SmartCane App

Di Nicco, Luis Demetrio; Antonioli, Iván Oscar; Nahuel Ezequiel López Ferme;
Kevin Sanchez; Intrieri, Gabriel Yamil
43664669; 43630151; 43991086; 41173649; 38128822
Comisión del día Martes 02-2900, Grupo M1

[1]Universidad Nacional de La Matanza,
Departamento de Ingeniería e Investigaciones Tecnológicas,
Florencio Varela 1903 - San Justo, Argentina

Resumen: SmartCane App es una aplicación móvil complementaria al bastón inteligente SmartCane, diseñada para personas con discapacidad visual y sus familiares. En el rol de usuario no vidente, permite encender/apagar el bastón de forma remota y recibir alertas sobre obstáculos detectados, así como su posición. En el rol de familiar, la app muestra el estado del dispositivo (encendido, apagado), notifica si se ha activado la alarma por caída o pérdida del bastón, y en caso de emergencia, abre automáticamente la ubicación del usuario no vidente en Google Maps para facilitar una respuesta rápida. Además, ambos roles pueden consultar el historial de pasos recorridos. SmartCane App promueve la autonomía, seguridad y acompañamiento activo, integrando accesibilidad y conectividad en una solución tecnológica inclusiva.

Palabras claves: Bastón inteligente, Discapacidad visual, Movilidad asistida, Detección de obstáculos, MQTT, Android, Mobile, IoT

1 Introducción

SmartCane App es una aplicación móvil desarrollada como complemento esencial del bastón inteligente SmartCane, extendiendo su funcionalidad mediante una arquitectura IoT distribuida que mejora la experiencia del usuario con discapacidad visual y facilitar la asistencia de un familiar en tiempo real. Mientras el bastón inteligente proporciona detección activa de obstáculos y alertas sensoriales al usuario, la aplicación actúa como un nodo clave dentro de esta arquitectura IoT, gestionando la comunicación, supervisión y control remoto del dispositivo a través del protocolo MQTT (Message Queuing Telemetry Transport).

Esta arquitectura se basa en un enfoque cliente-servidor, donde la app móvil y el bastón se comunican de forma bidireccional mediante un broker MQTT, que permite el intercambio eficiente, ligero y en tiempo real de mensajes.

La persona con discapacidad visual utiliza la aplicación SmartCane App en su smartphone para interactuar de forma sencilla e intuitiva con el bastón. Desde esta interfaz puede:

- Encender o apagar el bastón remotamente, útil cuando el usuario necesita conservar energía o gestionar el dispositivo sin necesidad de manipularlo físicamente.
- Recibir notificaciones activas cuando el bastón detecta obstáculos. Estas notificaciones llegan con información contextual sobre la dirección del objeto detectado, contribuyendo a una mejor toma de decisiones.
- Consultar el seguimiento de pasos, que le permite tener un registro diario o histórico de su actividad física y desplazamientos. Esta funcionalidad se basa en la API de reconocimiento de actividad física del sistema Android y para dicho fin es necesario que el usuario le brinde los permisos pertinentes.

Además, la app también solicita permisos para acceder a la ubicación del usuario. Esta información se utiliza exclusivamente en caso de emergencia, cuando el sistema detecta que el bastón ha sido soltado inesperadamente (mediante el sensor de presión incorporado). En ese caso, la aplicación comparte automáticamente la ubicación geográfica del usuario a través del broker MQTT con la interfaz del familiar, permitiendo una respuesta rápida y eficiente.

El rol del familiar ha sido especialmente diseñado para brindar asistencia y monitoreo sin invadir la autonomía del usuario. Desde su perfil en la app, el familiar puede:

- Visualizar el estado del bastón en tiempo real, incluyendo si se encuentra encendido, apagado, o si ha sido activada alguna alarma por caída o pérdida.
- Recibir alertas de emergencia en caso de que el bastón haya sido soltado repentinamente. Esta alerta viene acompañada de la ubicación precisa del usuario, abriendo automáticamente la ubicación en Google Maps, facilitando la asistencia inmediata.
- Monitorear el conteo de pasos, proporcionando una referencia del nivel de actividad del usuario, útil para el seguimiento del bienestar físico general.

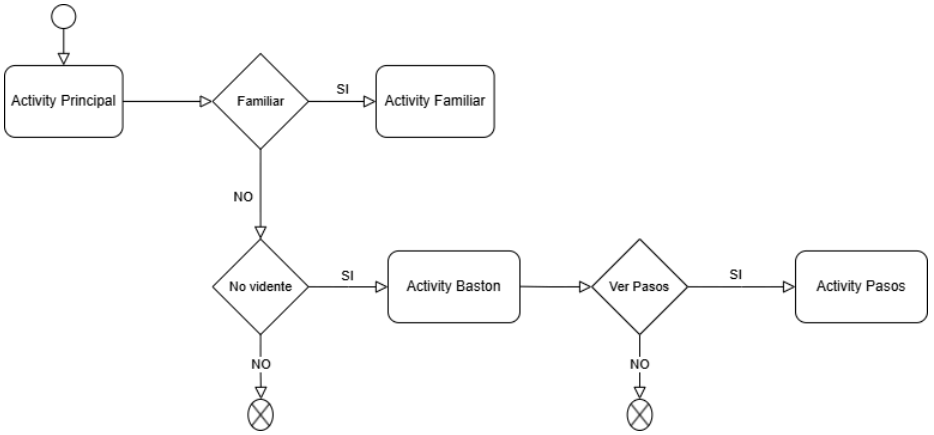
Estas funciones están diseñadas para mantener un canal de comunicación no invasivo pero efectivo entre el usuario y su red de apoyo. Por ello SmartCane App está diseñada con un enfoque en la privacidad del usuario, asegurando que la ubicación solo se transmite en contextos de alerta o riesgo.

2 Desarrollo

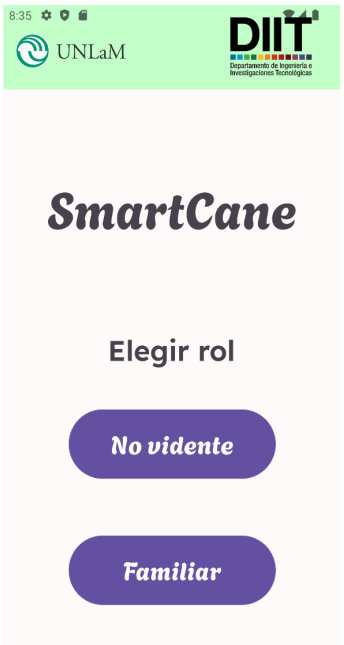
URL del repositorio GitHub del proyecto:
https://github.com/UNLAM-SOA/2025_SOA_Q1_M1

URL del proyecto en WOKWI:
https://wokwi.com/projects/435028546370994177

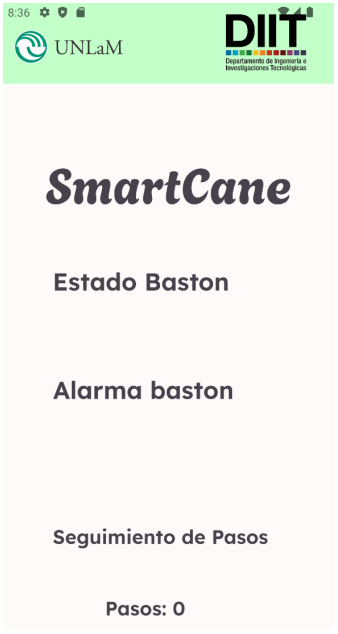
Diagrama Funcional/Navegación de las Activities:



Activity Principal:



Activity Familiar:



Activity Baston:

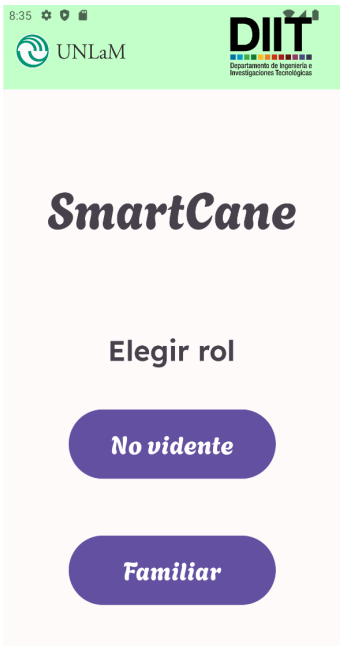
Activity Pasos:

3 Manual de Usuario

Descripción General

SmartCane App es una aplicación Android diseñada para complementar el funcionamiento del bastón inteligente SmartCane. Ofrece dos modos de uso (roles): “Usuario no vidente” y “Familiar”. A través de una interfaz accesible y sencilla, permite interactuar con el bastón, recibir información sobre obstáculos, monitorear la actividad física y reaccionar ante emergencias.

Pantalla Principal:



Al iniciar la aplicación, se presenta la pantalla principal con las siguientes opciones:

- **No Vidente**
- **Rol Familiar**

Cada botón lleva al usuario a su respectiva interfaz con funciones específicas según su rol.

Pantalla para el usuario no vidente:

Esta sección está pensada para el usuario con discapacidad visual. Contiene funciones clave para el uso diario del bastón.

Botón "Encender":

Activa el bastón inteligente a través de una conexión MQTT. Permite comenzar la detección de obstáculos y la recolección de datos.

Botón "Apagar":

Desactiva el bastón de forma remota. Útil para conservar batería o pausar su funcionamiento.

Detección de Obstáculos:

Muestra en tiempo real el estado de detección del bastón. Las leyendas posibles son:

- **"obstacle_right":** Obstáculo detectado a la derecha.
- **"obstacle_left":** Obstáculo detectado a la izquierda.
- **"obstacle_both":** Obstáculos detectados en ambos lados.

- **"no_obstacle":** No se detecta ningún obstáculo cercano.

Importante: Las leyendas de detección de obstáculos se integran con un sistema de síntesis de voz , que permite su reproducción en audio de forma automática. Esta funcionalidad garantiza la accesibilidad del contenido para el usuario no vidente, eliminando la necesidad de interacción visual con la pantalla.

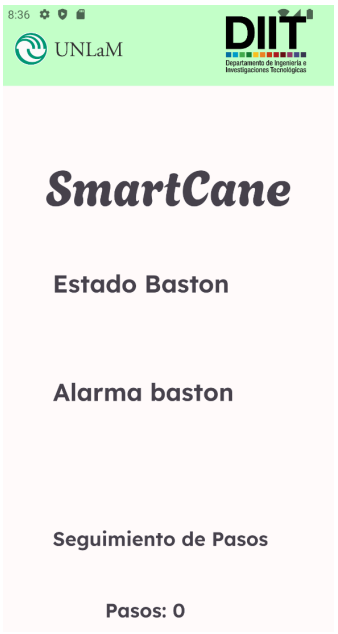
Botón de activación/desactivación de voz:

Permite silenciar o activar el sistema de síntesis de voz que lee el contenido del cuadro de texto con las alertas de obstáculos.

Botón "Ver Pasos":

Accede a la Pantalla de visualización del conteo de pasos registrados por el sistema Android, permitiendo al familiar hacer un seguimiento de su movilidad diaria:

Pantalla para el usuario familiar:



Diseñada para permitir el seguimiento y la asistencia remota del usuario no vidente.

Estado del Bastón:

Muestra el estado actual del bastón inteligente. Las leyendas posibles son:

- **"ON":** El bastón está activado.
- **"OFF":** El bastón está desactivado.
- **"CANE_REL":** El sistema detectó que el bastón fue soltado, lo que puede indicar una posible caída o emergencia.

Alarma baston:

Informa si la alarma del bastón está sonando (por ejemplo, si se soltó inesperadamente). Los valores posibles son:

- **"ON":** La alarma está activada.
- **“OFF”:** La alarma está desactivada.

Permisos de la Aplicación:

Al instalar o abrir la app por primera vez, se solicitarán los siguientes permisos:

- **Internet**: Para conectarse con el servidor y enviar/recibir información del SmartCane.
- **Acelerómetro**: Para detectar los pasos del usuario no vidente.
- **Ubicación**: Para acceder a la localización exacta o aproximada del usuario no vidente en caso de emergencia.

Recomendaciones de Uso:

- Mantener el dispositivo con batería suficiente si se espera utilizar el bastón y la app por períodos prolongados.
- Verificar que el bastón esté conectado a Internet.
- Revisar regularmente el conteo de pasos para hacer seguimiento a la movilidad.
- En el rol de familiar, estar atento a posibles activaciones de alarma para actuar con rapidez.

4 Conclusiones

En el desarrollo del sistema IoT SmartCane (baston inteligente + App), se tuvieron que tomar ciertos recaudos para que el sistema sea tolerante a fallos. Uno de esos recaudos fue la Reconexión Automática de Wi-Fi desde el lado del sistema embebido para nunca perder la conexión entre el bastón y la aplicación cliente. De la misma forma se tuvo que asegurar que el sistema embebido se mantuviese conectado al broker MQTT, ya que tendía a desconectarse y se debía reintentar la conexión. Además también se implementaron estructuras condicionales para verificar que exista una conexión activa antes de enviar mensajes, para evitar sobrecarga en el sistema y posible errores de sobrecarga al reintentar indefinidamente el reenvío de mensajes. Por el lado de la aplicación, uno de los inconvenientes que surgieron fue el bloqueo del hilo principal por exceso de memory lick, generando que la app se cierre inesperadamente y muestre el mensaje “SmartCane isn’t responding”, que fue resuelto al generar hilos secundarios para las tareas que se podían ejecutar en segundo plano sin bloquear al hilo UI. Uno de los problemas más críticos que surgieron fue la desconexión de forma inesperada cuando se abrían múltiples Activities o se conectaban varios dispositivos. Tras investigar, se descubrió que el problema estaba en el uso de IDs de cliente MQTT hardcodeados en el código.

MQTT no bloquea conexiones con IDs duplicados, pero sí desconecta automáticamente al cliente anterior cuando se conecta uno nuevo con el mismo ID. Para resolverlo, se implementó una generación de ID única por dispositivo utilizando la clase SharedPreferences. De esta forma, cada instancia de la app mantiene un identificador único persistente, que solo se pierde si se desinstala la app. Esto permitió que múltiples dispositivos y Activities puedan conectarse al broker MQTT simultáneamente sin interferencias, solucionando completamente el conflicto de conexión. También surgieron errores relacionados con la obtención y el uso de la ubicación, especialmente cuando no se había otorgado el permiso de ubicación o los datos venían mal formateados. Para lo cual se agregó manejo de errores al parsear la ubicación y validaciones previas para asegurarse de que el valor recibido tenga el formato correcto (lat,lon). Además, se gestionaron los permisos de ubicación correctamente para evitar fallos en tiempo de ejecución. Por último, durante la recepción de los datos de pasos en la ActivityFamiliar, se presentaba un error debido a que el valor se enviaba como un String por el Broadcast, pero se intentaba tratar directamente como un int sin verificar o convertir apropiadamente. Esto causaba que el valor no se mostrara correctamente o incluso que la app se bloquee si el dato no era numérico. Se corrigió el código del receptor (ReceptorStepOperacion) para que primero valide si el valor recibido no es nulo y luego realice una conversión segura a entero usando Integer.parseInt(). También se añadió un valor por defecto en caso de error. Así se evitó que la app se caiga o que los datos se procesen de forma incorrecta.

Algunas lecciones que aprendimos a través de la resolución de este trabajo fueron:

- Evitar el hardcodeo de variables (en especial las que poseen valores críticos como identificadores). Hacerlo puede generar conflictos silenciosos difíciles de detectar, como desconexiones inesperadas.
- Utilizar identificadores únicos para dispositivos, para contemplar la conexión de múltiples clientes a la vez.
- La importancia de manejar adecuadamente la comunicación asíncrona para mantener una app estable y con buena experiencia de usuario.

- La gestión correcta del ciclo de vida de los componentes (como servicios y receptores) es clave para evitar memory lick.

- El manejo de permisos en Android para ser claro con el usuario y justificar su uso, con el fin de asegurar la aceptación.

- Implementar mensajes y notificaciones claras ayuda a mantener al usuario informado sobre el estado de la app, especialmente en casos de fallo o pérdida de conexión.

- Conocer cómo funciona la tecnología que se utiliza, ya que por medio del error del harcodeo de los IDs se comprendió el comportamiento predeterminado de MQTT ante IDs duplicados, el cual es desconectar al cliente anterior. Esto demostró lo esencial que es entender a fondo el funcionamiento de los protocolos y tecnologías que se integran en una app para poder prever problemas y diseñar correctamente desde el principio.
- Validar y controlar todos los datos externos. Muchos errores pueden venir por datos inesperados o mal formateados (por ejemplo, en la ubicación GPS). Se aprendió que siempre hay que validar los datos antes de usarlos en una app, y que es fundamental manejar excepciones para evitar cierres inesperados.
- Validar y controlar los tipos de datos al usar BroadcastReceiver. Se debe prestar suficiente atención al tipo de dato que se envía y se recibe por los Broadcast, para prevenir errores o imposibilidad de mostrar el valor correctamente.

5 Referencias

1. Android Developers, “Develop”, Android Developers. [Online]. Available: https://developer.android.com/develop?hl=es-419. [Accessed: 30-Jun-2025].

2. MQTT ORG, “MQTT - The Standard for IoT Messaging”, MQTT. [Online]. Available: https://mqtt.org/. [Accessed: 30-Jun-2025].

3. UNLaM - Sistemas Operativos Avanzados, “Android”, Wiki SOA-UNLaM. [Online]. Available:https://www.soa-unlam.com.ar/wiki/index.php/PUBLICO:Android. [Accessed: 30-Jun-2025].

4. UNLaM - Sistemas Operativos Avanzados, “Sistemas embebidos e Internet de las Cosas”, Wiki SOA-UNLaM. [Online]. Available: https://www.soa-unlam.com.ar/wiki/index.php/PUBLICO:Sistemas_embebidos_e_Internet_de_las_Cosas. [Accessed: 30-Jun-2025].