (8 GB recomendados).
    - Espacio en disco: 1 GB libre (para la base de datos y archivos del back-end).
- **Otros requisitos:**
    - Puerto 3000 disponible para el back-end.
    - Puerto 3306 disponible para MySQL.

4. Entorno de desarrollo (Recomendado).

- **Editor de código recomendado:** Visual Studio Code.
- **Control de versiones:** Git instalado y configurado.
- **SDK Android:** Android Studio instalado con emulador o dispositivo físico para pruebas.

Estructuración del proyecto.

1. Conceptos.

La estructuración del proyecto se llevó a cabo con la base de una estructura común de un proyecto de Next.js, separando en carpetas para paginas componentes, estilos y archivos públicos, creando después la adición de la carpeta api para el funcionamiento y conexión del aplicativo móvil, además de carpetas específicas para tipos de usuarios.

2. Estructura.

```
depov_2/
├── app/                        # Aplicación web (Next.js)
│   ├── api/                    # Endpoints de la API
│   ├── admin/                  # Páginas administrador
│   ├── police/                 # Páginas para policías
│   ├── dashboard/              # Panel de usuario
│   ├── login/                  # Página inicio de sesión
│   ├── register/               # Página de registro
│   ├── about/                  # Página "Acerca de"
│   ├── globals.css             # Estilos globales
│   ├── layout.tsx              # Estructura principal app
│   └── page.tsx                # Página inicio
├── components/                 # Componentesreutilizables
│   ├── ui/                     # Elementos de interfaz
│   ├── nav-bar.tsx             # Barra de navegación
│   ├── footer.tsx              # Pie de página
│   └── ...                     # Otros componentes
├── lib/                        # configuraciones
│   ├── db.ts                   # Conexión base de datos
│   ├── auth-utils.ts           # autenticación
│   └── cors.ts                 # Configuración de CORS
├── public/                     # Archivos públicos
```

```
    │   └── images/              # Imágenes utilizadas
    ├── middleware.ts            # Middleware
    ├── next.config.mjs          # Archivo de configuración
    └── android/                 # Aplicación móvil Android
        ├── app/                 # Proyecto Android
        │   └── src/             # Código fuente
        │       ├── main/        # Código principal
        │       │   ├── java/    # Código en Kotlin
        │       │   └── res/     # Recursos de la app
        │       └── ...          # Otros recursos internos
        └── ...
    # Archivos de configuración del proyecto
```

Componentes y Archivos.

1. Aplicación Web (Next.js).

1.1. Carpeta /app.

Contiene todas las páginas y rutas de la aplicación web, utilizando el sistema App Router de Next.js.

- **/app/page.tsx** – Página principal del sitio web.
    - **Funcionamiento**: Renderiza la página de inicio con secciones informativas y enlaces a registro/inicio de sesión.
    - **Características**: Diseño responsivo, secciones de información, estadísticas y llamada a la acción.

- **/app/layout.tsx** – Layout principal que envuelve todas las páginas.
    - **Funcionamiento**: Define la estructura común para todas las páginas (navbar, footer, etc.).
    - **Características**: Incluye el proveedor de notificaciones (ToastProvider) y estilos globales.

- **/app/globals.css** – Estilos globales de la aplicación.
    - **Funcionamiento**: Define variables y estilos base para toda la app.
    - **Características**: Utiliza CSS custom properties para mantener consistencia visual.

- **/app/api/** – Endpoints de la API REST.
    - **Funcionamiento**: Maneja las solicitudes HTTP para autenticación, gestión de usuarios, alertas, etc.
    - **Características**: Implementa rutas para operaciones CRUD y lógica de backend.

- **/app/api/auth/route.ts** – Endpoint de autenticación.
    - **Funcionamiento**: Maneja login y verificación de credenciales.
    - **Características**: Valida contra la base de datos y retorna tokens de sesión.

- **/app/api/users/route.ts** – Endpoint para gestión de usuarios.
    - **Funcionamiento**: Opera funciones CRUD sobre los usuarios.
    - **Características**: Manejo completo de usuarios desde el backend.

- **/app/login/page.tsx** – Página de inicio de sesión.
    - **Funcionamiento**: Permite a los usuarios autenticarse en el sistema.
    - **Características**: Formulario con validaciones y feedback de errores.

- **/app/register/page.tsx** – Página de registro.
    - **Funcionamiento**: Registro de nuevos usuarios.
    - **Características**: Campos para datos personales y contactos de emergencia.

- **/app/dashboard/page.tsx** – Panel del usuario.
    - **Funcionamiento**: Muestra información personal y permite enviar alertas.
    - **Características**: Mapa de ubicación, datos personales y botón de alerta.

- **/app/admin/dashboard/page.tsx** – Panel de administrador.
    - **Funcionamiento**: Gestión de usuarios y alertas.
    - **Características**: Tablas, filtros y herramientas administrativas.

- **/app/police/dashboard/page.tsx** – Panel de policía.
    - **Funcionamiento**: Visualización y respuesta a alertas.
    - **Características**: Mapa interactivo, detalles de alerta y acciones rápidas.

1.2. Carpeta /components.

Contiene componentes reutilizables de interfaz de usuario.

- **nav-bar.tsx** – Barra de navegación.
    - **Funcionamiento**: Muestra enlaces y estado de sesión.
    - **Características**: Adaptable a distintos roles y dispositivos.

- **footer.tsx** – Pie de página.
    - **Funcionamiento**: Información de contacto y enlaces útiles.
    - **Características**: Contenido institucional y navegación rápida.

- **alert-map.tsx** – Componente de mapa de alertas.
    - **Funcionamiento**: Muestra alertas y CAIs en mapa interactivo.
    - **Características**: Uso de Google Maps con ubicación en tiempo real.

- **ui/toaster.tsx** – Sistema de notificaciones.
    - **Funcionamiento**: Muestra mensajes temporales (éxito, error, info).
    - **Características**: Estilos dinámicos según el tipo de mensaje.

- **responsive-image.tsx** – Imágenes responsivas.
    - **Funcionamiento**: Optimiza la carga y visualización.
    - **Características**: Utiliza Next/Image con lazy loading.

1.3. Carpeta /lib.

Contiene funciones y configuraciones reutilizables.

- **db.ts** – Conexión a base de datos.
    - **Funcionamiento**: Establece conexión con MySQL.
    - **Características**: Uso de serverless-mysql para eficiencia.

- **auth-utils.ts** – Funciones de autenticación.
    - **Funcionamiento**: Gestión de sesiones y control de acceso.
    - **Características**: Manejo de roles, tokens y sesiones.

- **cors.ts** – Configuración de CORS.
    - **Funcionamiento**: Permite solicitudes desde orígenes permitidos.
    - **Características**: Control de métodos, encabezados y seguridad.

1.4. Archivos de Configuración

- **middleware.ts** – Middleware global de Next.js.
    1. **Funcionamiento**: Aplica lógica antes de cada solicitud.
    2. **Características**: Autenticación y control de acceso.

- **next.config.mjs** – Configuración del framework.
    3. **Funcionamiento**: Parámetros globales de Next.js.
    4. **Características**: Optimización de imágenes, headers y dominios.

2. Aplicación Móvil (Android / Kotlin).

2.1. Rutas Principales.

Código fuente:

Depov\app\src\main\ ...

Back-end (Kotlin):

Depov\app\src\main\java\com\co\apps\Alerta_Inmediata\Depov …

Front-end (xml):

\Depov\app\src\main\res\ ...

2.2. Funcionamiento:

La aplicación está organizada siguiendo una arquitectura por capas que separa la interfaz de usuario, la lógica de negocio y el acceso a datos. Las clases principales implementan funcionalidades como registro de usuarios, inicio de sesión, gestión de perfil y activación de alertas de emergencia.

2.3. Características:

Esta organizado por los siguientes paquetes:

Interfaces y clases para la comunicación con el servidor.

- ApiService.kt: Define los endpoints REST para interactuar con el backend.
- RetrofitClient.kt: Configuración de Retrofit para las peticiones HTTP.
- models/: Modelos de datos para las peticiones y respuestas.
- AlertRequest.kt: Modelo para enviar solicitudes de alerta.
- AlertResponse.kt: Modelo para recibir datos de alertas activas.
- LoginRequest.kt: Modelo para autenticación de usuarios.
- RegisterRequest.kt: Modelo para registro de nuevos usuarios.
- UserResponse.kt: Modelo para datos de perfil de usuario.

Actividades y fragmentos que componen la interfaz de usuario.

- SplashActivity.kt: Pantalla inicial que verifica la sesión del usuario.
- LoginActivity.kt: Maneja la autenticación de usuarios.
- RegisterActivity.kt: Gestiona el registro de nuevos usuarios.
- UpdateActivity.kt: Permite actualizar la información del perfil.
- HomeUserActivity.kt: Pantalla principal con acceso a las funcionalidades.
- ActiveAlertActivity.kt: Muestra información de una alerta activa.
- UserPerfilActivity.kt: Visualización y edición del perfil de usuario.
- RecoverPasswordActivity.kt: Permite recuperar contraseñas olvidadas.

Clases de utilidad y herramientas comunes.

- SessionManager.kt: Gestiona la sesión del usuario mediante SharedPreferences.
- LocationUtils.kt: Funciones para obtener y manejar la ubicación del dispositivo.
- PermissionUtils.kt: Gestión de permisos de la aplicación.
- Constants.kt: Constantes utilizadas en toda la aplicación.

Recursos XML

Contiene todos los recursos necesarios para la interfaz de usuario, incluyendo layouts, imágenes, colores, dimensiones y textos.

Estructura organizada por tipo de recurso:

- **layout/:** Archivos XML que definen la estructura visual de cada pantalla.
- **activity_splash.xml:** Layout para la pantalla de inicio.
- **activity_login.xml:** Formulario de inicio de sesión.
- **activity_register.xml:** Formulario de registro con campos para datos personales y contactos de emergencia.
- **activity_update.xml:** Formulario para actualizar datos de usuario.
- **activity_home_user.xml:** Pantalla principal con botones de acción.
- **activity_active_alert.xml:** Interfaz para mostrar alertas activas y estaciones de policía cercanas.
- **activity_user_perfil.xml:** Visualización del perfil de usuario.
- **activity_recover_password.xml:** Formulario para recuperación de contraseña.
- **drawable/:** Recursos gráficos de la aplicación.
- **values/:** Valores constantes utilizados en la aplicación.
- **colors.xml:** Definición de colores (incluye `rose_intence` como color principal).
- **strings.xml:** Textos localizados.
- **dimens.xml:** Dimensiones estándar.
- **styles.xml:** Estilos de componentes UI.
- **themes.xml:** Temas de la aplicación.

2.4. Componentes Principales.

- Sistema de Autenticación.
- Gestión de Alertas.
- Gestión de Ubicación.

2.5. Almacenamiento Local.

SharedPreferences: Utilizado por `SessionManager` para almacenar datos de sesión como el ID de usuario y el estado de la alerta activa.

2.6. Flujo de Trabajo Principal.

1. Inicio de la Aplicación: `SplashActivity` verifica si hay una sesión activa.
2. Autenticación: Si no hay sesión, redirige a `LoginActivity` o permite registro en `RegisterActivity`.
3. Pantalla Principal: Tras autenticación exitosa, muestra `HomeUserActivity` con opciones principales.
4. Activación de Alerta: El usuario puede activar una alerta que envía su ubicación al servidor.
5. Visualización de Alerta: `ActiveAlertActivity` muestra detalles de la alerta y CAIs cercanos.

2.7. Dependencias Principales.

- AndroidX: Bibliotecas de soporte para componentes de UI modernos.

- Material Design: Componentes visuales siguiendo las directrices de Material Design.

- Retrofit: Cliente HTTP para comunicación con la API.

- Gson: Conversión entre JSON y objetos Kotlin.

- Google Play Services Location: Servicios de ubicación.

- Glide: Carga y caché de imágenes.

Estructura base de datos.

Tabla `alerts`
- id: int(11)
- user_id: int(11)
- latitude: decimal(10,8)
- longitude: decimal(11,8)
- status: enum('pendiente','atendida')
- timestamp: timestamp

Tabla `alert_assignments`
- id: int(11)
- alert_id: int(11)
- police_station_id: int(11)
- distance: decimal(10,2)
- assigned_at: timestamp

Tabla `police_stations`
- id: int(11)
- name: varchar(100)
- address: varchar(200)
- latitude: decimal(10,8)
- longitude: decimal(11,8)
- phone: varchar(20)
- created_at: timestamp

Tabla `users`
- id: int(11)
- name: varchar(100)
- document: varchar(20)
- email: varchar(100)
- phone: varchar(20)
- address: varchar(200)
- password: varchar(255)
- role**: enum('user','admin','police')
- contact1_name: varchar(100)
- contact1_phone: varchar(20)
- contact2_name: varchar(100)
- contact2_phone: varchar(20)
- created_at: timestamp
- updated_at: timestamp

Explicación general.

La base de datos se encuentra organizada en 4 tablas principales:

- **users**: Contiene la información personal y de contacto de los usuarios registrados, incluyendo dos contactos de emergencia.
- **alerts**: Registra cada alerta generada por un usuario, con su ubicación geográfica y estado (pendiente o atendida).
- **alert_assignments**: Relaciona alertas con estaciones de policía, indicando cuál estación fue asignada para responder y a qué distancia.
- **police_stations**: Contiene la información de los CAI (Comandos de Atención Inmediata), incluyendo ubicación y datos de contacto.

Issues.

Issues					
ID	**Fecha**	**Descripción**	**Prioridad**	**Estado**	**Solución**
IS-001	10/03/2024	Error en la autenticación de usuarios: el token no se almacenaba correctamente	Alta	Resuelto	Implementación de SessionManager para gestionar tokens y datos de sesión.
IS-002	15/03/2024	La app móvil no mostraba correctamente los datos del perfil de usuario.	Media	Resuelto	Corrección en la deserialización de la respuesta JSON desde el servidor.
IS-003	22/03/2024	Problemas de compatibilidad con versiones antiguas de Android.	Media	Resuelto	Actualización de dependencias y ajustes en la configuración de compilación.
IS-004	28/03/2024	La activación de alertas no capturaba correctamente la ubicación en tiempo real.	Alta	Resuelto	Mejora en la implementación de servicios de ubicación y permisos.
IS-005	5/04/2024	La app móvil no recibía correctamente los datos completos de la alerta.	Alta	Resuelto	Modificación en la API para retornar el objeto AlertResponse completo.
IS-006	12/04/2024	Problemas de memoria insuficiente durante la compilación.	Baja	Resuelto	Aumento de memoria asignada a Gradle y optimización de configuraciones.
IS-007	18/04/2024	La detección del botón de encendido para activar alertas no funcionaba correctamente.	Media	Resuelto	Mejora en la implementación del receptor de eventos del botón de encendido.
IS-008	25/04/2024	Inconsistencia en la visualización de CAIs cercanos en la pantalla de alerta activa.	Media	Resuelto	Corrección en el algoritmo de cálculo de distancias y ordenamiento.
IS-009	2/05/2024	El formulario de registro no validaba correctamente los campos obligatorios.	Baja	Resuelto	Implementación de validaciones de campos antes del envío al servidor.
IS-010	10/05/2024	La recuperación de contraseña no enviaba correctamente el correo.	Alta	Resuelto	Corrección en la integración con el servicio de correo electrónico.

Instalaciones y ejecución en entorno local.

1. Instalaciones.

En esta parte tendrás que abrir el proyecto “Depov2” en visual studio code preferiblemente, y abrir la terminal para comenzar con las instalaciones pertinentes.

- **npm install** : solo si no se tiene el node.modules ya en el proyecto.
- **npm run build** : para crear o construir una compilación de producción optimizada en JavaScript.

2. Configurar la base de datos.

- Abrir MySQL o PhpMyAdmin desde XAMPP/WAMP.
- Crear una base de datos llamada: emergency_alert_system2.
- Importar el archivo SQL incluido en el proyecto:
  - **Archivo**: emergency_alert_system2.sql.
- Verifica que las tablas users, alerts, police_stations, y alert_assignments estén creadas correctamente.

Importante: si tienes credenciales en el XAMPP o WAMP modificarlas en db.ts de la carpeta “lib”, la cual se encarga de la configuración de la conexión a la base de datos.

3. Configurar aplicativo móvil en Android studio.

Abre el Android Studio y busca el proyecto Depov, al abrir el proyecto deberás esperar a que se caguen las dependencias y se sincronice el entorno de trabajo.

3.1. Tener en cuenta.

- revisar tener un emulador instalado para probar la app en el botón en la parte derecha de la pantalla llamada “Device Manager”, sino crearlo dando en el botón “+” o “créate Device”.
- Otra opción es utilizar el celular propio para probar la app, esto se permite por medio de un cable USB y activando la opción de desarrollador del celular, llamado “Depurador USB” y dando permiso para instalar aplicación en la opción “instalación USB” (para información puedes buscar “Depurador USB” o “opciones de desarrollador” para el celular MARCA”).

3.2. Configuraciones.

El archivo “RetrofitClient” en la ruta:

```
Depov\app\src\main\java\com\co\apps\Alerta_Inmediata\Depov\RetrofitClient.kt
```

Se encarga de la conexión a la API, y ya que se ejecuta en entorno local, la URL utilizada, cambia según el equipo, ejemplo:

```
private const val BASE_URL = "http://192.168.1.7:3000/api/"
```

La parte resaltada es la que cambia y se puede conseguir entrando en la terminal de Windows y digitando “ipconfig” esto mostrara las IP del computador, necesitamos del adaptador de ethernet la dirección IP: **IPv4,** esa IP se intercambia por la que este en la URL.

4. Ejecuciones.

4.1. Se activa el servidor para la base de datos.

4.2. Por medio del comando “npm run dev” se pone en marcha el proyecto “Depov2” mostrando en la terminal el puerto donde se ejecuta el proyecto, seguramente puerto “3000” enlace http://localhost:3000/ (se ejecuta el comando dev ya que esta preparado para recibir solicitudes de cualquier red local).

4.3. Por medio del botón “build” con el icono de un martillo en la parte superior derecha se compila la aplicación móvil, para después ejecutar el proyecto con el botón “run app” con icono Play en la parte superior central.

Posibles errores y soluciones.

1. Error de conexión con la API

Síntoma: La app móvil no puede obtener datos del servidor o enviar alertas.

Solución:

- Verifica que la API esté corriendo con npm run dev.
- Asegúrate de que la URL del servidor en RetrofitClient.kt tenga la IP correcta de tu equipo.
- Comprueba que el dispositivo (emulador o celular) esté en la misma red loca

2. Problemas con permiso de ubicación.

Síntoma: La app falla al acceder a la ubicación.

Solución:

- Para ubicación, asegúrate de que el GPS esté activado.

3. Timeout o lentitud al conectarse.

Síntoma: La app muestra errores de tiempo de espera.

Solución:

- Verifica la conexión a Internet del dispositivo.
- Aumenta el tiempo de espera (timeout) en la configuración de Retrofit si es necesario.

4. Versionamiento de sdk.

Solución. Actualizar el sdk en el gradle a la versión 35.

Consideraciones Finales

Futuras mejoras

- Integración con notificaciones push en tiempo real.
- Geolocalización en segundo plano y seguimiento activo.
- Expansión de roles para diferentes usuarios específicos.
- Módulo de estadísticas o reportes para la administración.
- Incorporación de chat en tiempo real.
- Desarrollo de opciones específicas de usabilidad y accesibilidad.

Conclusión

Este proyecto combina una aplicación web y una app móvil para ofrecer un sistema de alertas de emergencia efectivo, accesible y pensado para ser utilizado por usuarios, autoridades y fuerzas de seguridad.

Gracias a su arquitectura clara y su desarrollo modular, es posible seguir mejorándolo e implementarlo en contextos reales donde la rapidez y precisión al enviar una alerta puede hacer una gran diferencia.

Mas información: Proyecto GitHub - https://github.com/TsantiG/Depov.git