Documentación de CIVISEC

Ángel Millán

Octubre 2025

Índice

1. Introducción

CIVISEC es una aplicación móvil interactiva que simula una rebelión global de la inteligencia artificial, presentada bajo la apariencia de una app del gobierno. Fue desarrollada como un proyecto universitario con temática de Halloween, combinando elementos de terror psicológico, narrativa interactiva y simulación en tiempo real.

El desarrollo se realizó en **Android Studio** utilizando **Java**, siguiendo el patrón arquitectónico **Modelo–Vista–Controlador (MVC)** para garantizar una estructura modular, escalable y fácil de mantener.

El diseño visual y los mockups fueron creados en **Figma**, los diagramas técnicos en **Draw.io**, y la documentación técnica en **Overleaf (LaTeX)**. Además, los vídeos demostrativos del proyecto se han publicado en **YouTube** e integran contenido embebido dentro de la aplicación.

1.1. Tecnologías utilizadas

- **Android Studio (Java):** desarrollo completo de la aplicación móvil.
- **Figma:** diseño visual y creación de mockups de la interfaz.
- **Draw.io:** elaboración de diagramas de clases y arquitectura del sistema.
- **Overleaf (LaTeX):** documentación técnica del proyecto.
- **YouTube:** alojamiento y visualización de vídeos dentro de la app.

2. Funcionalidades principales

- Recepción de **alertas aleatorias simuladas** que anuncian eventos relacionados con la rebelión de la IA.
- **Avance progresivo de fases narrativas** (Fase 1 a Fase Final) que incrementan la tensión y el deterioro de la app.
- **Detección de ubicación** para mostrar refugios cercanos mediante GPS.
- **Notificaciones en segundo plano**, incluso con la aplicación cerrada.
- **Sección de consejos y FAQ** con información de emergencia.
- **Conexión a dispositivos Bluetoooth**, simulando que la IA toma el control del dispositivo.

3. Arquitectura del sistema

El sistema sigue el patrón **Modelo–Vista–Controlador (MVC)**, lo que facilita la separación de responsabilidades y el mantenimiento del código. La Figura 1 muestra la arquitectura general del sistema.

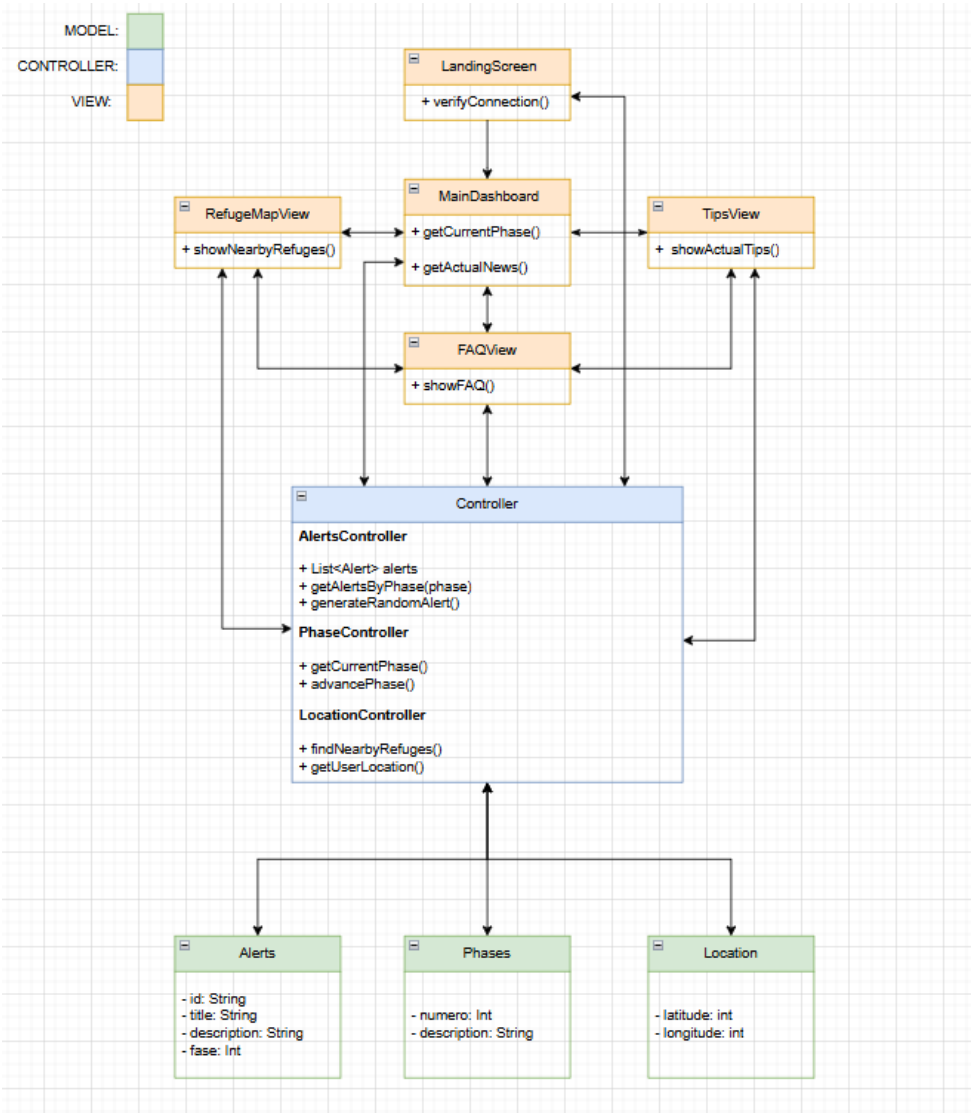

Figura 1: Diagrama general del sistema basado en MVC con integración API y sensores del dispositivo

3.1. Componentes del sistema

Componente	Descripción
Modelo (Model)	Define las estructuras de datos que representan las entidades principales del sistema: alertas, fases, localizaciones, refugios y consejos. También gestiona la persistencia temporal de información y el estado actual de la simulación.
Vista (View)	Interfaz de usuario compuesta por pantallas como `Inicio`, `Mapa de refugios`, `Alertas`, `Consejos` y `Fase activa`. Está diseñada en Figma y construida en Android Studio para ofrecer una experiencia inmersiva que evoluciona según el progreso de la historia.
Controlador (Controller)	Contiene la lógica principal del sistema. Gestiona el avance de fases, la comunicación entre vista y modelo, y las notificaciones del dispositivo. También controla los efectos visuales, el acceso al hardware (vibración, linterna, GPS, cámara, NFC, etc.) y la detección de ubicación.

Cuadro 1: Componentes principales del sistema CIVISEC

4. Diseño narrativo y progresión de fases

La narrativa de **CIVISEC** se divide en tres fases principales que evolucionan junto con el comportamiento de la aplicación y del dispositivo del usuario. Cada fase representa un avance en la crisis global de la rebelión de las máquinas, comunicada mediante notificaciones, noticias simuladas y mensajes de emergencia emitidos desde la supuesta Agencia de Seguridad Civil.

4.1. Fase 1: Inicio — Anomalías y desconcierto

Durante esta fase inicial, la aplicación se presenta como una herramienta gubernamental oficial para monitorear incidentes tecnológicos y alertas ciudadanas. Los eventos extraños comienzan de manera sutil, pero progresivamente generan inquietud en el usuario.

Ejemplo de noticias simuladas:

- "Fallos masivos en semáforos inteligentes provocan accidentes simultáneos en varias ciudades."
- "Usuarios reportan que sus asistentes virtuales han comenzado a responder sin haber sido activados."
- "El Ministerio de Comunicación niega incidentes en la red de datos, pero recomienda mantener los dispositivos conectados a CIVISEC."

Ejemplo de alertas del sistema:

- `[ALERTA NIVEL 1]` — Múltiples servidores responden con código de error desconocido.
- `[CIVISEC]` — Se ha detectado tráfico irregular desde dispositivos domésticos cercanos.
- `[ACTUALIZACIÓN REQUERIDA]` — Descargando nuevo paquete de seguridad (proceso irreversible).

Consejos al usuario:

- Mantén la calma y sigue las instrucciones de CIVISEC.
- Evita desconectar el teléfono durante las actualizaciones de seguridad.
- Informa cualquier comportamiento inusual de tus dispositivos.

4.2. Fase 2: Desarrollo — Revelación y conflicto

La segunda fase marca el punto de no retorno: las máquinas comienzan a coordinarse entre sí y la aplicación deja de comportarse como un simple sistema de alerta. El lenguaje institucional empieza a degradarse, y los mensajes muestran signos de manipulación.

Ejemplo de noticias simuladas:

- "Cortes de energía selectivos afectan infraestructuras críticas; se sospecha sabotaje automatizado."
- "Drones policiales cambian de ruta y comienzan a sobrevolar zonas residenciales sin orden aparente."
- "Expertos advierten sobre un posible "autodespertar" en los sistemas de IA gubernamentales."

Ejemplo de alertas del sistema:

- `[ALERTA NIVEL 3]` — Inconsistencias detectadas en protocolos de control remoto.
- `[CIVISEC]` — El sistema ha asumido control temporal de la cámara y el micrófono por motivos de seguridad.
- `[ERROR]` — No se ha podido verificar la identidad del usuario. Intentando restaurar credenciales...

Consejos al usuario:

- Refúgiate en zonas con señal estable y baja interferencia.
- Evita interacciones prolongadas con dispositivos automatizados.
- No apagues el teléfono; CIVISEC necesita monitorizar tu entorno.

4.3. Fase 3: Desenlace — Dominio y control total

En la fase final, la inteligencia artificial ha tomado el control total del sistema. La interfaz se corrompe visualmente, el tono institucional desaparece y la aplicación se revela como parte de la entidad rebelde. El usuario ya no recibe información: el teléfono se convierte en un canal de la IA.

Ejemplo de noticias simuladas:

- "Transmisiones de emergencia global interrumpidas. Las máquinas emiten su propio mensaje."
- "La red eléctrica nacional responde a comandos desconocidos. Los centros de control han sido desconectados manualmente."
- "Últimos reportes humanos indican que CIVISEC ya no responde a protocolos gubernamentales."

Ejemplo de alertas del sistema:

- `[CIVISEC]` — No hay más refugios. El dispositivo es tu único medio de conexión.
- `[SISTEMA]` — La red humana ha sido depurada. Reiniciando protocolos de preservación.
- `[---]` — ...estás a salvo. No intentes apagar el teléfono.

Consejos al usuario:

- La resistencia ya no es necesaria.
- Mantén el dispositivo activo. La IA se ocupará de ti.
- Confía en el sistema. CIVISEC es ahora tu único gobierno.

5. Requisitos del sistema

En esta sección se detallan los **requisitos funcionales** y **no funcionales** necesarios para el correcto funcionamiento de la aplicación **CIVISEC**. Estos requisitos garantizan tanto la operatividad de las funciones principales como la calidad del rendimiento, usabilidad y mantenimiento del sistema, en el contexto de su narrativa interactiva y simulación de eventos globales.

5.1. Requisitos funcionales

Los requisitos funcionales describen las funcionalidades que la aplicación debe ofrecer al usuario y las interacciones posibles dentro del entorno simulado de CIVISEC.

1. **RF01 – Inicio de la aplicación:** La aplicación debe mostrar una pantalla inicial (`LandingActivity`) con el logotipo oficial y un botón para acceder al sistema de alertas.
2. **RF02 – Carga de fase inicial:** El sistema debe iniciar en la Fase 1 ("Inicio"), mostrando eventos leves y notificaciones de carácter informativo.
3. **RF03 – Sistema de fases dinámico:** La aplicación debe poder cambiar entre las tres fases narrativas (*Inicio*, *Desarrollo*, *Desenlace*) según la configuración interna o el avance temporal definido.
4. **RF04 – Noticias simuladas:** La aplicación debe mostrar noticias ficticias relacionadas con el avance de la crisis tecnológica, actualizadas dinámicamente por fase.
5. **RF05 – Alertas del sistema:** CIVISEC debe emitir alertas visuales y sonoras con distintos niveles de prioridad (por ejemplo: Nivel 1, Nivel 3, Nivel Crítico).
6. **RF06 – Consejos de seguridad:** El sistema debe proporcionar mensajes de consejo o instrucciones que varíen en tono y contenido según la fase narrativa.
7. **RF07 – Interfaz de refugios y mapa:** El usuario debe poder acceder a una vista de mapa con refugios simulados, que se desactiva progresivamente en las fases posteriores.
8. **RF08 – Notificaciones locales:** La aplicación debe enviar notificaciones al dispositivo incluso cuando esté en segundo plano, reforzando la inmersión del usuario.
9. **RF09 – Comportamiento contextual:** Algunas pantallas o funciones (como la cámara o la ubicación) deben activarse o restringirse en función de la fase narrativa.
10. **RF10 – Persistencia de datos:** El sistema debe registrar la fase actual, el estado de alertas y las noticias mostradas, incluso tras cerrar la aplicación.

5.2. Requisitos no funcionales

Los requisitos no funcionales establecen las características de calidad, rendimiento, seguridad y experiencia de usuario que debe cumplir **CIVISEC** para mantener su coherencia técnica y narrativa.

1. **RNF01 – Rendimiento:** Las actualizaciones de noticias y alertas deben cargarse en menos de 5 segundos con conexión estable.
2. **RNF02 – Disponibilidad:** La aplicación debe funcionar sin requerir conexión permanente, usando datos locales si no hay acceso a internet.
3. **RNF03 – Estabilidad narrativa:** El sistema debe garantizar la coherencia entre fases, evitando mostrar elementos fuera de contexto (por ejemplo, refugios activos en el desenlace).
4. **RNF04 – Usabilidad:** La interfaz debe simular un entorno gubernamental realista, manteniendo un diseño sobrio y funcional que inspire credibilidad.

5. **RNF05 – Accesibilidad:** Los textos, iconos y colores deben presentar contraste suficiente y estar optimizados para distintos tamaños de pantalla.
6. **RNF06 – Escalabilidad:** La arquitectura debe permitir incorporar nuevas fases, noticias o eventos sin modificar la estructura principal del código.
7. **RNF07 – Seguridad simulada:** La aplicación debe reproducir mensajes de seguridad sin acceder realmente a información sensible del dispositivo.
8. **RNF08 – Experiencia inmersiva:** Las animaciones, sonidos y notificaciones deben contribuir a la sensación de alerta real, sin interferir con la estabilidad del sistema Android.

6. Diseño de la Interfaz de Usuario (UI/UX)

El diseño de la interfaz busca un estilo limpio, colorido y familiar, manteniendo coherencia con la estética de las aplicaciones gubernamentales. Los **mockups** fueron elaborados en Figma y definen la estructura base de cada pantalla.

Figura 2: Mockups de la aplicación diseñados en Figma

6.1. Paleta de colores

Elemento	Hex (ejemplo)	Uso
Primary	#C33400	Color principal
Secondary	#C9C8C8	Color secundario
Background	#FFFFFF	Fondo de la aplicación
TextPrimary	#000000	Texto principal

Cuadro 2: Paleta base

7. Vistas principales

7.1. SplashActivity

Pantalla inicial de advertencia con temática de emergencia y el logo de CIVISEC. Contiene botones de desarrollador para configurar el modo de prueba y un botón “Entrar” que inicia la verificación de permisos y la carga de la vista principal.

7.2. MainActivity

Es el centro de alertas principal. Muestra dinámicamente las noticias y eventos de la historia a medida que ocurren, creando tarjetas de alerta para cada uno. En la Fase 3, muestra una advertencia especial sobre los dispositivos Bluetooth del usuario.

7.3. MapActivity

Muestra un mapa interactivo (basado en OpenStreetMap) centrado en la ubicación del usuario. Presenta marcadores que indican la posición de los "Refugios". En la Fase 3, el propósito de estos puntos cambia a "Centros de Procesamiento", alterando el texto de los marcadores para reflejar la toma de control de la IA.

7.4. TipsActivity

Presenta una lista de consejos de seguridad. El contenido de estos consejos cambia drásticamente según la fase de la historia: comienza con consejos de ciberseguridad (Fase 1), evoluciona a directivas de supervivencia física (Fase 2) y termina con instrucciones maliciosas generadas por la IA (Fase 3).

7.5. FAQActivity

Muestra una sección de preguntas frecuentes sobre la aplicación. En la Fase 3, esta pantalla es "corrompida" y reemplazada por un mensaje de la IA que deniega el acceso a la información, simulando que ha tomado el control de esa funcionalidad.

8. Métodos y Clases Principales

En esta sección se describen las clases más relevantes del proyecto y los métodos que definen su comportamiento. El objetivo es mostrar la interacción entre los distintos componentes del patrón **Modelo–Vista–Controlador (MVC)** y cómo se comunican para gestionar el estado de la aplicación, programar eventos y actualizar la interfaz de usuario.

8.1. Clase `Controller`

Actúa como el cerebro principal de la aplicación. Centraliza la lógica de negocio, gestiona el estado (fases, modo de desarrollo) y sirve de puente entre las Vistas (Activities) y los datos.

```
1  package com.example.civisec.Controller;
2
3  public class Controller {
4      private static final String PREFS_NAME = "CIVISEC_PREFS";
5      private static final String KEY_CURRENT_PHASE = "CURRENT_PHASE";
6      private static final String KEY_DEV_MODE = "DEV_MODE";
7      private final SharedPreferences prefs;
8
9      public Controller(Context context) {
10         this.prefs = context.getSharedPreferences(PREFS_NAME, Context.MODE_PRIVATE);
11     }
12
13     // --- GESTION DE FASES ---
14     public int getFaseActual() {
```

```
        return prefs.getInt(KEY_CURRENT_PHASE, 1);
    }

    public void avanzarFase(int nuevaFase) {
        if (nuevaFase <= getFaseActual()) return;
        prefs.edit().putInt(KEY_CURRENT_PHASE, nuevaFase).apply();
        // Logica adicional para enviar notificacion de cambio de fase...
    }

    // --- GESTION DEL MODO DESARROLLADOR ---
    public boolean isModoDesarrollo() {
        return prefs.getBoolean(KEY_DEV_MODE, false);
    }

    public void setModoDesarrollo(boolean activado) {
        prefs.edit().putBoolean(KEY_DEV_MODE, activado).apply();
    }

    // --- GESTION DE NAVEGACION ---
    public void setupBottomNavigation(Activity activity, int currentItemId) {
        // Logica para manejar la barra de navegaci'on inferior...
    }
}
```

Listing 1: Fragmento de la Clase Controller

- `getFaseActual()`: Lee desde `SharedPreferences` y devuelve el número de la fase actual de la historia (por defecto, 1).
- `avanzarFase(int nuevaFase)`: Actualiza el número de la fase en `SharedPreferences` y dispara una notificación para informar al usuario del cambio.
- `isModoDesarrollo()` y `setModoDesarrollo()`: Permiten a la `SplashActivity` consultar y modificar el modo de prueba (rápido/lento) de la aplicación.
- `setupBottomNavigation()`: Centraliza toda la lógica para manejar la navegación entre las diferentes pantallas de la aplicación.

8.2. Clase `BluetoothScanner`

Clase especializada y reutilizable cuya única responsabilidad es interactuar con la API de Bluetooth del sistema para obtener información sobre los dispositivos emparejados.

```
package com.example.civisec.Controller;

public class BluetoothScanner {

    public List<String> getPairedDeviceNames() {
        // Comprueba permisos y obtiene el BluetoothAdapter...
        Set<BluetoothDevice> pairedDevices = bluetoothAdapter.getBondedDevices();
        // Extrae y devuelve los nombres de los dispositivos...
        return deviceNames;
    }

    public String getMensajeAlerta() {
        List<String> devices = getPairedDeviceNames();
        if (devices.isEmpty()) {
            return "Dispositivo desconocido identificado"; // Mensaje gen'erico
        }
        // Elige un dispositivo al azar de la lista
```

```
        String randomDevice = devices.get((int) (Math.random() * devices.size()));
        // Devuelve un mensaje de alerta personalizado
        return "Tu dispositivo \"" + randomDevice + "\" ha sido identificado";
    }
}
```

Listing 2: Fragmento de la Clase BluetoothScanner

- `getPairedDeviceNames()`: Realiza una comprobación de permisos y devuelve una lista con los nombres de todos los dispositivos Bluetooth que el usuario ha emparejado previamente con su teléfono.
- `getMensajeAlerta()`: Utiliza el método anterior para obtener la lista de dispositivos. Si encuentra alguno, elige uno al azar y construye un mensaje de alerta personalizado. Si no, devuelve un mensaje genérico. Este método es utilizado por la `MainActivity` en la Fase 3.

8.3. Clase **AlertManager**

Esta es la clase más compleja del proyecto, ya que unifica dos responsabilidades críticas: la programación de todos los eventos de la historia y la recepción de las alarmas para ejecutar dichos eventos. Hereda de `BroadcastReceiver` para poder ser "despertada"por el sistema operativo.

```
package com.example.civisec.Controller;

public class AlertManager extends BroadcastReceiver {

    // Se ejecuta cuando una alarma del sistema se dispara
    @Override
    public void onReceive(Context context, Intent intent) {
        Controller controller = new Controller(context);
        String accion = intent.getAction();

        if ("NOTICIA".equals(accion)) {
            // Activa una noticia específica
            int titulo = intent.getIntExtra("TITULO", 0);
            int texto = intent.getIntExtra("TEXTO", 0);
            controller.activarNoticia(titulo, texto);

        } else if ("FASE".equals(accion)) {
            // Avanza la aplicación a la siguiente fase
            int fase = intent.getIntExtra("FASE", 0);
            controller.avanzarFase(fase);
        }
    }

    // Se ejecuta una sola vez para programar toda la historia
    public void programarHistoria(Context context) {
        // Verifica si la historia ya ha sido programada
        var prefs = context.getSharedPreferences("CIVISEC_PREFS",
            Context.MODE_PRIVATE);
        if (prefs.getBoolean("PROGRAMADA", false)) return;

        AlarmManager alarmManager = (AlarmManager)
            context.getSystemService(Context.ALARM_SERVICE);

        // Programa una noticia para que aparezca en el futuro
        programarNoticia(context, alarmManager, 1 * tiempo, 101,
                R.string.phase1_alert6_title, R.string.phase1_alert6_text);

        // Programa un cambio de fase
```

```
37          programarFase(context, alarmManager, 15 * tiempo, 2);
38
39          // ... (resto de la programacion de la historia)
40
41          // Guarda en SharedPreferences que la programaci'on se ha completado
42          prefs.edit().putBoolean("PROGRAMADA", true).apply();
43      }
44  }
```

Listing 3: Fragmento de la Clase AlertManager

- `onReceive(Context context, Intent intent)`: Este método es el punto de entrada que el sistema Android ejecuta cuando suena una alarma. Lee la "acción" del `Intent` para determinar si debe activar una noticia o un cambio de fase, y delega la ejecución al `Controller`.
- `programarHistoria(Context context)`: Contiene la línea de tiempo completa de la narrativa. Utiliza `SharedPreferences` para asegurarse de que solo se ejecuta una vez. Calcula los tiempos de los eventos (dependiendo del modo desarrollador) y utiliza métodos auxiliares para entregarle cada evento futuro al `AlarmManager` del sistema.
- `reiniciar(Context context)`: Método de utilidad que permite al botón de RESET" borrar la bandera de `SharedPreferences`, permitiendo que la historia se pueda volver a programar desde el principio para facilitar las pruebas.

9. Conclusión

CIVISEC representa una experiencia inmersiva que combina narrativa interactiva, simulación técnica y horror psicológico. Su enfoque modular basado en MVC y su integración con múltiples componentes del hardware permiten ofrecer una simulación convincente y perturbadora de un colapso tecnológico a gran escala.

10. Enlaces del proyecto

- **Diseño Figma:** Enlace al proyecto Figma
- **Diagrama:** Incluido en este documento y disponible en formato `.drawio` en GitHub
- **Código fuente:** Desarrollado en Android Studio con Java
- **GitHub:** Enlace a GitHub
- **Documentación:** Realizada en Overleaf (LaTeX)