

---

Desarrollo de una aplicación para recorridos virtuales mediante el uso de realidad aumentada para visitas guiadas en el Centro de Innovación y Tecnología de la Universidad del Valle de Guatemala

---

Guillermo Santos





UNIVERSIDAD  
DEL VALLE  
DE GUATEMALA

Universidad del Valle de Guatemala  
Facultad de Ingeniería  
Ingeniería en Ciencia de la Computación y Tecnologías de la Información

**Desarrollo de una aplicación para recorridos virtuales mediante el uso de realidad aumentada para visitas guiadas en el Centro de Innovación y Tecnología de la Universidad del Valle de Guatemala**

**Desarrollo de interfaz gráfica para aplicación de recorridos virtuales en la plataforma iOS**

PARA OPTAR AL GRADO DE LICENCIATURA EN:  
Ingeniería en Ciencia de la Computación y Tecnologías de la Información

EN MODALIDAD DE MEGAPROYECTO TECNOLÓGICO

Guillermo Santos 191517

Guatemala  
noviembre del 2024

Vo.Bo.:

(f) \_\_\_\_\_  
Ing. Francisco Anzueto

Tribunal Examinador:

(f) \_\_\_\_\_

(f) \_\_\_\_\_

(f) \_\_\_\_\_

---

## Agradecimientos

---

El resultado de esta investigación representa para mí el fruto de un esfuerzo constante a lo largo de mi carrera universitaria, un esfuerzo nutrido por el apoyo invaluable de personas que deseo reconocer. En primer lugar, agradezco profundamente a mis padres, quienes me han respaldado en mi educación superior y me han motivado a seguir adelante, sin importar las dificultades que se presentaran. Extiendo también mi gratitud al Departamento de Ciencias de la Computación, que durante todo mi recorrido como estudiante me brindó las oportunidades necesarias para formarme como un profesional capaz de enfrentar desafíos y contribuir al desarrollo del país. Agradezco de manera especial a mi asesor de tesis, Francisco Anzueto Marroquín, por su orientación y la revisión de este trabajo, así como por compartir sus conocimientos en el área de desarrollo móvil, los cuales fueron fundamentales para tomar decisiones que llevaron a un mejor resultado final.

|  |             |
|--|-------------|
| <b>Agradecimientos</b>   | <b>III</b>  |
| <b>Lista de Figuras</b>  | <b>VII</b>  |
| <b>Resumen</b>   | <b>VIII</b> |
| <b>1. Introducción</b>   | <b>1</b>    |
| <b>2. Objetivos</b>  | <b>3</b>    |
| 2.1. Objetivo General . . . . .  | 3           |
| 2.2. Objetivos Específicos . . . . .   | 3           |
| <b>3. Justificación</b>  | <b>4</b>    |
| <b>4. Marco Teórico</b>  | <b>6</b>    |
| 4.1. Aplicaciones Móviles . . . . .  | 6           |
| 4.1.1. Aplicaciones con Localización en Interiores . . . . .                           | 6           |
| 4.2. Aplicaciones móviles para el sistema operativo iOS . . . . .                      | 8           |
| 4.2.1. Utilizando Lenguaje Swift para desarrollo móvil . . . . .                       | 8           |
| 4.2.2. SwiftUI: Una librería para desarrollo rápido de interfaces . . . . .            | 8           |
| 4.2.3. ARKit y RealityKit: Librerías para experiencias de Realidad Aumentada . . . . . | 9           |
| 4.2.4. Accediendo a sensores del celular . . . . .                                     | 10          |
| 4.3. Buenas prácticas en el desarrollo de aplicaciones móviles . . . . .               | 11          |
| 4.3.1. Arquitectura de la aplicación . . . . .   | 11          |
| 4.3.2. Organización del código . . . . .   | 12          |
| 4.3.3. Diseño . . . . .  | 12          |
| 4.3.4. Accesibilidad . . . . .   | 13          |
| 4.3.5. Control de versiones utilizando GIT . . . . .                                   | 14          |
| 4.3.6. Pruebas Unitarias . . . . .   | 14          |
| 4.4. Sensores UWB . . . . .  | 17          |
| 4.4.1. Compatibilidad de los Sensores UWB con celulares existentes                     | 17          |

|  |           |
|--|-----------|
| 4.4.2. Estimote Proveedor de sensores UWB . . . . .  | 17        |
| 4.4.3. Sensores BLE . . . . .  | 18        |
| 4.4.4. WebSocket . . . . .   | 19        |
| <b>5. Metodología</b>  | <b>20</b> |
| <b>6. Alcance</b>  | <b>23</b> |
| <b>7. Resultados y Discusión</b>   | <b>24</b> |
| 7.1. Prototipo de baja fidelidad . . . . .   | 24        |
| 7.2. Flujo de la aplicación . . . . .  | 26        |
| 7.3. Desarrollo de simulador de sensores . . . . .   | 27        |
| 7.4. Selección de un patrón de diseño para la aplicación y organización del código . . . . . | 31        |
| 7.5. Desarrollo de la aplicación utilizando simulador . . . . .                              | 32        |
| 7.6. Implementación de componentes principales de la interfaz gráfica . . . . .              | 34        |
| 7.7. Implementando funcionalidad de Tours . . . . .  | 39        |
| 7.8. Utilizando sensores UWB para reemplazar el simulador . . . . .                          | 42        |
| 7.9. Componente de realidad aumentada . . . . .  | 46        |
| 7.10. Combinando sensores y Realidad Aumentada para rutas no lineales .                      | 50        |
| 7.11. Aproximando ubicación utilizando el sensor más cercano . . . . .                       | 55        |
| 7.12. Limitaciones observadas . . . . .  | 56        |
| 7.13. Localización de la aplicación . . . . .  | 57        |
| 7.14. Paleta de colores para modo oscuro . . . . .   | 60        |
| 7.15. Pruebas Unitarias . . . . .  | 61        |
| <b>8. Conclusiones</b>   | <b>63</b> |
| <b>9. Recomendaciones</b>  | <b>64</b> |
| <b>Bibliografía</b>  | <b>68</b> |
| <b>Anexos</b>  | <b>69</b> |
| <b>A. Código Aplicación Móvil</b>  | <b>69</b> |
| <b>B. Código Interfaz Gráfica Simulador de Sensores</b>                                      | <b>70</b> |
| <b>C. Código Servidor para el Simulador de Sensores</b>                                      | <b>71</b> |
| <b>D. Demo Biblioteca</b>  | <b>72</b> |
| <b>E. Demo Cafetería</b>   | <b>73</b> |
| <b>F. Código para inicializar el servidor del simulador</b>                                  | <b>74</b> |

---

## Lista de Figuras

---

|       |  |    |
|-------|--|----|
| 4.1.  | Aplicación de Navegación en el Museo Americano de Historia Natural   | 7  |
| 4.2.  | Uso básico de SwiftUI . . . . .                                      | 9  |
| 4.3.  | Uso de CoreLocation para obtener la dirección del teléfono . . . . . | 10 |
| 4.4.  | Esfuerzo Requerido Para Nuevas Funcionalidades (1) . . . . .         | 15 |
| 4.5.  | Ejemplo de una prueba unitaria . . . . .                             | 16 |
| 4.6.  | Sensores UWB de Estimote . . . . .                                   | 18 |
| 4.7.  | Registro de conexión con los sensores . . . . .                      | 18 |
| 7.1.  | Prototipo de baja fidelidad con figuras . . . . .                    | 24 |
| 7.2.  | Diseño preliminar de la aplicación para iOS . . . . .                | 25 |
| 7.3.  | Diagrama de flujo - Funcionalidad Tour . . . . .                     | 26 |
| 7.4.  | Interfaz gráfica inicial simulador de sensores . . . . .             | 27 |
| 7.5.  | Interfaz gráfica con un sensor . . . . .                             | 28 |
| 7.6.  | Editando un sensor . . . . .   | 28 |
| 7.7.  | Interfaz gráfica con varios sensores . . . . .                       | 28 |
| 7.8.  | Depuración del servidor . . . . .                                    | 29 |
| 7.9.  | Servidor actualizando a los clientes . . . . .                       | 29 |
| 7.10. | Diagrama de interacción Simulador - App . . . . .                    | 30 |
| 7.11. | Organización del proyecto . . . . .                                  | 31 |
| 7.12. | Pantalla de prueba para los sensores . . . . .                       | 32 |
| 7.13. | Contrato de implementación para los sensores . . . . .               | 33 |
| 7.14. | Definición de estructura de un sensor . . . . .                      | 33 |
| 7.15. | Aplicación recibe información del simulador . . . . .                | 34 |
| 7.16. | Vista de progreso . . . . .  | 35 |
| 7.17. | Vista de progreso - Un punto ha sido visitado . . . . .              | 36 |
| 7.18. | Detalles de una parada . . . . .                                     | 37 |
| 7.19. | Distintos estados del indicador de distancia . . . . .               | 38 |
| 7.20. | Se muestra la distancia hacia la siguiente parada . . . . .          | 39 |
| 7.21. | El usuario ha llegado a la primera parada . . . . .                  | 40 |
| 7.22. | Se muestra una distancia hacia la siguiente parada . . . . .         | 40 |
| 7.23. | Tour finalizado . . . . .  | 41 |

|   |    |
|---|----|
| 7.24. Sensor en Biblioteca - a 2 metros . . . . .   | 42 |
| 7.25. Detalles de la primera parada . . . . .   | 43 |
| 7.26. Aplicación detecta que está cerca del sensor . . . . .  | 44 |
| 7.27. Aplicación detecta que se ha llegado al sensor . . . . .  | 45 |
| 7.28. Se muestra la dirección hacia la siguiente parada . . . . .   | 47 |
| 7.29. Se actualiza la dirección de la flecha . . . . .  | 48 |
| 7.30. La flecha desaparece cuando el usuario llega . . . . .  | 49 |
| 7.31. El <i>waypoint</i> w permite guiar al usuario entre el punto de interés A y B. . . . .                  | 50 |
| 7.32. El usuario ha llegado al punto inicial del tour, el salón N. . . . .                                    | 51 |
| 7.33. El usuario obtiene la dirección a seguir. . . . .   | 52 |
| 7.34. El usuario observa la nueva dirección una vez se ha acercado lo suficiente al <i>waypoint</i> . . . . . | 53 |
| 7.35. El usuario ha llegado al Salón Silencioso. . . . .  | 54 |
| 7.36. Herramienta <i>Localizable</i> . . . . .  | 57 |
| 7.37. Herramienta <i>Localizable</i> para Español . . . . .   | 57 |
| 7.38. Vista con localización en Español . . . . .   | 58 |
| 7.39. Vista con localización en Inglés . . . . .  | 59 |
| 7.40. Interfaz utilizada para configurar colores en <i>XCode</i> . . . . .                                    | 60 |
| 7.41. Vista de pruebas unitarias en <i>XCode</i> . . . . .  | 61 |

---

## Resumen

---

El objetivo principal de esta tesis era desarrollar una interfaz gráfica para el sistema operativo *iOS* que permita validar el uso de sensores de geolocalización interna para su futura incorporación en el proyecto de la aplicación de recorridos virtuales UVG. Esto se realizó utilizando sensores adquiridos, los cuales utilizan el protocolo de comunicación *UWB*. Para hacer posible la prueba de este concepto se llevó a cabo una metodología que se enfocaba en el desarrollo de una aplicación móvil en el lenguaje de programación *Swift*, el cuál es el estándar para el sistema operativo *iOS*. Asimismo, se desarrolló una herramienta que permite simular estos sensores para que futuras iniciativas puedan contribuir a este proyecto fácilmente. Durante la discusión de resultados se da a conocer que fue posible implementar la conexión entre la aplicación y los sensores, lo cuál indica que es una opción viable para mejorar la experiencia de los usuarios. Además, se indicaron las limitaciones observadas, en su mayoría causadas por la falta de cobertura de la señal de los sensores o la capacidad de ciertos teléfonos de detectar correctamente esta señal. Por lo tanto, la recomendación final es utilizar estos sensores para la aplicación de recorridos virtuales utilizando una cobertura adecuada para asegurar una experiencia ininterrumpida.

# CAPÍTULO 1

---

## Introducción

---

El proyecto de tesis propuesto tiene como finalidad el desarrollo de una interfaz gráfica para una aplicación de recorridos virtuales del Centro de Innovación y Tecnología de la Universidad del Valle de Guatemala, específicamente para el sistema operativo *iOS*. Esta iniciativa se enmarca en el contexto de un megaproyecto tecnológico que busca integrar tecnologías avanzadas como la realidad aumentada (AR) y la geolocalización para ofrecer una experiencia interactiva y educativa a los usuarios.

En la actualidad, el Centro de Innovación y Tecnología recibe una gran cantidad de visitas, tanto de nuevos estudiantes como de visitantes interesados en conocer las instalaciones y los recursos disponibles. Los recorridos guiados tradicionales, realizados por estudiantes voluntarios, han demostrado ser efectivos, pero presentan limitaciones significativas en eventos de alta demanda. La implementación de una aplicación de recorridos virtuales permitirá no solo ampliar la capacidad de atención, sino también proporcionar una experiencia personalizada y accesible en cualquier momento.

El desarrollo de la interfaz gráfica se basará en las prácticas recomendadas de desarrollo de software, asegurando la calidad mediante el control de versiones y pruebas unitarias. Se aprovecharán las capacidades tecnológicas de los dispositivos móviles *iOS*, utilizando sensores de localización interna y tecnologías de AR para crear una experiencia de usuario inmersiva. El uso de patrones de diseño de *Apple* garantizarán que la aplicación no solo cumpla con altos estándares de calidad y accesibilidad, sino que también sea intuitiva y relevante para los usuarios.

El proyecto se propone justificar la adquisición y uso de sensores de ubicación avanzada, como los *UWB* (*Ultra-Wideband*), para mejorar la precisión de la geolocalización dentro del campus. Estos sensores permitirán a los usuarios iniciar el recorrido desde cualquier ubicación dentro del campus, proporcionando indicaciones precisas y mejorando significativamente la experiencia de navegación. Además, se planea integrar la información obtenida por estos sensores con un indicador de realidad aumentada, ofreciendo indicaciones visuales a los estudiantes.

En resumen, este proyecto no solo facilitará la exploración del Centro de Innovación y Tecnología, sino que también promoverá una conexión emocional y personal con el espacio, incentivando el descubrimiento personalizado de los recursos y oportunidades que ofrece la universidad. Mediante la incorporación de tecnologías avanzadas y un diseño centrado en el usuario, se busca crear una herramienta valiosa tanto para los nuevos estudiantes como para los visitantes, contribuyendo al prestigio y la innovación educativa de la Universidad del Valle de Guatemala.

# CAPÍTULO 2

---

## Objetivos

---

### 2.1. Objetivo General

Desarrollar una interfaz gráfica para el sistema operativo iOS que permita validar el uso de sensores de geolocalización interna para su futura incorporación en el proyecto de la aplicación de recorridos virtuales UVG.

### 2.2. Objetivos Específicos

- Mostrar por medio de la interfaz gráfica información relevante al usuario sobre los puntos de interés a visitar.
- Utilizar los sensores de los celulares iPhone y la información proveída por los sensores UWB para calcular una dirección hacia el siguiente punto de interés.
- Implementar las siguientes prácticas recomendadas para el desarrollo de software: control de versiones y pruebas unitarias para asegurar el correcto funcionamiento de la aplicación y recuperación de errores.
- Desarrollar una interfaz gráfica que siga los patrones de diseño establecidos por Apple en su guía (Human Interface Guidelines) con el fin de desarrollar una interfaz intuitiva y fácil de usar.

## CAPÍTULO 3

---

### Justificación

---

La razón de ser de esta aplicación de recorridos virtuales con realidad aumentada para el Centro de Innovación y Tecnología de la Universidad del Valle de Guatemala responde al deseo de proporcionar a los nuevos estudiantes y visitantes una experiencia única y enriquecedora al explorar físicamente el campus. Actualmente, estos recorridos son impartidos por estudiantes voluntarios. Sin embargo, en eventos especiales estos voluntarios no son suficientes para cumplir con la demanda de personas que solicitan el tour. Por lo tanto, una solución virtual permitirá realizar más recorridos simultáneos; lo cual puede incrementar la cantidad de personas interesadas en un proceso de admisión en la universidad.

El valor de esta herramienta es inmenso para aquellos que se encuentran en las primeras etapas de familiarización con el campus. Al permitir que los usuarios sigan rutas y puntos de interés marcados mediante realidad aumentada, se facilita una comprensión espacial y contextual más profunda del entorno universitario. Esto no solo hace que la orientación sea más interactiva y atractiva, sino que también mejora la confianza de los nuevos miembros de la comunidad universitaria, quienes pueden sentirse abrumados por la vastedad y complejidad de un nuevo entorno académico.

Además, al requerir la presencia física de los usuarios para la interacción con la aplicación, se fomenta la exploración activa y la inmersión en el campus, promoviendo una conexión emocional y personal con el espacio. Este enfoque práctico y vivencial es fundamental para construir una sensación de pertenencia y para incentivar el descubrimiento personalizado de los recursos y oportunidades que ofrece el Centro de Innovación y Tecnología.

Un aspecto crucial de este proyecto es su capacidad para innovar en el uso de tecnología educativa. Al integrar sensores avanzados y técnicas de realidad aumentada, la aplicación no solo servirá como una herramienta de orientación, sino también como un ejemplo de la aplicación práctica de tecnologías emergentes en el ámbito educativo. Esto posicionará a la Universidad del Valle de Guatemala como una institución a la

vanguardia de la innovación tecnológica, atrayendo tanto a estudiantes interesados en estas áreas como a colaboradores y patrocinadores interesados en apoyar proyectos de alto impacto.

La necesidad de desarrollar el módulo de iOS es especialmente importante para validar el uso de sensores UWB (Ultra-Wideband), que son esenciales para mejorar la precisión de la geolocalización dentro del campus. La incorporación de estos sensores permitirá a los usuarios iniciar el recorrido desde cualquier ubicación, proporcionando indicaciones precisas y mejorando significativamente la experiencia de navegación. La integración de la tecnología UWB en el proyecto no solo demostrará la viabilidad técnica del uso de estos sensores en aplicaciones educativas, sino que también sentará las bases para futuras expansiones del proyecto, permitiendo una mayor escalabilidad y funcionalidad.

Por otra parte, este módulo es necesario para asegurar una calidad en la experiencia de usuario en la interfaz gráfica, siguiendo prácticas recomendadas por Apple (2). También permitirá comprobar la funcionalidad de los sensores y la capacidad de utilizarlos a una escala mayor.

# CAPÍTULO 4

---

Marco Teórico

---

## 4.1. Aplicaciones Móviles

En la última década, las aplicaciones móviles han transformado de manera radical la forma en que las personas interactúan con el mundo que les rodea. Con más de 7,000 millones de dispositivos móviles en uso a nivel mundial, las aplicaciones se han convertido en herramientas esenciales que facilitan la vida diaria, desde la comunicación y la educación hasta el entretenimiento y el comercio. Este crecimiento ha sido impulsado por la accesibilidad y conveniencia que ofrecen las aplicaciones, permitiendo a los usuarios realizar tareas complejas de manera sencilla y rápida desde cualquier lugar. En el contexto empresarial, las aplicaciones móviles son fundamentales para mantener la competitividad, mejorar la eficiencia operativa y ofrecer experiencias personalizadas al usuario. Además, con la creciente digitalización de servicios y el auge de tecnologías emergentes como la inteligencia artificial y el Internet de las cosas (IoT), las aplicaciones móviles continúan evolucionando, desempeñando un papel crucial en la innovación tecnológica y el desarrollo económico global.

### 4.1.1. Aplicaciones con Localización en Interiores

Las aplicaciones de localización en interiores son herramientas tecnológicas diseñadas para determinar con precisión la ubicación de personas u objetos dentro de espacios cerrados, como edificios, centros comerciales, aeropuertos y museos. A diferencia de los sistemas de posicionamiento global (GPS), que son altamente efectivos en espacios abiertos, las soluciones de indoor localization utilizan una combinación de tecnologías como señales de Bluetooth, Wi-Fi, ultra-wideband (UWB), sensores inerciales y realidad aumentada, para mapear y rastrear movimientos en entornos donde las señales de GPS son débiles o inexistentes. Estas aplicaciones son esenciales para mejorar la experiencia del usuario, permitiendo desde la navegación precisa

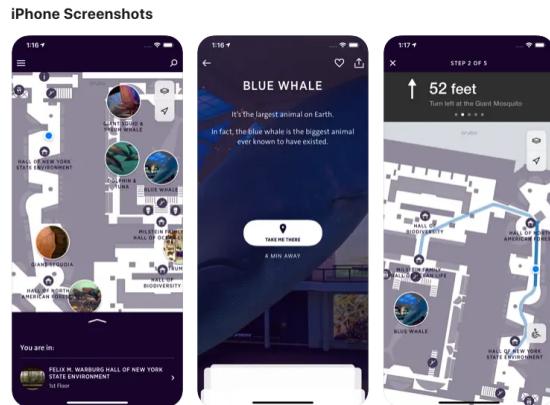


Figura 4.1: Aplicación de Navegación en el Museo Americano de Historia Natural

en espacios complejos hasta el monitoreo de activos en tiempo real. Además, su implementación está revolucionando sectores como la logística, la seguridad y la salud, ofreciendo nuevas oportunidades para la automatización, la gestión eficiente de recursos, y la personalización de servicios. Una aplicación que utiliza esta tecnología es *Explorer - AMNH NYC* que permite navegar en el Museo Americano de Historia Natural en Nueva York, Estados Unidos, como se puede observar en la Figura 4.1. Esta aplicación ayuda a los visitantes a ubicar atracciones del museo y diseñar planes de visita a este para optimizar el tiempo del viaje, esto lo han logrado colocando más de 700 *BLE Beacons* en los pasillos (3).

## 4.2. Aplicaciones móviles para el sistema operativo iOS

El sistema operativo *iOS* fue desarrollado y actualmente mantenido por la compañía de tecnología *Apple*, se lanzó en el año 2007. Hasta la fecha ha llegado a su versión 18, incluyendo en cada actualización mejoras relevantes a la experiencia del usuario. Su línea de teléfonos *iPhone* es de las más populares; en Guatemala, un 17,19 % del tráfico web surge a través de estos dispositivos. En la Universidad del Valle de Guatemala 30 % de los estudiantes utilizan estos dispositivos.

Para desarrollar aplicaciones móviles que puedan ser consumidas por este sistema operativo existen dos alternativas utilizadas comercialmente: aplicación nativa y aplicación híbrida. Una aplicación nativa es programada con los lenguajes de programación de *Apple*, estos son *Objective-C* y *Swift*. Mientras que una aplicación híbrida busca reducir el tiempo de entrega de una aplicación al utilizar un lenguaje conocido como *Javascript* y luego compilar esta aplicación tanto para *Android* como para *iOS*. Una de las dificultades de seguir el camino de las aplicaciones híbridas es el acceso a funcionalidades específicas de cada teléfono (4). En el caso de una aplicación de geolocalización, nos referimos a las antenas UWB y a los sensores Bluetooth del celular. Asimismo, una aplicación nativa, usualmente, refleja una mayor calidad porque se adecúa a las características visuales y funcionales de cada sistema operativo.

### 4.2.1. Utilizando Lenguaje Swift para desarrollo móvil

Este lenguaje fué lanzado en 2014, ha revolucionado el desarrollo de aplicaciones para *iOS*, *macOS*, *watchOS* y *tvOS*. Diseñado para ser rápido, seguro y fácil de usar, *Swift* combina la potencia y el rendimiento de un lenguaje compilado con la simplicidad y expresividad de un lenguaje moderno (5). Su sintaxis clara y concisa facilita la escritura de código, lo que reduce la posibilidad de errores y mejora la legibilidad, incluso para desarrolladores con menos experiencia. Además, *Swift* está profundamente integrado en el ecosistema de *Apple*, permitiendo a los desarrolladores aprovechar al máximo las capacidades de los dispositivos *iOS*, como la cámara, los sensores y las funciones de realidad aumentada. La adopción de *Swift* ha acelerado la innovación en el desarrollo de aplicaciones móviles, permitiendo a los desarrolladores crear aplicaciones más rápidas, seguras y ricas en funciones, lo que contribuye a mantener el ecosistema de *Apple* relevante en un mercado constantemente en evolución.

### 4.2.2. SwiftUI: Una librería para desarrollo rápido de interfaces

*SwiftUI* es un marco de desarrollado para construir interfaces de usuario de manera declarativa (6), lo que representa un cambio significativo respecto a enfoques anteriores como *UIKit*. Con *SwiftUI*, es posible definir la apariencia y el comportamiento de la interfaz gráfica en función de los estados de datos, permitiendo que la interfaz se actualice automáticamente en respuesta a los cambios. Este enfoque simplifica el proceso de desarrollo y también reduce la cantidad de código necesario, haciendo que

la creación de interfaces modernas sea más rápida y eficiente, por ejemplo en la figura 4.2 se puede observar la sencillez para definir una lista de elementos. Asimismo el código se explica por sí mismo, lo que permite un proceso de desarrollo más ágil en equipos multidisciplinarios.

Esta librería es compatible con todas las plataformas de *Apple*, como *iOS*, *macOS*, *watchOS* y *tvOS*, lo que facilita la creación de aplicaciones multiplataforma. *SwiftUI* también se integra perfectamente con el resto del ecosistema de *Apple*, aprovechando características avanzadas como el acceso a gráficos de alta calidad y una profunda integración con las APIs nativas, por ejemplo los sensores de los dispositivos.

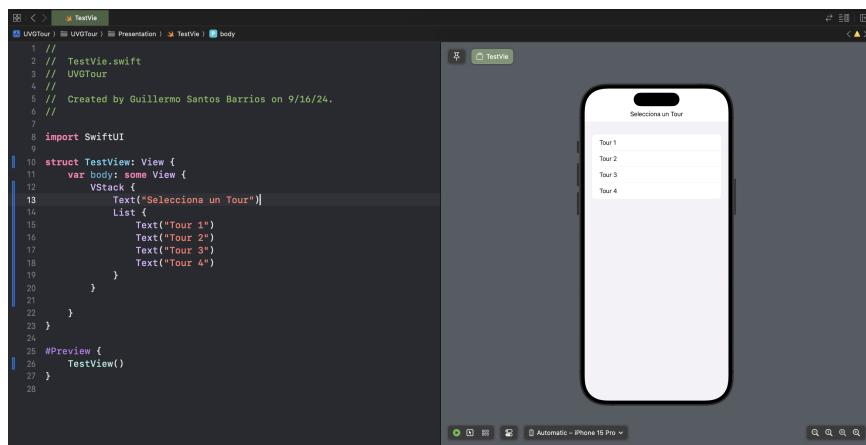


Figura 4.2: Uso básico de SwiftUI

#### 4.2.3. ARKit y RealityKit: Librerías para experiencias de Realidad Aumentada

*ARKit* es la librería de *Apple* que permite a los desarrolladores crear experiencias de realidad aumentada en dispositivos *iOS*. Esta utiliza sensores avanzados como el giroscopio y la cámara del dispositivo para detectar superficies planas, medir profundidad y posicionar objetos virtuales en el mundo real. *ARKit* combina información del mundo físico y permite superponer objetos digitales, brindando una interacción inmersiva con el entorno a través de la pantalla.

Por otro lado, *RealityKit* es una herramienta más avanzada que complementa *ARKit*, enfocándose en la creación de experiencias interactivas en 3D y en AR. *RealityKit* proporciona físicas realistas, iluminación avanzada y renderizado de alta calidad, facilitando la creación de escenas virtuales y objetos que parecen tener presencia física en el entorno real. A diferencia de *ARKit*, *RealityKit* simplifica la manipulación de objetos, colisiones y animaciones, lo que permite experiencias más envolventes (7).

Ambos marcos permiten a los usuarios interactuar con objetos virtuales en el espacio físico de forma intuitiva. *ARKit* detecta superficies y los ancla, mientras que *RealityKit* se encarga de que estos objetos se comporten de manera realista. En conjunto crean una herramienta imprescindible para aplicaciones de realidad aumentada.

en este sistema operativo.

#### 4.2.4. Accediendo a sensores del celular

*CoreLocation* es una librería de *Apple* que otorga acceso a la ubicación geográfica y datos relacionados con la orientación del dispositivo *iOS*. Aunque *CoreLocation* se utiliza principalmente para obtener la ubicación GPS del dispositivo, también permite acceder a los valores del *heading*, o dirección hacia la que apunta el dispositivo en grados. Este *heading* se mide en relación al norte magnético o verdadero, dependiendo de la configuración del dispositivo (8).

El *heading* en *CoreLocation* se puede utilizar para determinar la orientación del dispositivo en el espacio, lo que es útil en aplicaciones de realidad aumentada (AR) y navegación. Al acceder a la propiedad *CLHeading*, es posible obtener el ángulo en grados que indica hacia dónde está orientado el teléfono. Esto permite crear aplicaciones que ajusten su interfaz o contenido en función de la orientación física del dispositivo, proporcionando una experiencia más interactiva y precisa para el usuario. El uso de esta librería es sencillo gracias a la capa de abstracción que ofrece. El código necesario se puede observar en la figura 4.3

```

5  // Created by Guillermo Santos Barrios on 8/31/24.
6  //
7
8 import Foundation
9 import Corelocation
10 import Combine
11
12 class LocationManager: NSObject, ObservableObject, CLLocationManagerDelegate {
13     let manager = CLLocationManager()
14     @Published var degrees: Double = 0
15
16     override init() {
17         super.init()
18         manager.delegate = self
19         manager.startUpdatingHeading()
20         manager.requestWhenInUseAuthorization()
21     }
22
23     // MARK: Conforming to protocol
24
25     func locationManager(_ manager: CLLocationManager, didUpdateHeading newHeading: CLHeading) {
26         degrees = newHeading.trueHeading
27     }
28
29 }
30

```

Figura 4.3: Uso de CoreLocation para obtener la dirección del teléfono

## 4.3. Buenas prácticas en el desarrollo de aplicaciones móviles

Desarrollar una aplicación móvil que pueda ser mantenida por mucho tiempo y diferentes equipos requiere el seguimiento de ciertas prácticas recomendadas por profesionales. A continuación se discute de estas herramientas y cuál es su impacto en una solución móvil.

### 4.3.1. Arquitectura de la aplicación

Las arquitecturas en una aplicación móvil definen la estructura y organización del código, lo que tiene un impacto directo en la mantenibilidad, escalabilidad y calidad del proyecto a largo plazo. Una arquitectura bien elegida permite una separación clara de responsabilidades entre los componentes de la aplicación, facilitando el desarrollo, las pruebas y la capacidad de realizar cambios sin afectar otras partes del sistema. Por ejemplo, arquitecturas como *MVVM* o *VIPER* ayudan a modularizar el código, lo que es crucial cuando la aplicación crece en complejidad.

Seleccionar la arquitectura correcta desde el inicio del proyecto es vital, ya que una mala elección puede llevar a un código difícil de mantener, con dependencias fuertes entre componentes, lo que complica la introducción de nuevas funcionalidades o la corrección de errores. A medida que una aplicación evoluciona, una arquitectura bien estructurada permite a los desarrolladores agregar nuevas características de manera más eficiente, mejorar la experiencia del usuario y asegurar que la aplicación siga siendo robusta y fácil de gestionar a lo largo del tiempo.

#### Arquitecturas más comunes en aplicaciones móviles

En el desarrollo de aplicaciones móviles, la elección de la arquitectura juega un papel fundamental en cómo se estructura y mantiene el código a lo largo del tiempo. Tres de las arquitecturas más utilizadas en el desarrollo de aplicaciones iOS son *Model-View-Controller (MVC)*, *Model-View-ViewModel (MVVM)*, y *VIPER*. Cada una de estas arquitecturas tiene sus propias ventajas y desafíos, y la elección entre ellas depende en gran medida de las necesidades específicas del proyecto y del equipo de desarrollo, segúin descritos por Alex Anderson en su artículo “*Choosing a Design Pattern for your SwiftUI App*” (9).

El patrón **MVC** es uno de los más tradicionales y se enfoca en dividir la aplicación en tres componentes principales: el Modelo, que maneja los datos y la lógica de negocio; la Vista, que es responsable de la interfaz de usuario; y el Controlador, que actúa como un intermediario entre el Modelo y la Vista. Este patrón permite realizar prototipos rápidamente, sin embargo no es recomendado para aplicaciones a gran escala ya que la lógica de negocio está entrelazada con la interfaz gráfica.

El patrón **MVVM** introduce un nuevo componente llamado ViewModel, que se

encarga de manejar la lógica de presentación y de preparar los datos para ser mostrados en la Vista. Esto permite una mejor separación de responsabilidades y facilita el testeo unitario de la lógica de la aplicación. MVVM es especialmente útil en aplicaciones donde se necesita una interfaz de usuario altamente reactiva, ya que se integra bien con *SwiftUI*.

Finalmente, **VIPER** es una arquitectura más modular y formalizada, que divide la aplicación en cinco componentes: *View*, *Interactor*, *Presenter*, *Entity*, y *Router*. **VIPER** promueve una alta separación de responsabilidades y es ideal para aplicaciones complejas y grandes, donde la modularidad y la posibilidad de escalar son cruciales. Sin embargo, esta técnica tiene mejores resultados al ser utilizada con la librería *UIKit*, ya que se basa en patrones más tradicionales.

#### 4.3.2. Organización del código

Organizar el código correctamente permite identificar fácilmente los distintos archivos que componen a la aplicación. El error más común es separar los archivos en 3 grupos: *Views*, *Models*, *Controllers*. Con el paso del tiempo esta separación causa un repositorio resistente al cambio (10). Otras fuentes recomiendan seguir un patrón similar al establecido en el libro *Clean Architecture* (11), acoplándose a la arquitectura *MVVM* en *SwiftUI*. Básicamente, el objetivo es mantener una separación adecuada entre los componentes que generan vistas y los componentes encargados de manejar los datos y el estado de la aplicación (12). Los componentes principales a tomar en cuenta son:

1. Presentación: Esta capa contiene todos los elementos visuales. Separados en vistas re-utilizables y pantallas independientes. El objetivo es que esta capa sea lo más posible portátil para poder reciclar el código en otros proyectos de ser necesario.
2. Aplicación: Mantiene archivos que sean útiles para toda la aplicación. Por ejemplo, en el proyecto hay un archivo *AppState* que contiene las variables necesarias para saber si un usuario ha escogido un tour.
3. Datos: Maneja el acceso a repositorios de información, ya sean remotos o locales. Por ejemplo, obtener información de una *API Web* o de una base de datos.
4. Dominio: Contiene los archivos que manejan aspectos lógicos de la aplicación. Es importante que esta capa esté desacoplada de cualquier librería.

#### 4.3.3. Diseño

El diseño de aplicaciones móviles requiere una consideración profunda de las dimensiones de usabilidad, ya que influyen directamente en la experiencia del usuario. Aspectos como la efectividad, eficiencia y satisfacción son fundamentales para garantizar que las aplicaciones cumplan con las expectativas de los usuarios, permitiendo

que estas se utilicen de manera intuitiva y sin fricciones. Según Baharuddin et al. (13), estos elementos no solo mejoran la funcionalidad de la aplicación, sino que también contribuyen a su aceptación en una amplia gama de contextos.

Las *Human Interface Guidelines* (HIG) de *Apple* juegan un papel crucial en el diseño de aplicaciones *iOS* (2), proporcionando un conjunto de recomendaciones que buscan garantizar la coherencia y la usabilidad. Estas guías, cuando se implementan correctamente, logran que las aplicaciones sean atractivas y fáciles de usar. Sin embargo, como señalan Baharuddin et al. (13), las guías genéricas para la usabilidad pueden no cubrir todas las peculiaridades de las aplicaciones móviles, lo que hace necesario adaptar las recomendaciones a los dispositivos y entornos específicos para mejorar la efectividad y satisfacción del usuario.

En resumen, la consideración de dimensiones clave de usabilidad como la simplicidad, la estética y la facilidad de aprendizaje es esencial para el éxito de cualquier aplicación móvil. Estas dimensiones permiten a los diseñadores enfocarse en los aspectos que más influyen en la experiencia del usuario, asegurando que las aplicaciones no solo sean útiles, sino también agradables.

#### 4.3.4. Accesibilidad

La accesibilidad en el diseño de aplicaciones móviles es crucial para asegurar que todos los usuarios, independientemente de sus capacidades, puedan interactuar con la aplicación de manera efectiva. Uno de los principales enfoques es ofrecer opciones multilingües dentro de la aplicación, adaptando la aplicación al idioma preferido del usuario. Esto extiende el alcance global de la aplicación.

Otra consideración importante es el soporte para los modos claro y oscuro. Estos ajustes permiten a los usuarios seleccionar el esquema de color que mejor se adapte a sus preferencias o necesidades, ya sea por comodidad visual o por razones de salud, como la reducción de la fatiga ocular en entornos de baja luz (14). Implementar estos modos no solo mejora la experiencia de usuario, sino que también demuestra una atención cuidadosa a las necesidades de accesibilidad visual.

Además, ofrecer la posibilidad de ajustar el tamaño del texto dentro de la aplicación es fundamental. Al permitir diferentes tamaños de texto, las aplicaciones pueden ser más inclusivas para personas con discapacidades visuales o simplemente usuarios que prefieren un texto más grande o más pequeño. Este tipo de personalización, que está alineada con las *Human Interface Guidelines* de *Apple*, asegura que la aplicación sea utilizable por la mayor cantidad de personas posible, sin comprometer la calidad del diseño o la experiencia de usuario.

#### 4.3.5. Control de versiones utilizando GIT

GIT es un sistema de control de versiones distribuido que permite a los desarrolladores gestionar y registrar los cambios en el código fuente de un proyecto de manera eficiente. A diferencia de los sistemas de control de versiones centralizados, GIT almacena una copia completa del historial del proyecto en cada máquina de los colaboradores, lo que permite trabajar de manera independiente y sin conexión al servidor principal. Esta arquitectura distribuida no solo mejora la velocidad de las operaciones como *commits*, *merges* y *checkouts*, sino que también proporciona mayor resiliencia, ya que no se depende de un único punto de falla (15, Section 1.3).

El uso de GIT facilita la colaboración entre desarrolladores, permitiendo que varios miembros de un equipo trabajen en paralelo en diferentes partes de un proyecto. Los cambios realizados por cada desarrollador se integran mediante un proceso de *merge* que GIT maneja de manera eficiente, resolviendo conflictos de manera automática en la mayoría de los casos. Además, GIT soporta ramas (*branches*), que permiten trabajar en nuevas características o corregir errores sin afectar la estabilidad del código principal. Al final del proceso, los cambios pueden ser fusionados con la rama principal, asegurando un flujo de desarrollo ordenado y coherente (15, Section 3.1).

#### Utilizando GIT en proyectos grandes

*Git Flow* es una estrategia de ramificación en *Git* que organiza el desarrollo de software en distintas ramas para gestionar de manera efectiva el ciclo de vida de un proyecto grande. En *Git Flow*, la rama principal (*main* o *master*) se utiliza para el código que está desplegado, mientras que la rama *develop* actúa como la base para el desarrollo activo. A partir de *develop*, se crean ramas específicas para cada nueva característica (*feature/feature-name*), correcciones de bugs (*hotfix/*) o preparaciones para lanzamientos (*release/*). Esta estrategia permite a los equipos trabajar en nuevas funcionalidades sin afectar la estabilidad del código principal y facilita la integración de cambios de manera estructurada y ordenada. Al finalizar el desarrollo de una característica, la rama correspondiente se fusiona de nuevo en *develop*, y eventualmente, las ramas *develop* y *release* se fusionan en *main* para marcar un nuevo lanzamiento de la aplicación. *Git Flow*, por lo tanto, proporciona un flujo de trabajo robusto que ayuda a mantener un código limpio y organizado a lo largo de todo el proceso de desarrollo (16).

#### 4.3.6. Pruebas Unitarias

Las pruebas unitarias son una herramienta que asegura el funcionamiento correcto de un proyecto. Como Khorikov dice en su libro acerca del tema “La meta de las pruebas unitarias es asegurar el crecimiento sostenible de un proyecto”(1, p. 5). Un crecimiento sostenible permite que el proyecto siempre tenga la mejor calidad posible. Usualmente los equipos deciden evitar el uso de pruebas unitarias ya que consumen

más tiempo a diferencia de únicamente desarrollar el proyecto (17). Sin embargo, llega un momento en el que el proyecto alcanza un nivel de complejidad suficientemente alto y el desarrollo de nuevas funcionalidades sin utilizar pruebas resulta ser aún más lento que al utilizarlas. De manera visual, es posible imaginar el esfuerzo requerido para realizar nuevas en un proyecto al observar la figura 4.4.

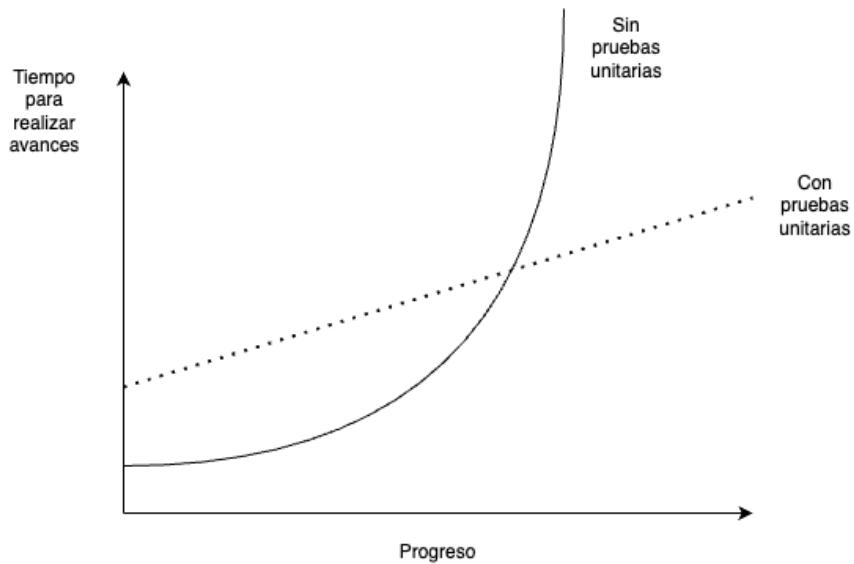


Figura 4.4: Esfuerzo Requerido Para Nuevas Funcionalidades (1)

### Anatomía de las pruebas unitarias

Las pruebas unitarias deben ser consideradas como código importante (17). Por lo tanto, es necesario plantear una anatomía que permita su entendimiento claro. En este proyecto se decidió utilizar la técnica *Arrange, Act, Assert*. En donde cada parte tiene la siguiente responsabilidad:

1. *Arrange*: Preparar todo lo necesario para la prueba unitaria. Esto involucra configurar el sujeto de prueba, el cuál es la entidad específicamente que se desea probar. Así como otras variables necesarias para que la prueba funcione correctamente.
2. *Act*: En esta etapa se simula el comportamiento del usuario al ejecutar el método que se desea probar del sujeto de prueba. Es buena práctica que esta fase sea breve.
3. *Assert*: Se verifica que el comportamiento sea el esperado.

A continuación se muestra un ejemplo de esta técnica utilizada en este proyecto en la figura 4.5. El sujeto de prueba en este caso es una clase *Tour*, la cual contiene las paradas que el usuario puede visitar. En la fase de *Arrange*, se genera una instancia de este tour, utilizando información local. Posteriormente se marca cada parada como completada. Y por último se verifica que el booleano *completed* sea verdadero en el

sujeto de prueba. Esto asegura que la funcionalidad de completar un tour tenga un riesgo menor de ser alterada por algún error causado de manera no intencional.

```
◊      func testTour_whenAllStopsCompleted_isCompleted() {  
23          // Arrange  
24          var tour = LocalToursDatasource.tours[0]  
25          // Act  
26          for _ in tour.stops {  
27              tour.completeStop()  
28          }  
29          // Assert  
30          XCTAssertTrue(tour.completed)  
31      }  
32
```

Figura 4.5: Ejemplo de una prueba unitaria

## 4.4. Sensores UWB

*Ultra-Wideband* (UWB) es una tecnología de comunicación inalámbrica que opera en un rango de frecuencias amplio, permitiendo la transmisión de datos a través de pulsos de muy baja energía y corta duración. A diferencia de otras tecnologías de comunicación como *Bluetooth* o *Wi-Fi*, *UWB* no depende de la transmisión continua de ondas sino de ráfagas rápidas de energía en un espectro ancho, lo que le permite alcanzar una precisión de localización en el rango de centímetros. Esto es posible gracias a la medición exacta del tiempo de vuelo de las señales entre el transmisor y el receptor, lo que convierte a *UWB* en una solución ideal para aplicaciones que requieren una localización precisa en tiempo real, como la navegación en interiores, la realidad aumentada, y el seguimiento de objetos y personas en entornos complejos.

### 4.4.1. Compatibilidad de los Sensores UWB con celulares existentes

La tecnología *UWB* ha comenzado a integrarse en dispositivos móviles modernos, especialmente en *smartphones* de alta gama, lo que abre nuevas posibilidades para aplicaciones que requieren una localización precisa y segura. Empresas como *Apple* y *Samsung* han liderado la incorporación de *UWB* en sus dispositivos, con el chip U1 de *Apple* presente en modelos como el *iPhone 11* en adelante, y en dispositivos como el *Apple Watch* y el *HomePod mini*. Esta integración permite que los sensores *UWB* en teléfonos móviles se comuniquen directamente con otros dispositivos habilitados para *UWB*, lo que facilita aplicaciones como el seguimiento preciso de objetos, la autenticación basada en proximidad, y la transferencia rápida de archivos entre dispositivos. La compatibilidad nativa con *UWB* en estos *smartphones* no solo mejora las capacidades de localización en interiores, sino que también permite nuevas formas de interacción en el ecosistema de dispositivos inteligentes.

A pesar de las ventajas que ofrece la compatibilidad de *UWB* en dispositivos móviles, todavía existen desafíos relacionados con la adopción masiva de esta tecnología. Actualmente, la compatibilidad con *UWB* está limitada a un número relativamente pequeño de dispositivos de alta gama, lo que restringe su aplicación a usuarios que poseen estos modelos específicos. Además, la falta de un estándar unificado en la implementación de *UWB* entre diferentes fabricantes puede causar problemas de interoperabilidad. Sin embargo, mientras más aplicaciones sean desarrolladas con esta tecnología, se llegará a un estándar que logre cubrir un mayor rango de dispositivos móviles.

### 4.4.2. Estimote Proveedor de sensores UWB

*Estimote* es una empresa de tecnología dedicada a proveer soluciones de *hardware* y *software* para distintos casos de uso, entre ellos, localización interna (18). Para esto tienen a su disposición sensores *UWB* que pueden ser adquiridos a través de su sitio web. En la siguiente figura se muestra la apariencia de estos sensores 4.6.

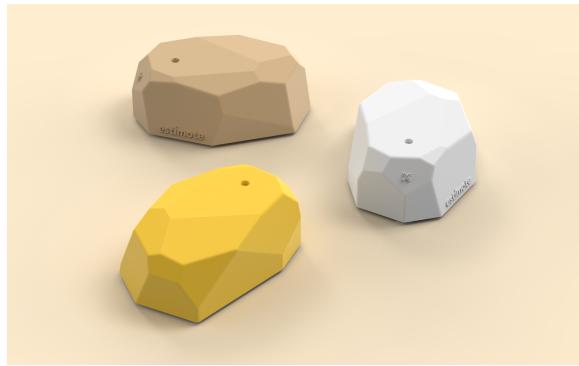


Figura 4.6: Sensores UWB de Estimote

El costo de un kit de desarrollo con 3 sensores es de USD 99.00. Cada sensor tiene un identificador único y es detectable a través de cualquier celular que soporte esta tecnología. Asimismo, ofrecen una librería de desarrollo para integrar fácilmente sus sensores tanto para el sistema operativo *iOS*, como para *Android*.

La librería de *iOS* de *Estimte* (19), tiene las instrucciones necesarias para capturar la distancia exacta desde el celular hasta el sensor hasta con una exactitud de 10 cm. Esta librería utiliza las antenas proveídas por el celular para conectarse a los sensores disponibles, una vez la conexión se ha inicializado es posible acceder a las actualizaciones de la ubicación del celular con respecto a cada sensor conectado (19). En la siguiente figura se visualiza como el celular detecta estos sensores 4.7.

```
Discovered device: 31787018dfdd4c2fb241f563081da8d2 rssi: -71
Discovered device: b288b83142c74bf0ab28178d93532e32 rssi: -40
```

Figura 4.7: Registro de conexión con los sensores

#### 4.4.3. Sensores BLE

Los sensores *BLE* (*Bluetooth Low Energy*) son dispositivos de comunicación inalámbrica que utilizan la tecnología *Bluetooth Low Energy* para transmitir datos a baja potencia (20). Esta tecnología es ideal para aplicaciones que requieren una transmisión de datos eficiente y de bajo consumo energético, como las aplicaciones de monitoreo de proximidad y localización en interiores. A diferencia de la tecnología *Bluetooth* clásica, BLE está optimizado para mantener una conexión constante mientras consume significativamente menos energía (20). Los sensores BLE pueden emitir señales conocidas como *beacons*, que permiten a otros dispositivos, como celulares, detectar y medir su proximidad. Sin embargo, una limitante es que esta tecnología no permite al celular calcular directamente un valor de distancia, únicamente provee la intensidad de la señal.

#### 4.4.4. WebSocket

*WebSocket* es un protocolo de comunicación que permite establecer una conexión bidireccional en tiempo real entre un cliente y un servidor a través de un único canal. A diferencia de otros protocolos como *HTTP*, que siguen un modelo de solicitud-respuesta, *WebSocket* permite que tanto el cliente como el servidor envíen datos de forma independiente y continua sin la necesidad de abrir nuevas conexiones (21). Esta característica lo hace ideal para aplicaciones que requieren actualizaciones en tiempo. En el caso de esta investigación, puede ser aplicado para el desarrollo de un simulador de los sensores.

# CAPÍTULO 5

---

## Metodología

---

1. Obtención de información sobre los recorridos a mostrar: Esta información será proveída por el módulo de Marketing; esta es una lista de recorridos, donde cada recorrido tiene un nombre, descripción, puntos de interés e imágenes a mostrar. Asimismo, se realizará una encuesta donde se busque encontrar una muestra significativa que permita dar a conocer la proporción de estudiantes que utilizan dispositivos con sistema operativo *iOS*.
2. Investigación sobre patrones de arquitectura para aplicaciones móviles: La investigación sobre patrones de arquitectura es esencial para encontrar la estructura más adecuada que soporte el desarrollo y mantenimiento eficiente de la aplicación. Entre las opciones más comunes y recomendadas para aplicaciones *iOS* se encuentran *MVVM* (*Model-View-ViewModel*), *MVC* (*Model-View-Controller*), *VIPER* (*View-Interactor-Presenter-Entity-Router*) y *MVP* (*Model-View-Presenter*). Cada uno de estos patrones tiene sus propias ventajas y desventajas que deben ser cuidadosamente evaluadas en el contexto específico de este proyecto. *MVVM*, por ejemplo, es conocido por facilitar el enlace de datos bidireccional, lo que puede simplificar la gestión de estados en la interfaz de usuario. *MVC*, por otro lado, es un patrón más tradicional que separa claramente las responsabilidades, pero puede llevar a controladores muy grandes en aplicaciones complejas (22). Para llevar a cabo esta investigación, se realizará una revisión exhaustiva de la literatura y documentación técnica disponible sobre estos patrones. Esto incluirá el análisis de artículos académicos, blogs técnicos y documentación oficial de *Apple* y otras fuentes relevantes. Además, se revisarán casos de estudio y ejemplos prácticos de aplicaciones que hayan implementado estos patrones. Esta revisión permitirá entender cómo cada patrón ha sido aplicado en la práctica, sus impactos en el rendimiento de la aplicación, y las mejores prácticas para su implementación. Se prestará especial atención a cómo estos patrones manejan la separación de preocupaciones, la reutilización de código y la escalabilidad (23).
3. Desarrollo de diseño de baja fidelidad en una herramienta gráfica: Un diseño

de baja fidelidad permitirá visualizar la distribución de componentes gráficos y asegurar que cumplan con la calidad deseada (24).

4. Desarrollo de interfaz para el caso de uso principal: Antes de trabajar con información proveniente de sensores y enviar señales al módulo de realidad aumentada, es necesario tener una visualización del comportamiento que tendrá la interfaz gráfica. Esto permitirá detectar los momentos específicos en los que el módulo debe comunicarse con otros para obtener información o invocar alguna acción. El lenguaje de programación para el desarrollo de la interfaz gráfica es *Swift* y la librería a utilizar es *SwiftUI*. Este es el nuevo estándar recomendado por Apple desde 2021 (25).
5. Desarrollo de simulación de sensores: Para asegurar cierta independencia del presente módulo y poder realizar pruebas sin necesidad de estar físicamente en los puntos donde se instalarán los sensores, es necesario tener un tipo de simulación. La alternativa seleccionada es la implementación de un servidor *Web Socket*. Este servidor se realizará con la librería *SocketIO*, utilizando *Node.js* para el servidor, lo que permitirá una comunicación eficiente y en tiempo real entre el servidor y los clientes. Esta configuración facilita la simulación de datos de sensores y su integración con la aplicación iOS de manera ágil y escalable.

La herramienta debe tener una interfaz simple para poder modificar los siguientes parámetros: ubicación del sensor, identificador del sensor e intensidad de la señal. Para ello, se desarrollará un *frontend* utilizando *React*, una biblioteca de *JavaScript* que permite crear interfaces de usuario interactivas y dinámicas (26). La aplicación de *React* proporcionará una interfaz de usuario intuitiva y fácil de usar, donde los desarrolladores podrán ajustar los parámetros de los sensores simulados en tiempo real, lo que permitirá realizar pruebas exhaustivas y ajustar la aplicación según sea necesario sin depender de la instalación física de los sensores (27).

Esta simulación no solo permitirá realizar pruebas de funcionalidad y rendimiento, sino que también ayudará a identificar y solucionar posibles problemas antes de la implementación en un entorno real. Además, facilitará la colaboración entre diferentes miembros del equipo de desarrollo, ya que podrán acceder al servidor de simulación y realizar pruebas desde distintas ubicaciones. Este enfoque metodológico garantiza una mayor flexibilidad y eficiencia en el desarrollo y prueba de la aplicación, asegurando que se cumplan todos los requisitos técnicos y funcionales del proyecto.

6. Integrar simulación de sensores con la interfaz gráfica: Una vez el servidor de simulación esté listo, la aplicación implementará su contrato para permitir el desarrollo de pruebas. Esta integración se diseñará de manera suficientemente flexible para que, en el futuro, se pueda reemplazar fácilmente la simulación con los sensores reales. Esto significa que la arquitectura del software será modular, permitiendo que los componentes que interactúan con los sensores puedan cambiarse sin necesidad de realizar modificaciones significativas en la lógica de la aplicación. La idea es que el código que actualmente se comunica con el servidor de simulación sea capaz de conectarse y trabajar con los sensores *UWB* reales mediante una simple re-configuración o sustitución de módulos.

Esta flexibilidad se logrará mediante el uso de interfaces y patrones de diseño adecuados, como la inyección de dependencias, que permitirán cambiar entre diferentes implementaciones de sensores sin alterar el comportamiento general de la aplicación. Por ejemplo, se puede definir una interfaz para la gestión de datos de sensores y tener dos implementaciones diferentes: una para la simulación y otra para los sensores reales. Durante el desarrollo y pruebas iniciales, la implementación de simulación será utilizada. Posteriormente, para las pruebas finales y la implementación en el entorno real, la aplicación podrá utilizar la implementación que maneja los sensores *UWB* reales. La estrategia permitirá hacer una transición rápida hacia los sensores físicos y hacer distintas pruebas para simular ambientes reales. Además, facilita la identificación y solución de problemas específicos relacionados con la integración de sensores, mejorando la robustez y fiabilidad de la aplicación antes de su despliegue final. Al adoptar este enfoque, el desarrollo de la aplicación se vuelve más ágil y adaptable, capaz de incorporar nuevas tecnologías y mejoras futuras con un esfuerzo mínimo.

7. Integrar indicador de realidad aumentada a la interfaz gráfica: La interfaz gráfica debe de interpretar la información obtenida por los sensores e indicar a la librería de realidad aumentada *iOS* instrucciones para renderizar indicaciones a los usuarios.
8. Integrar sensores *UWB* con la interfaz gráfica: Una vez comprobado el correcto funcionamiento de la interfaz, se reemplazará el componente simulado de sensores por una comunicación real con el protocolo *UWB*.
9. Completar desarrollo de casos de uso: Los requerimientos del proyecto incluyen el desarrollo de varios casos de uso adicionales para asegurar que la aplicación cumpla con las expectativas y necesidades de todos los usuarios. Entre estos casos de uso se encuentran la implementación de medidas de accesibilidad y opciones de localización.

Para las medidas de accesibilidad, se pondrá especial énfasis en proporcionar una experiencia de usuario inclusiva. Esto incluirá la implementación de una variación de colores entre modos claro y oscuro, lo cual no solo ayuda a los usuarios con diferentes preferencias de visualización, sino que también es crucial para aquellos con sensibilidad a la luz o problemas de visión. Además, se ofrecerán opciones para ajustar el tamaño del texto, permitiendo a los usuarios personalizar la interfaz según sus necesidades de legibilidad. Estas medidas aseguran que la aplicación sea accesible para un público amplio, incluidos aquellos con discapacidades visuales.

En cuanto a la localización, la aplicación ofrecerá soporte tanto en inglés como en español, los dos idiomas más hablados por la comunidad universitaria y los visitantes del campus. Esto se logrará mediante la implementación de archivos de recursos multilingües y la utilización de herramientas de internacionalización proporcionadas por *iOS*. Los usuarios podrán seleccionar su idioma preferido desde el menú de configuración de la aplicación, lo que garantizará que todos los textos, indicaciones y mensajes de la interfaz se muestren en el idioma seleccionado. Este enfoque no solo mejora la accesibilidad, sino que también hace que la aplicación sea más atractiva para un público más diverso (28).

## CAPÍTULO 6

---

### Alcance

---

Este proyecto busca crear una visión sobre la posibilidad de implementar sensores de localización interna en el campus de la Universidad. Este es un recurso que puede ayudar a las personas a familiarizarse con las instalaciones. Además, se puede utilizar esta herramienta para eventos especiales, por ejemplo la *Experiencia UVG*, para guiar a los usuarios a través de distintos puntos de interés.

Los recursos necesarios para desarrollar este módulo son: Sensores *UWB*, una computadora con capacidad de generar aplicaciones móviles para el sistema operativo *iOS*, el entorno de desarrollo integrado *XCode* y un teléfono iPhone para realizar pruebas. La Universidad ha brindado tres sensores de la marca *Estimote* (18), y la computadora ya se ha adquirido. Además, si la universidad desea publicar esta aplicación en la tienda virtual *App Store*, debe de adquirir una licencia de desarrollador, la cuál tiene un costo de 100.00 dólares anuales.

Se estima que el tiempo asignado para el proyecto será suficiente para cumplir con los objetivos establecidos, tomando en cuenta el desarrollo de la aplicación y del simulador de posiciones. Esto se espera lograr siguiendo un flujo de trabajo ágil, en donde cada iteración será probada y validada para una adaptación rápida a los cambios.

Sin embargo, este proyecto no se reduce a navegación a través de puntos definidos; si se tienen suficientes sensores será posible combinar distintas tecnologías como *BLE* y *UWB* para mostrar un mapa de la Universidad y saber exactamente donde se encuentran los usuarios. Extendiendo el uso de esta infraestructura para que los estudiantes puedan encontrar salones de clases, conferencias, ofertas en puestos de comida, etc. El alcance de la aplicación se ve limitada por la cantidad de sensores adquiridos para el desarrollo; por lo tanto se ha considerado el uso de un simulador para hacer el desarrollo más intuitivo y permitir que la aplicación pueda ser desarrollada sin sensores físicos en todo momento, dado que al momento de realizar este proyecto solo habían tres sensores. Asimismo, es posible que la aplicación tenga interferencias para conectarse a los sensores debido a factores externos como: orientación del celular y obstáculos.

# CAPÍTULO 7

## Resultados y Discusión

### 7.1. Prototipo de baja fidelidad

Antes de realizar el desarrollo de las funcionalidades, se decidió crear un prototipo de baja fidelidad que partiera de los requerimientos solicitados. En la siguiente figura se puede ver cómo se plantea un concepto inicial:

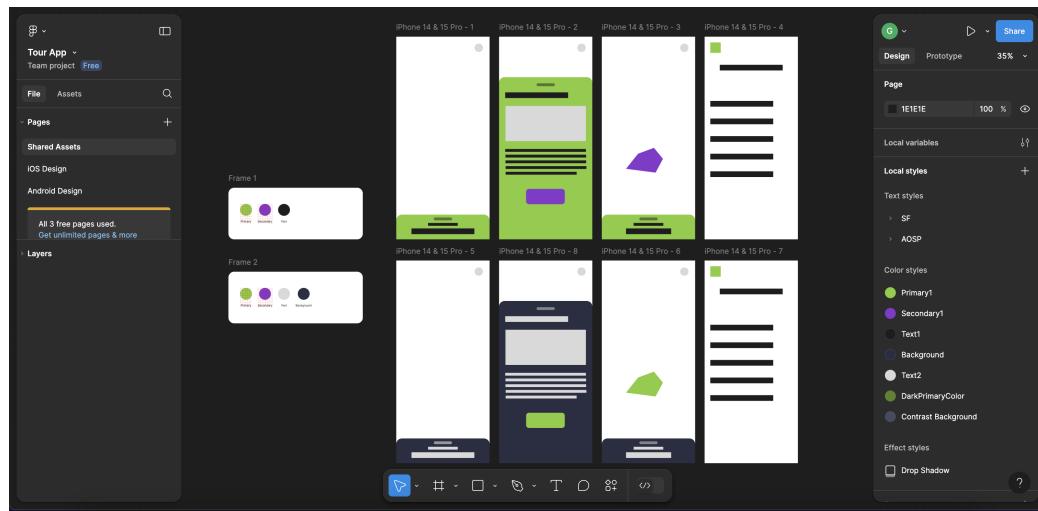


Figura 7.1: Prototipo de baja fidelidad con figuras

Posteriormente, se refina el prototipo para el sistema operativo iOS y así tener una idea clara de la estética de la aplicación:

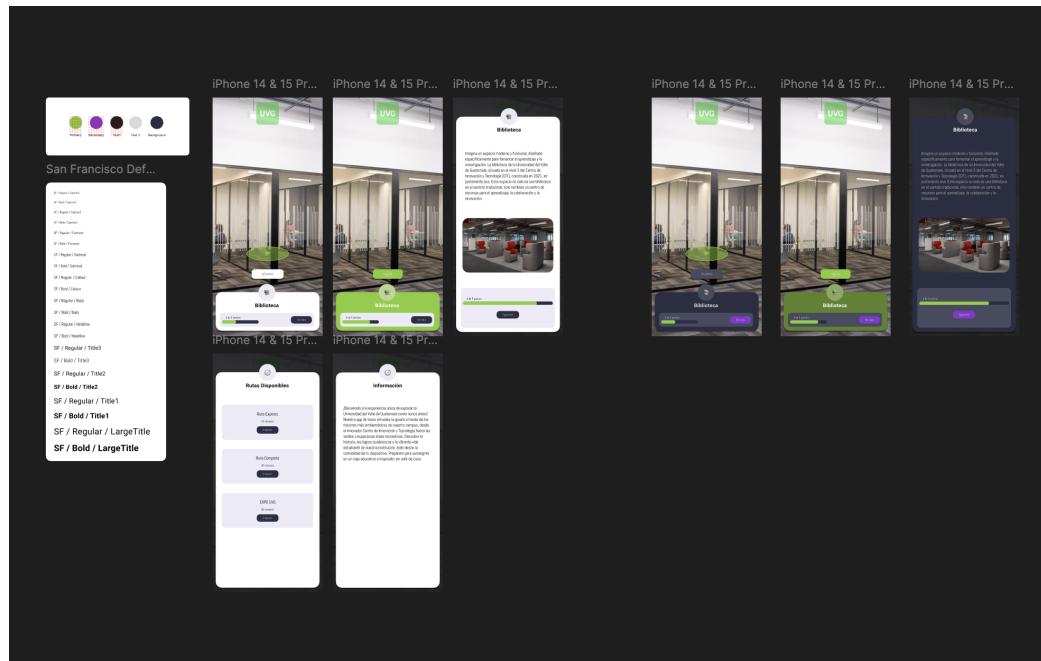


Figura 7.2: Diseño preliminar de la aplicación para iOS

## 7.2. Flujo de la aplicación

Una ventaja del proyecto es que el flujo de un tour es lineal, ya que las rutas están diseñadas previamente por el personal de la Universidad. Por lo tanto el diagrama de flujo para esta funcionalidad se ve así:

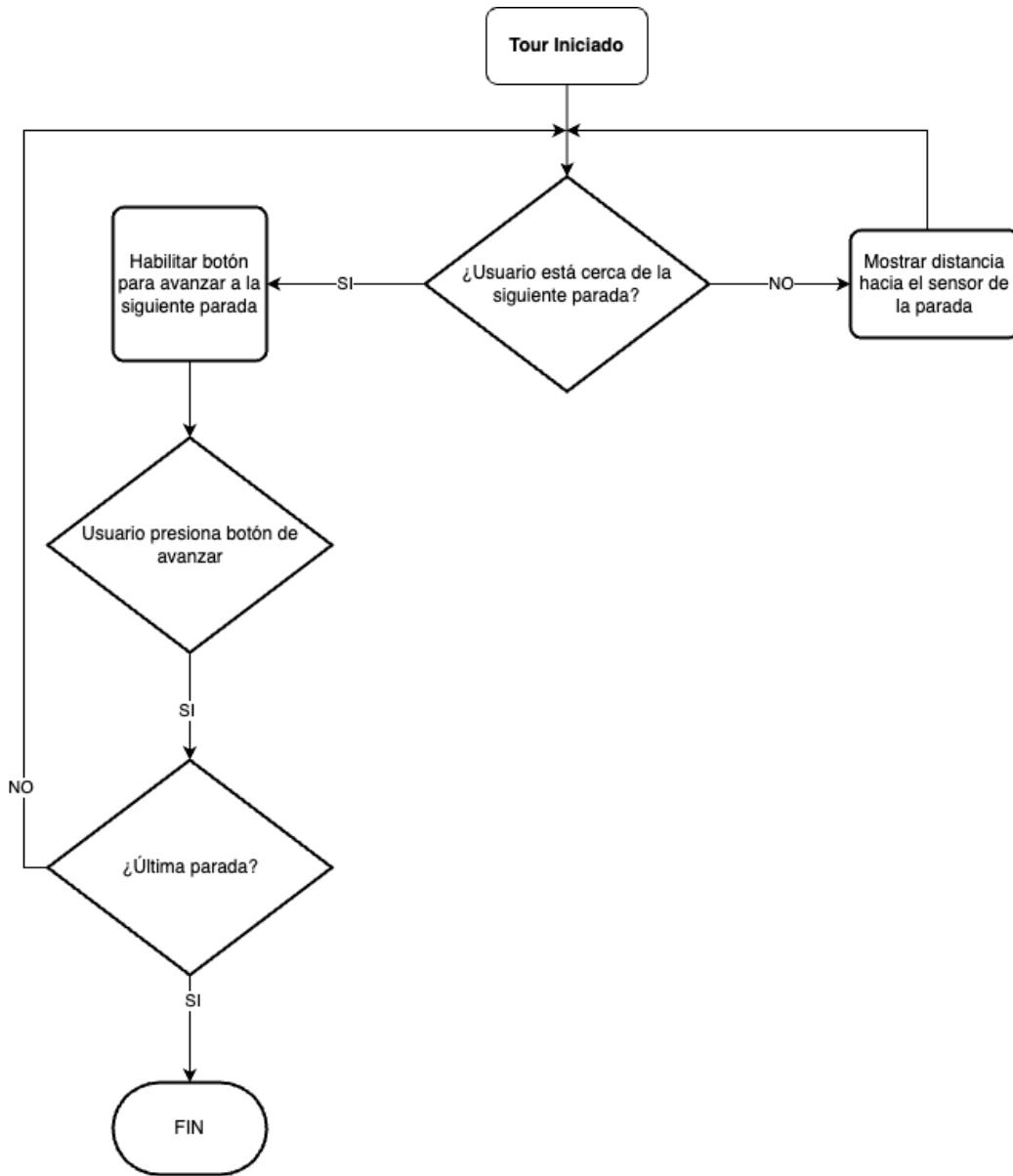


Figura 7.3: Diagrama de flujo - Funcionalidad Tour

### 7.3. Desarrollo de simulador de sensores

Dado que los tres sensores que se adquirieron de *Estimote* debían ser compartidos con otros módulos, se decidió desarrollar un simulador de sensores que utilizara el protocolo *UDP*. Esto permitiría un desarrollo más rápido de la aplicación y probar nuevos conceptos sin tener físicamente los sensores en ese momento. Asimismo, este simulador puede ayudar al futuro de este proyecto como una herramienta para validar ideas antes de iniciar su desarrollo. El desarrollo involucró dos proyectos: un cliente y un servidor que hiciera *broadcast* de los datos proporcionados. El lenguaje escogido fue *TypeScript* debido a la facilidad de uso y a la familiarización que ya se tenía con este.

#### Cliente

El cliente se desarrolló en la librería *React*. Los requerimientos funcionales para este proyecto fueron los siguientes:

1. Permitir creación de un sensor.
2. Permitir eliminar sensores creados
3. Asignar un identificador a cada sensor.
4. Cambiar la distancia detectada del sensor. Esto es necesario para simular que un usuario se está acercando o alejando.

La siguiente figura muestra la interfaz gráfica del cliente 7.4.

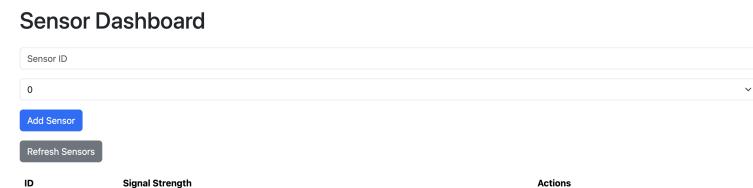


Figura 7.4: Interfaz gráfica inicial simulador de sensores

El formulario en la parte superior permite al usuario ingresar el identificador y la distancia en metros. Posteriormente al presionar el botón *Add Sensor*, el cliente envía una señal al servidor y se refresca la página para mostrar el nuevo sensor como se muestra en la siguiente figura 7.5.

**Sensor Dashboard**

| ID            | Signal Strength | Actions                 |
|---------------|-----------------|-------------------------|
| 12345-ABCDEFG | 5               | <button>Remove</button> |

Figura 7.5: Interfaz gráfica con un sensor

Para editar la distancia el usuario puede seleccionar el elemento en la columna *Signal Strength*, esto se puede verificar a continuación 7.6.

**Sensor Dashboard**

| ID            | Signal Strength  | Actions                 |
|---------------|--|-------------------------|
| 12345-ABCDEFG | <input type="text" value="0"/><br><input checked="" type="text" value="5"/><br><input type="text" value="10"/> | <button>Remove</button> |

Figura 7.6: Editando un sensor

Con este procedimiento es posible crear los sensores que el usuario desee. Permitiendo simular cualquier escenario para la aplicación. A continuación se muestra la interfaz con más sensores creados.

**Sensor Dashboard**

| ID                    | Signal Strength | Actions                 |
|-----------------------|-----------------|-------------------------|
| 12345-ABCDEFG         | 5               | <button>Remove</button> |
| 12333-432432-55435534 | 0               | <button>Remove</button> |
| 123213-432432         | 10              | <button>Remove</button> |

Figura 7.7: Interfaz gráfica con varios sensores

### Servidor

El servidor tiene dos responsabilidades: permitir a la interfaz web administrar sensores (discutido previamente) y enviar actualizaciones en intervalos de tiempo constante de los sensores para que la aplicación pueda interpretarlos y así actualizar su perspectiva de los sensores. Para programar este servidor utilizando la tecnología de *Websockets* se delegó la responsabilidad de inicializar el servidor web a la librería *Express* de *Javascript*. En la siguiente imagen se observan registros de depuración en donde se indica que el cliente se ha conectado, creado sensores y actualizado la distancia de algunos 7.8.

```

Client connected
received: {"type":"request-sensors"}
received: {"type":"update-sensor","body":{"id":"1","signalStrength":"5"}}
received: {"type":"remove-sensor","body":{"id":"1"}}
received: {"type":"add-sensor","body":{"id":"12324-12312312","signalStrength":"5"}}
added sensor Sensor { id: '12324-12312312', signalStrength: '5' }
received: {"type":"remove-sensor","body":{"id":"12324-12312312"}}
received: {"type":"add-sensor","body":{"id":"12345-ABCDEFG","signalStrength":"10"}}
added sensor Sensor { id: '12345-ABCDEFG', signalStrength: '10' }
received: {"type":"remove-sensor","body":{"id":"12345-ABCDEFG"}}
received: {"type":"add-sensor","body":{"id":"12345-ABCDEFG","signalStrength":"5"}}
added sensor Sensor { id: '12345-ABCDEFG', signalStrength: '5' }
received: {"type":"add-sensor","body":{"id":"12333-432432-55435534","signalStrength":"10"}}
added sensor Sensor { id: '12333-432432-55435534', signalStrength: '10' }
received: {"type":"add-sensor","body":{"id":"123213-432432","signalStrength":"5"}}
added sensor Sensor { id: '123213-432432', signalStrength: '5' }
received: {"type":"update-sensor","body":{"id":"12333-432432-55435534","signalStrength":"0"}}
received: {"type":"update-sensor","body":{"id":"123213-432432","signalStrength":"10"}}

```

Figura 7.8: Depuración del servidor

En cuanto a las actualizaciones constantes se decidió un intervalo de *2500 ms* para simular la latencia que pudiera existir entre los sensores reales y el dispositivo. A continuación se muestran los mensajes de depuración indicando que se está actualizando a los clientes conectados 7.9.

```

Broadcasting sensors data: [
  Sensor { id: '1', signalStrength: '0' },
  Sensor { id: '12323', signalStrength: '5' }
]
Broadcasting sensors data: [
  Sensor { id: '1', signalStrength: '0' },
  Sensor { id: '12323', signalStrength: '5' }
]
Broadcasting sensors data: [
  Sensor { id: '1', signalStrength: '0' },
  Sensor { id: '12323', signalStrength: '5' }
]
Broadcasting sensors data: [
  Sensor { id: '1', signalStrength: '0' },
  Sensor { id: '12323', signalStrength: '5' }
]
received: {"type":"update-sensor","body":{"id":"12323","signalStrength":"10"}}
Broadcasting sensors data: [
  Sensor { id: '1', signalStrength: '0' },
  Sensor { id: '12323', signalStrength: '10' }
]
received: {"type":"update-sensor","body":{"id":"1","signalStrength":"5"}}
Broadcasting sensors data: [
  Sensor { id: '1', signalStrength: '5' },
  Sensor { id: '12323', signalStrength: '10' }
]

```

Figura 7.9: Servidor actualizando a los clientes

Con estos dos elementos, fue posible desarrollar funcionalidades de la aplicación

incluso sin un sensor físico. Esta interacción se puede visualizar con el siguiente diagrama 7.10.

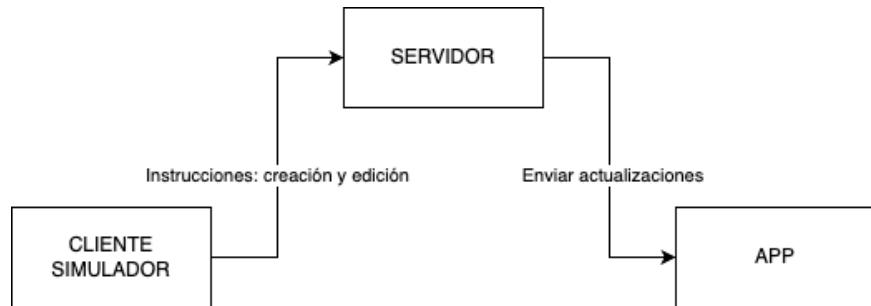


Figura 7.10: Diagrama de interacción Simulador - App

## 7.4. Selección de un patrón de diseño para la aplicación y organización del código

La arquitectura seleccionada para este proyecto fue *MVVM (Model- View- ViewModel)* combinada con conceptos de *Clean Architecture* para organizar el código eficientemente. Esta combinación fue elegida porque permite una clara separación de responsabilidades, asegurando que la lógica de negocio y la manipulación de datos se mantengan desacopladas de la interfaz gráfica. Gracias a esta estructura, el proyecto se desarrolló con una base sólida y escalable, lo que facilita futuras expansiones y la incorporación de nuevas funcionalidades. Además, esta arquitectura optimiza la implementación de pruebas unitarias, dado que la lógica principal se encuentra en componentes independientes, lo cual contribuye a mantener un alto nivel de calidad en el desarrollo y simplifica la detección y corrección de errores.

A continuación se muestra una estructura de archivos acoplándose a estas directivas.

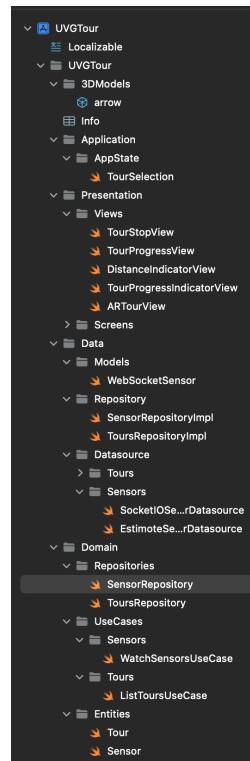


Figura 7.11: Organización del proyecto

## 7.5. Desarrollo de la aplicación utilizando simulador

Una vez el simulador de sensores ya estuvo listo, el principal avance era lograr conectar a la aplicación con este. Se decidió empezar con esto antes de cualquier otro elemento gráfico para validar el concepto y hacer ajustes de ser necesario. Por lo tanto, se empezó con una pantalla en blanco:

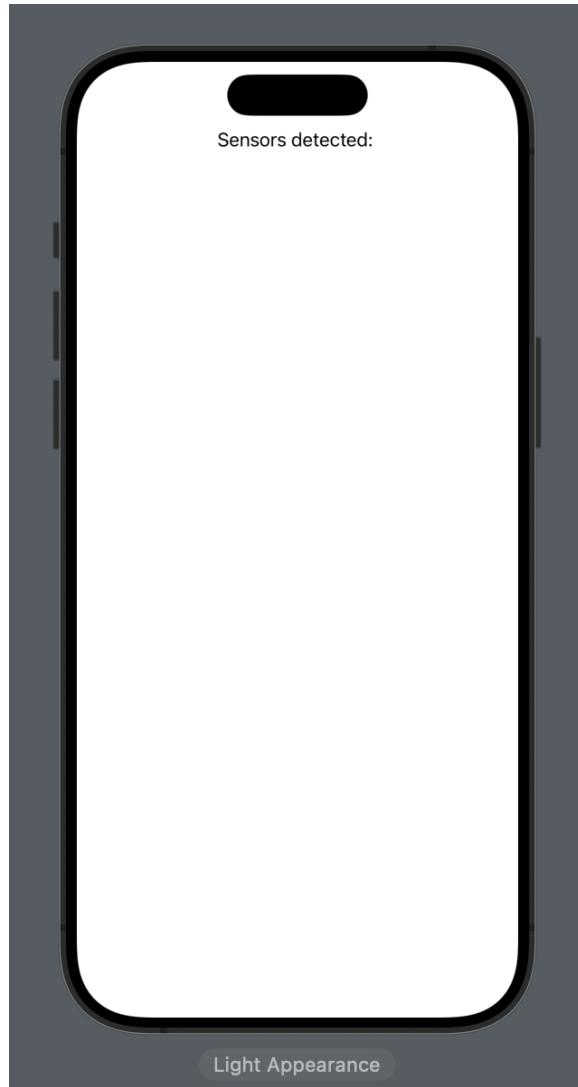


Figura 7.12: Pantalla de prueba para los sensores

Estando lista esta interfaz gráfica, se procedió a implementar la lógica para que la aplicación pudiese escuchar las actualizaciones proveídas por el simulador. Otra meta importante era implementar esta funcionalidad de cierta forma que desde el punto de vista de la aplicación móvil no existiera diferencia entre un sensor físico y un simulador. Por lo tanto, para esto se utilizó un contrato para definir las características que debe de tener el código que escuche los sensores; este se puede observar en la siguiente figura 7.13.

```

1 // 
2 //  SensorRepository.swift
3 //  UVGTour
4 //
5 //  Created by Guillermo Santos Barrios on 8/17/24.
6 //
7
8 import Foundation
9 import Combine
10
11
12 /// Repository protocol that will handle interaction with sensors.
13 /**
14 /// This repository will:
15 /// - Expose a ``sensorsPublisher`` that the client can listen to retrieve sensor updates.
16 /**
17 protocol SensorRepository {
18     var sensorsPublisher: AnyPublisher<[Sensor], Never> { get }
19 }
```

Figura 7.13: Contrato de implementación para los sensores

Como se puede observar, este contrato solo expone una propiedad llamada *sensorsPublisher*, la cuál permite ser escuchada por la interfaz gráfica para recibir nuevos valores. Además, vale la pena resaltar que el tipo del valor que se va a devolver en esta variable es un arreglo de la estructura interna llamada *Sensor*. La cuál tiene la siguiente definición:

```

1 // 
2 //  Sensor.swift
3 //  UVGTour
4 //
5 //  Created by Guillermo Santos Barrios on 8/17/24.
6 //
7
8 import Foundation
9
10
11 /// Data representation of a real sensor.
12 /**
13 /// Each sensor has a unique identifier ``id``.
14 /// The ``distance`` is calculated in meters from the user's phone.
15 struct Sensor: Identifiable, Equatable {
16     var id: String
17     var distance: Float
18 }
```

Figura 7.14: Definición de estructura de un sensor

Como se refleja en la figura 7.14, esta estructura tiene un atributo indicando el identificador, y otro que indica la distancia en metros del sensor al teléfono del usuario.

Una vez hecho el código para conectarse al servidor Websocket utilizando el contrato establecido previamente (En el archivo *SocketIOSensorDatasource.swift* en el

repositorio de la aplicación), fue posible terminar la implementación y mostrar en tiempo real las actualizaciones que se realizaban mediante el simulador:

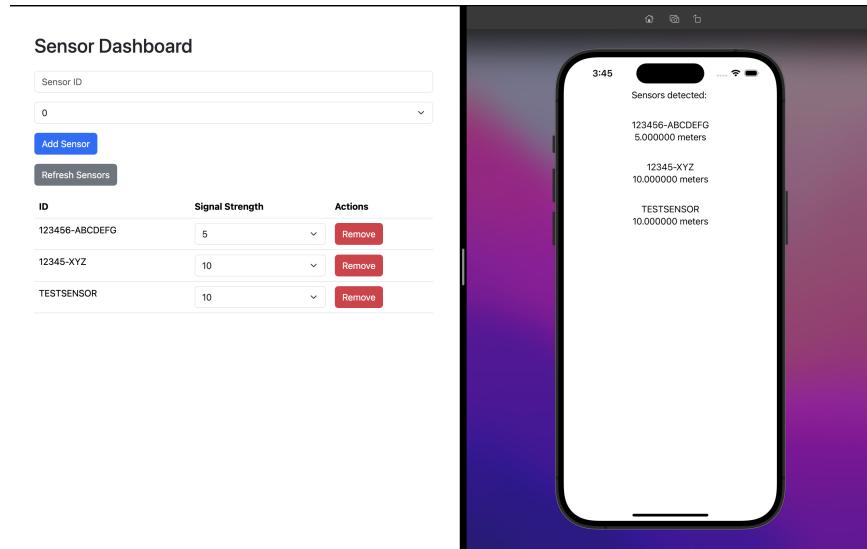


Figura 7.15: Aplicación recibe información del simulador

## 7.6. Implementación de componentes principales de la interfaz gráfica

La simplicidad de la interfaz gráfica es importante dado que se desea la interacción de los usuarios con los espacios de la Universidad. Por lo tanto, los componentes visuales deben agregar a la experiencia y no distraer. Por lo tanto los componentes principales que se involucran en el tour son:

1. Vista de progreso: Es necesario indicar al usuario un resumen de su progreso en el tour. La información importante a mostrar es el nombre de la parada, la cantidad de puntos que ha visitado y cuántos le faltan. Asimismo esta vista debe habilitar un botón para navegar hacia la siguiente una vez el usuario se acerca lo suficiente al destino. A continuación se puede ver el diseño principal de esta vista.



Figura 7.16: Vista de progreso



Figura 7.17: Vista de progreso - Un punto ha sido visitado

2. Detalles de una parada: Es indispensable mostrar una pequeña descripción de la parada así como una imagen que ayude al usuario a encontrar el espacio deseado y a entender su significado dentro del campus. El comportamiento esperado es que se pueda presionar en la vista de progreso y estos detalles se mostrarán:



Figura 7.18: Detalles de una parada

3. Indicador de distancia: Un elemento sutil indica al usuario su distancia en metros al sensor principal de la siguiente parada. Una vez el usuario ha llegado, el elemento se lo hace saber.



Figura 7.19: Distintos estados del indicador de distancia

## 7.7. Implementando funcionalidad de Tours

La funcionalidad de *tours* permite que los usuarios naveguen a través de ciertos puntos definidos. Como se describió anteriormente en el flujo de la aplicación: un Tour consiste de una o más paradas. Cada parada está asociada a un sensor y contiene información a mostrar sobre ese punto en particular. Esta parte del proyecto consistió en programar la lógica que permite la navegación en un tour. En esta iteración de la aplicación, las rutas fueron representadas como una estructura lineal (un arreglo). Sin embargo, se puede extender esta estructura a un grafo no dirigido con el fin de hacer la aplicación flexible a cambios inesperados. Para poner a prueba esta funcionalidad, se utilizó información ficticia, esto para agilizar el desarrollo. Esta información puede verse en el archivo *LocalToursDatasource.swift*. En la siguiente figura se puede observar al simulador y la aplicación compartiendo información sobre el tour. En el simulador se colocaron los identificadores que la aplicación espera para cada sensor. En este caso, se está simulando que el usuario está a 5 metros de la primera parada:

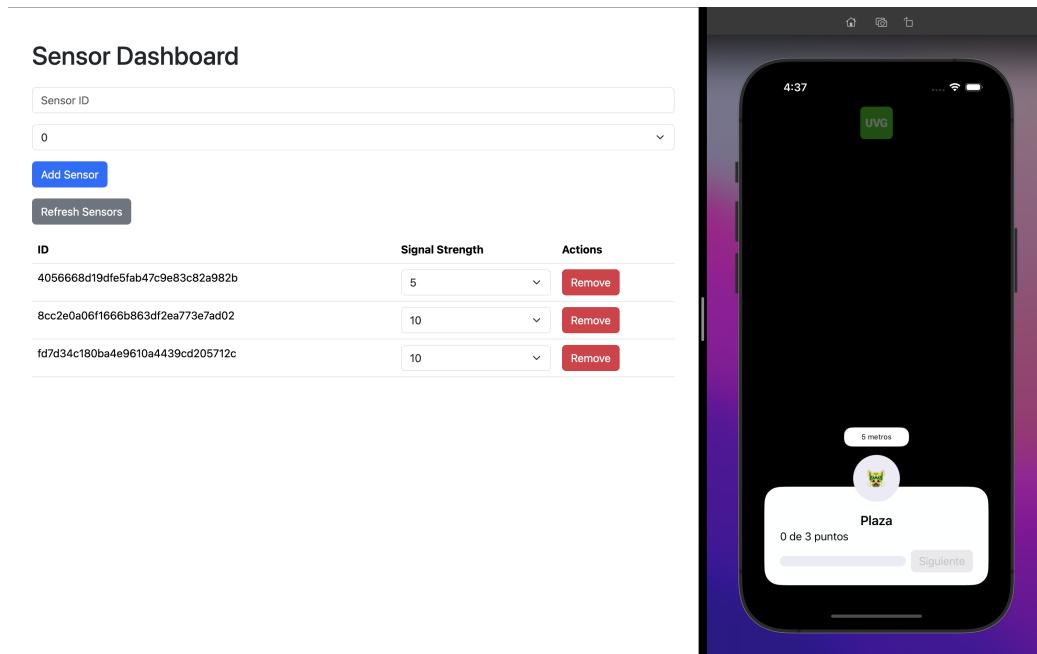


Figura 7.20: Se muestra la distancia hacia la siguiente parada

Posteriormente se edita el valor en el simulador a 0 para indicar que el usuario se ha acercado al sensor.

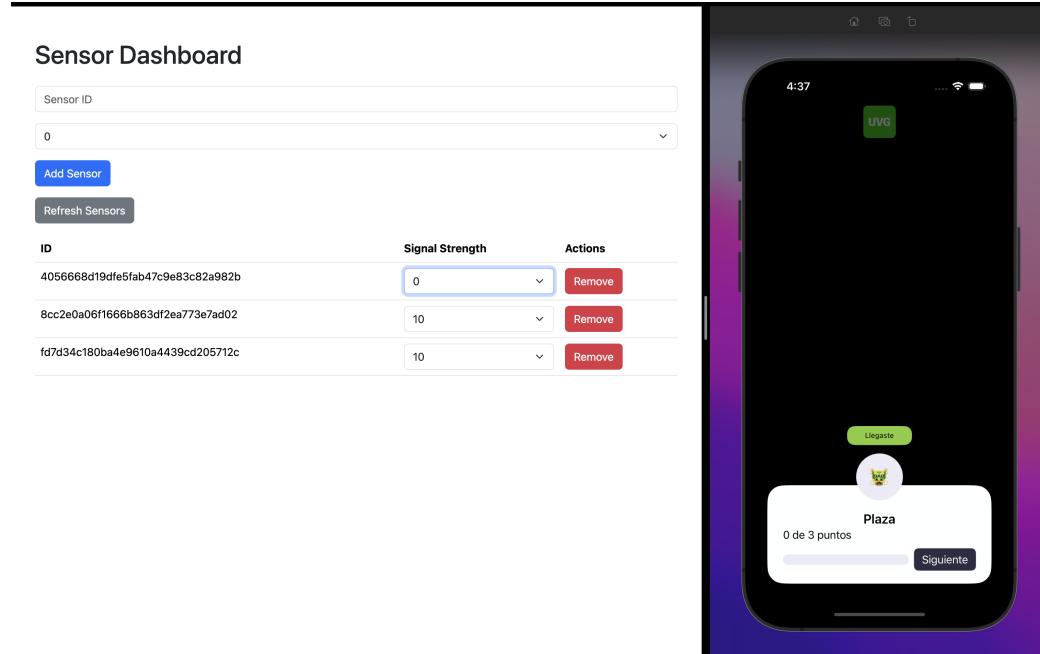


Figura 7.21: El usuario ha llegado a la primera parada

Cabe resaltar que el botón que permite al usuario avanzar a la siguiente parada se habilita hasta que el usuario se haya acercado al menos una vez al sensor, es decir, el usuario ha visitado el punto de interés.

Posteriormente el usuario puede observar la distancia hacia la siguiente parada.

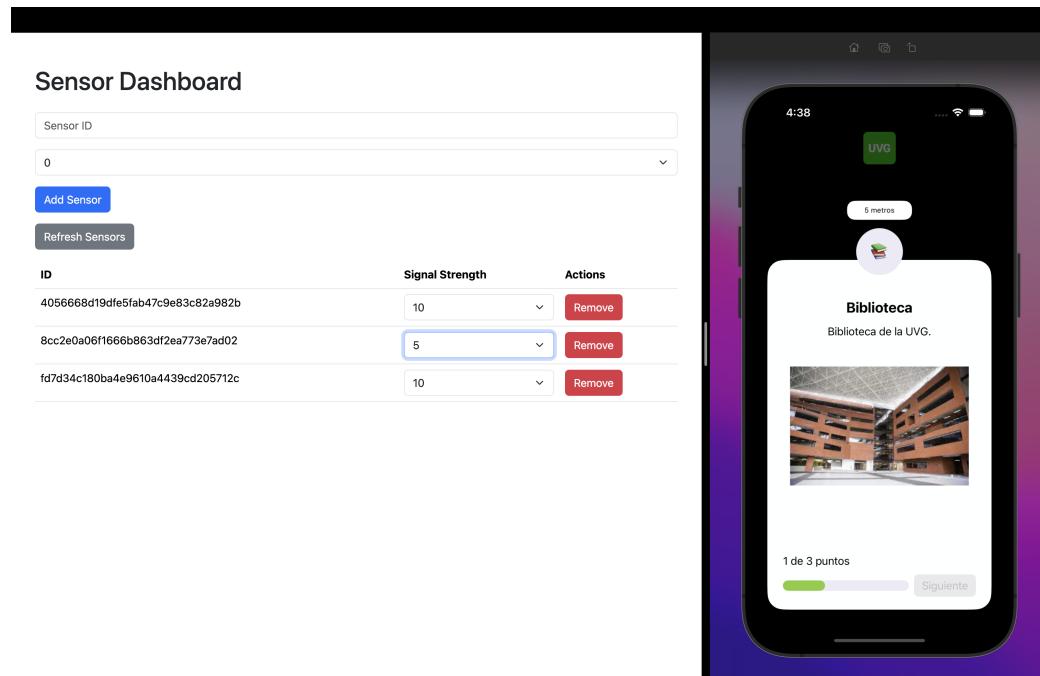


Figura 7.22: Se muestra una distancia hacia la siguiente parada

Este es el ciclo principal de un Tour. También se puede apreciar cómo los elementos mencionados en la sección anterior se unen para crear una interfaz gráfica simple y concisa que reacciona a los datos proveídos por los sensores o el simulador. Por último, cuando un usuario finaliza el tour se ve de la siguiente manera:

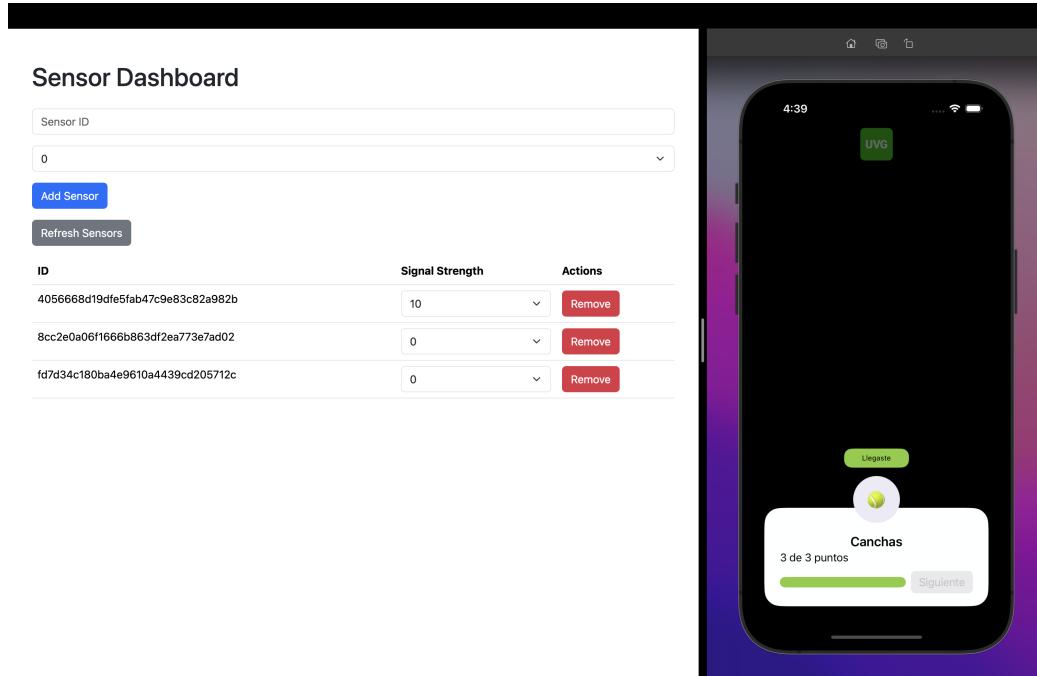


Figura 7.23: Tour finalizado

## 7.8. Utilizando sensores UWB para reemplazar el simulador

Dado que el funcionamiento correcto de los tours fue validado, se prosiguió con la integración de los sensores en la aplicación. Para esto se utilizó la librería proveída por *Estimote* (19). El código de la integración puede observarse en el archivo *EstimoteSensorDatasource.swift*.

Para probar esta funcionalidad se utilizó un sensor en la Biblioteca del CIT. Esto se puede observar en la siguiente figura:

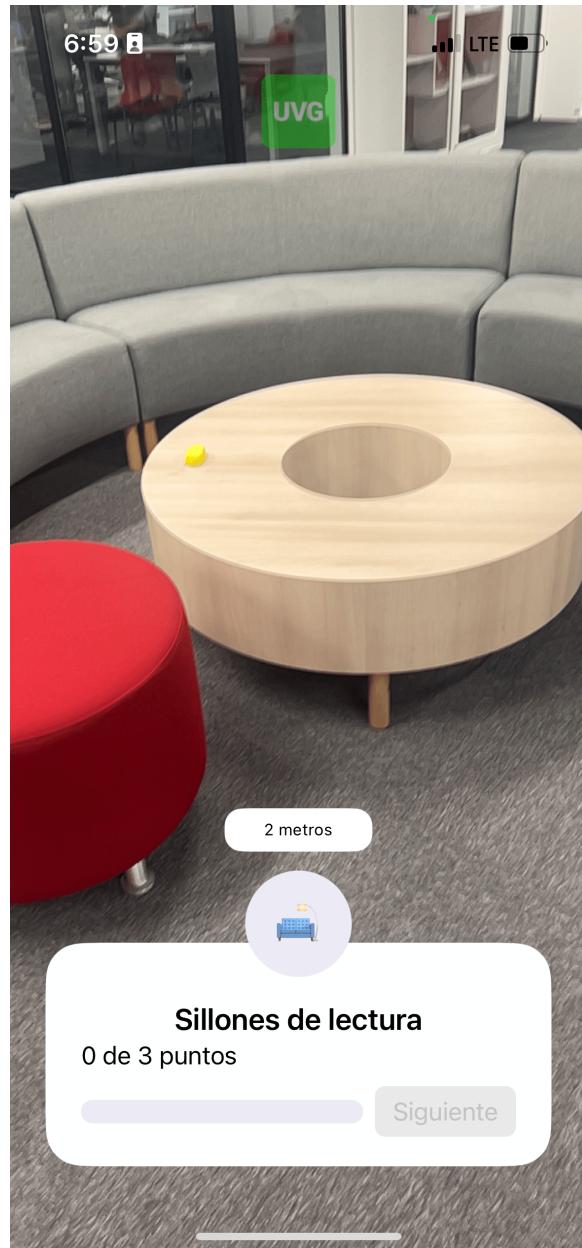


Figura 7.24: Sensor en Biblioteca - a 2 metros

Asimismo, se corroboró la funcionalidad de abrir los detalles de la parada:

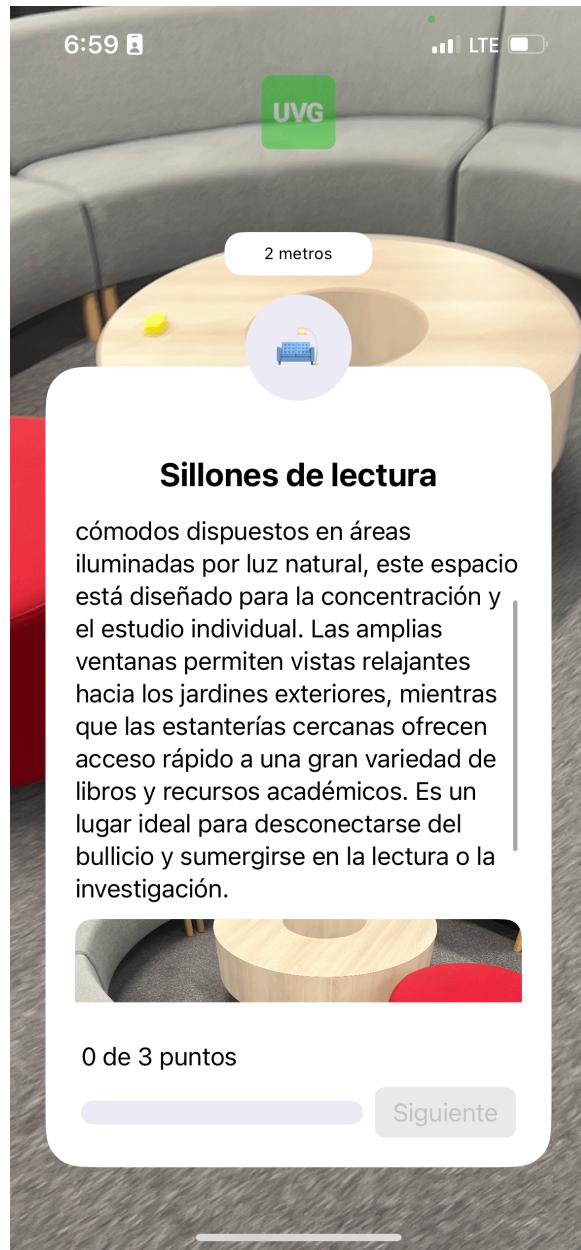


Figura 7.25: Detalles de la primera parada

Ahora, cuando el usuario se acerca lo suficiente al sensor el resultado esperado es un indicador que haga saber que se ha llegado a la ubicación. Asimismo, se agregó una retroalimentación con los motores del celular:

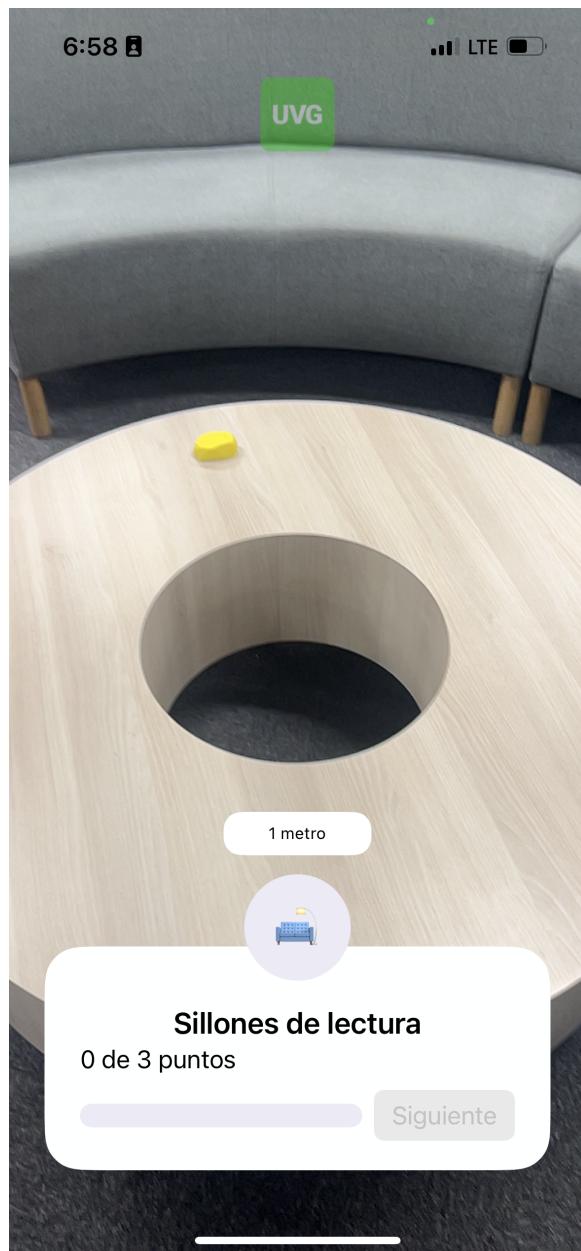


Figura 7.26: Aplicación detecta que está cerca del sensor

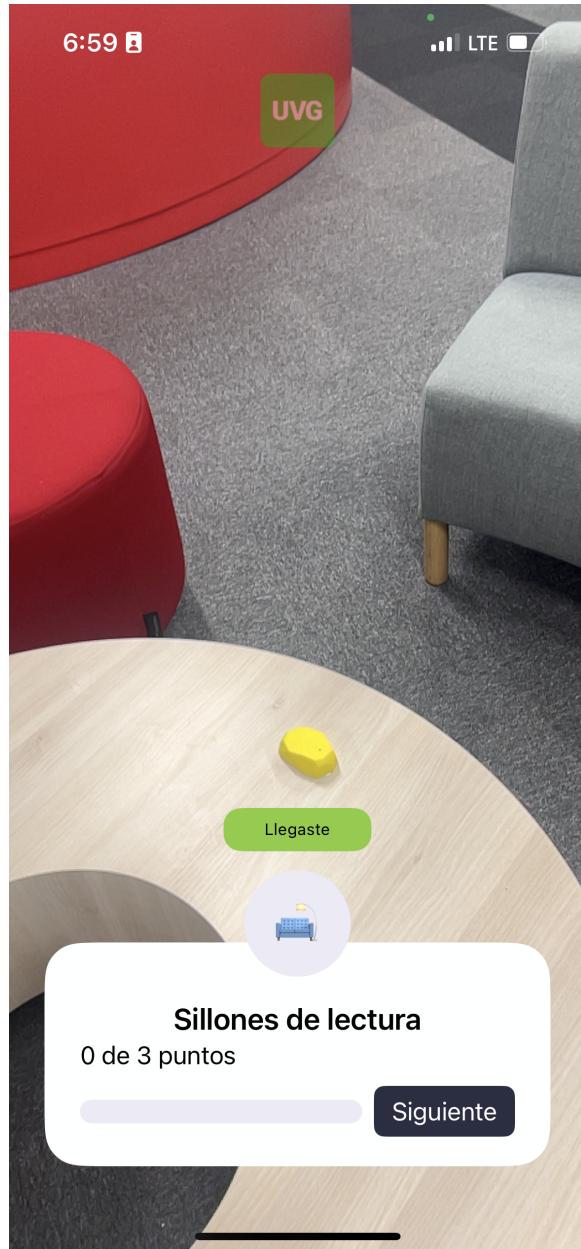


Figura 7.27: Aplicación detecta que se ha llegado al sensor

## 7.9. Componente de realidad aumentada

La realidad aumentada es una herramienta que permitirá fortalecer la experiencia del usuario con la aplicación. La idea es mostrar un indicador que muestre la dirección hacia la siguiente parada.

Debido a la limitante del número de sensores, no se siguió con un enfoque de triangulación. Más bien se está aprovechando el magnetómetro, para saber la dirección a la que el usuario está apuntando. La librería *CoreLocation* (8) permite acceder a este valor como un número de 0 a 360 grados con respecto al Norte. Por lo tanto, en la estructura de datos cada parada tiene un campo que indica la dirección hacia el siguiente sensor. Esto se puede observar en la siguiente figura:

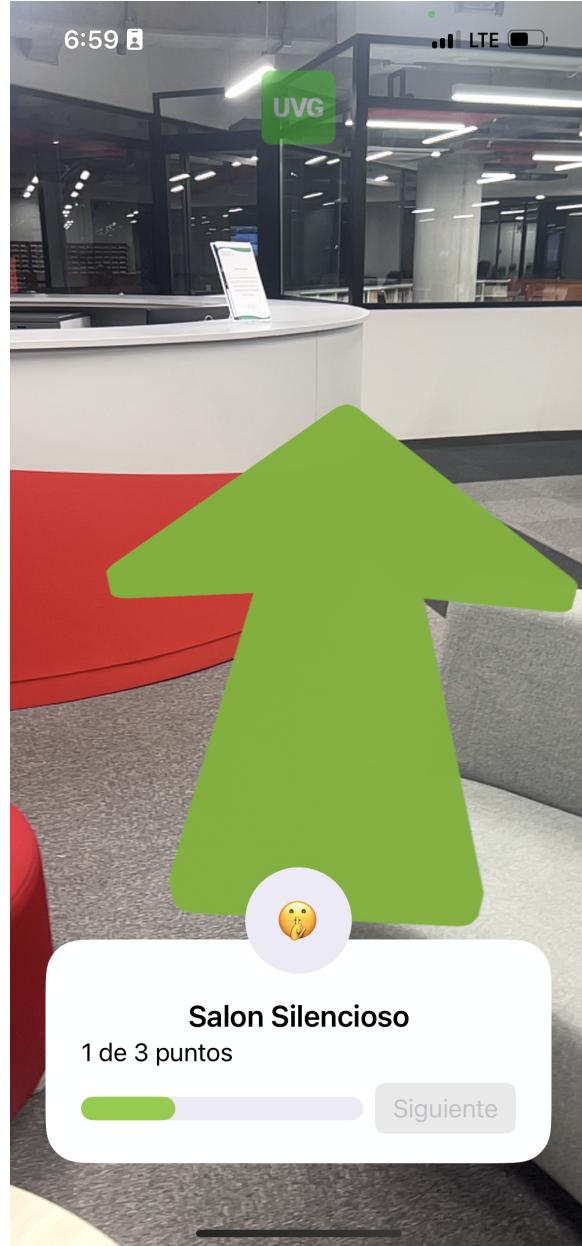


Figura 7.28: Se muestra la dirección hacia la siguiente parada

Cuando el usuario dirige el celular hacia otra dirección, la flecha se actualiza respectivamente:

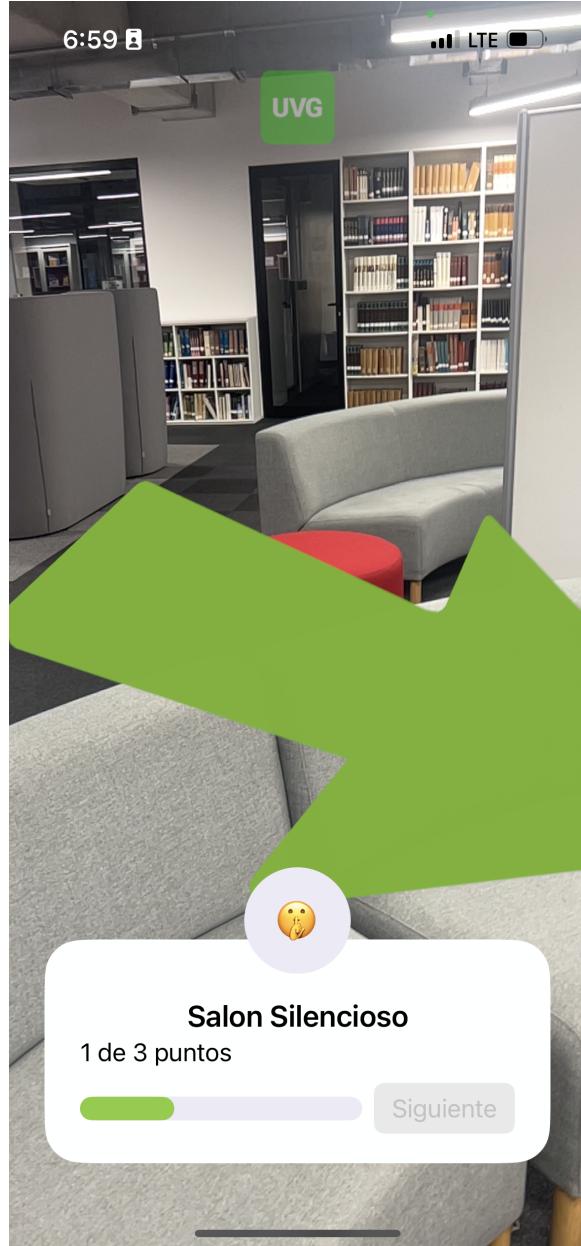


Figura 7.29: Se actualiza la dirección de la flecha

Por último, cuando se acerca al sensor, la flecha desaparece y se vuelve a mostrar el indicador.

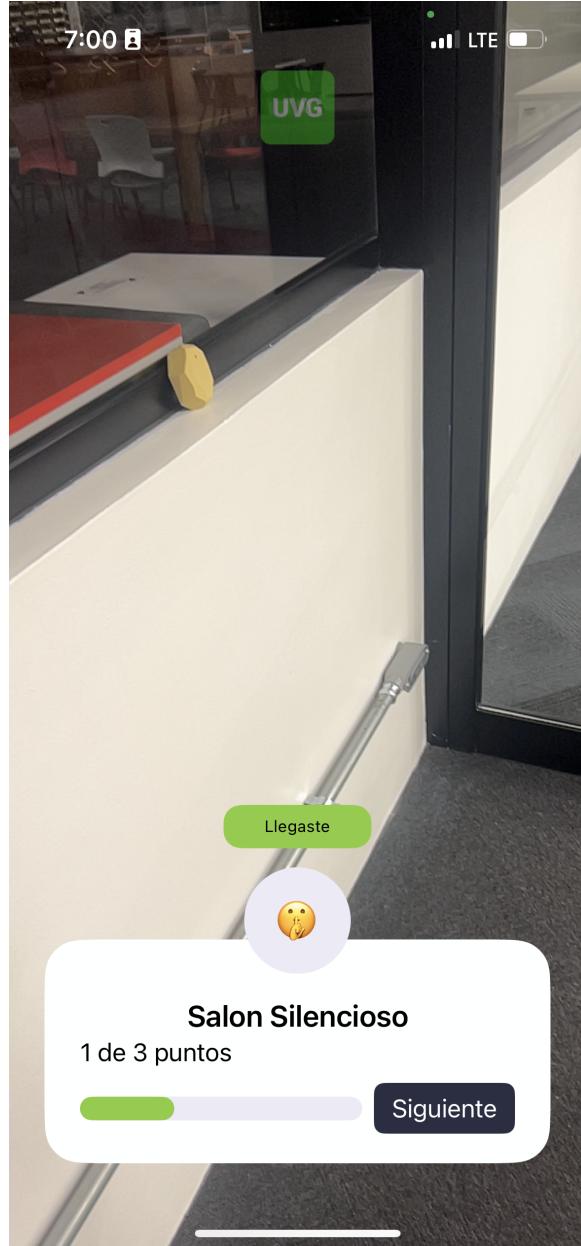


Figura 7.30: La flecha desaparece cuando el usuario lleag

## 7.10. Combinando sensores y Realidad Aumentada para rutas no lineales

Un requerimiento común será dirigir al usuario por sitios donde existan cambios de dirección entre cada punto de interés. Ya que se ha probado previamente la capacidad de detectar los sensores y de dirigir al usuario con realidad aumentada podemos combinar estos dos puntos para crear una mejor experiencia para el usuario final. El concepto que se introduce en el código es el de *waypoints*, el cuál es un término para referirnos a puntos de la ruta que pueden servir como guía para la aplicación y así saber en donde se encuentra el usuario. La idea es poder colocar sensores por los pasillos y así ir ajustando adecuadamente la dirección de la flecha. Mientras más sensores hayan entre cada parada mejor será el funcionamiento de la aplicación. A continuación se muestra visualmente la idea discutida:

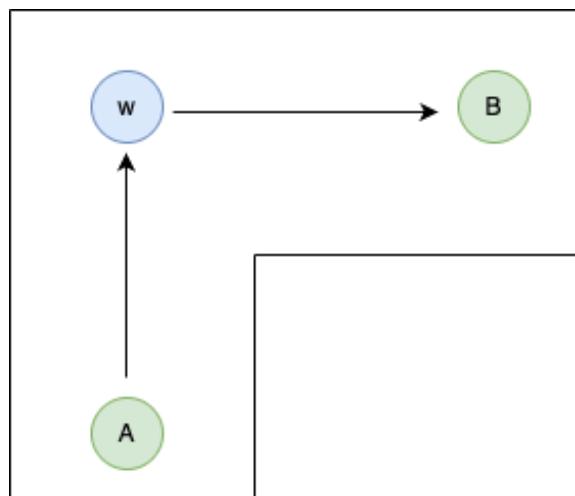


Figura 7.31: El *waypoint* w permite guiar al usuario entre el punto de interés A y B.

Para probar esta funcionalidad se hizo un tour con 2 paradas en la Biblioteca del CIT, utilizando un sensor como *waypoint* para manejar el cambio de dirección. Los puntos a mostrar fueron: Salón N, Salón Silencioso. La aplicación mostró la siguiente información:



Figura 7.32: El usuario ha llegado al punto inicial del tour, el salón N.

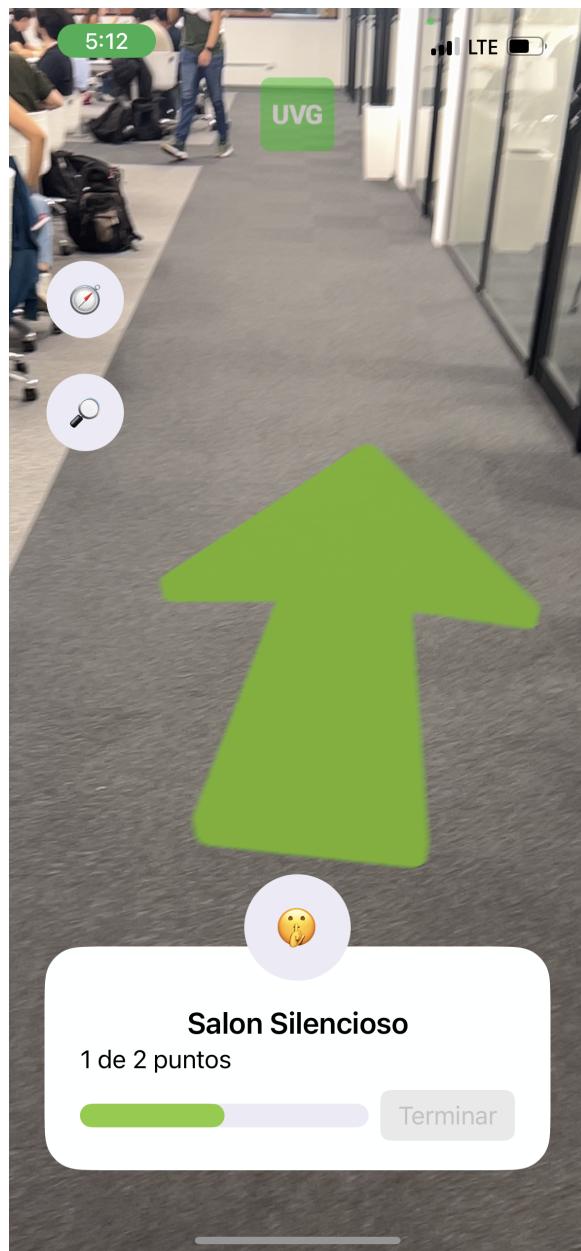


Figura 7.33: El usuario obtiene la dirección a seguir.

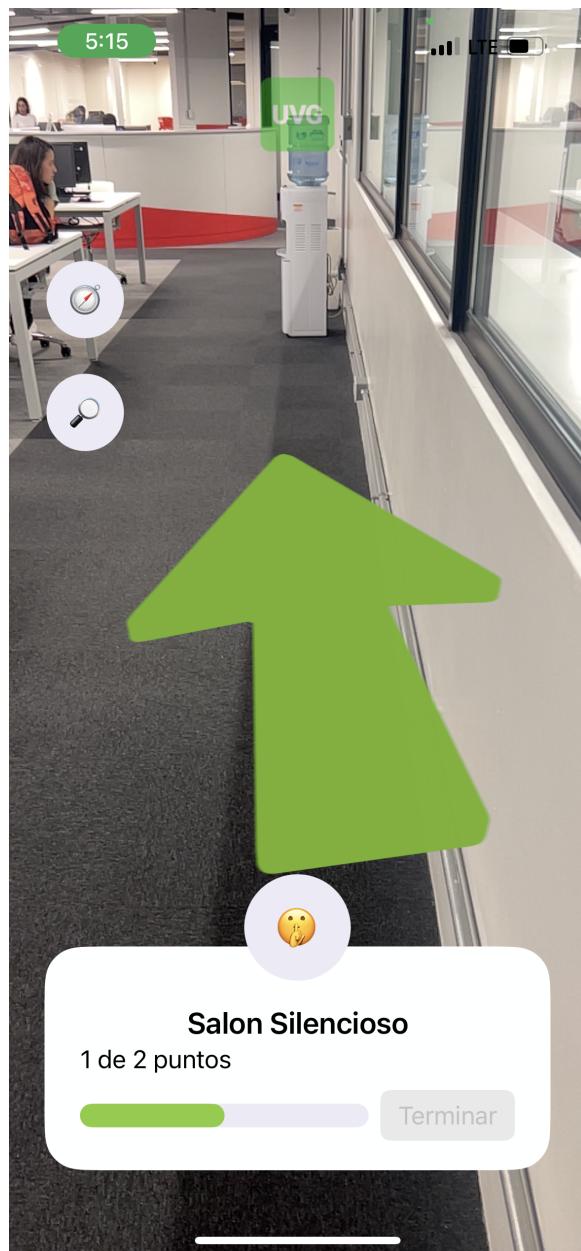


Figura 7.34: El usuario observa la nueva dirección una vez se ha acercado lo suficiente al *waypoint*.

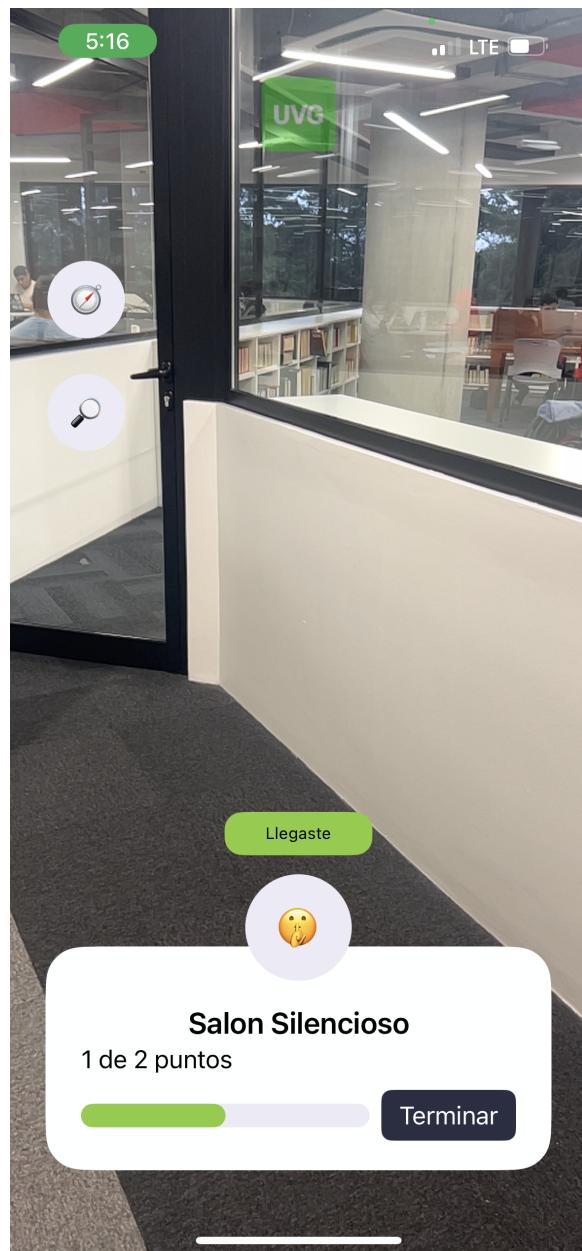


Figura 7.35: El usuario ha llegado al Salón Silencioso.

Un video de esta prueba se puede encontrar en los Anexos.

## 7.11. Aproximando ubicación utilizando el sensor más cercano

Se ha implementado una nueva funcionalidad en la que los puntos de interés pueden ser simulados en función de la distancia del usuario al sensor más cercano. Esto permite ampliar la cantidad de ubicaciones detectables sin necesidad de instalar sensores adicionales. Si un sensor detecta al usuario dentro de un radio configurable (por ejemplo, 10 o 12 metros), la aplicación asume que el usuario está cerca de un punto de interés alternativo. Esta aproximación permite reducir los costos, ya que no es necesario cubrir cada punto de interés con un sensor físico, manteniendo así la experiencia interactiva con un menor número de dispositivos. Esta funcionalidad puede visualizarse en un video adjuntado en el Anexo 5.

Sin embargo, esta metodología tiene limitaciones importantes. Dado que no hay triangulación para determinar la posición exacta, el radio de cada sensor se convierte en una aproximación que podría generar confusiones. En espacios abiertos, es posible que el usuario se encuentre en cualquier punto dentro del rango definido, lo que puede causar cierta ambigüedad al identificar en cuál de los puntos de interés secundarios se encuentra. Esta técnica es más efectiva en entornos acotados como pasillos o salas pequeñas, donde la cercanía física del usuario facilita la detección de un punto de interés con mayor certeza.

A pesar de las limitaciones, esta estrategia representa una solución práctica y eficiente para mantener la funcionalidad de la aplicación sin una inversión significativa en sensores adicionales. En el futuro, se podría complementar este enfoque con otras tecnologías, como BLE o incluso algoritmos predictivos basados en la trayectoria del usuario, para mejorar la precisión. Así, la aplicación seguirá ofreciendo una experiencia fluida y enriquecedora, con un equilibrio entre costos y funcionalidad adecuada para los recorridos virtuales.

## 7.12. Limitaciones observadas

Uno de los principales desafíos observados durante el desarrollo de la aplicación fue la intermitencia en la detección de los sensores UWB por parte de algunos teléfonos. En ciertas ocasiones, el dispositivo móvil no lograba establecer conexión con los sensores, lo que afectaba la precisión de la localización interna y generaba retrasos en las actualizaciones de la interfaz. Esta limitación puede presentarse especialmente en áreas donde la cobertura de los sensores es insuficiente, haciendo que la aplicación pierda noción de la ubicación exacta del usuario. En consecuencia, la experiencia del recorrido virtual puede verse comprometida al no poder brindar indicaciones precisas en tiempo real.

Para mitigar este problema, es fundamental ampliar la red de sensores, asegurando una mayor cobertura en las áreas críticas del recorrido. Con más sensores distribuidos estratégicamente, será posible detectar múltiples señales simultáneamente y aproximar mejor la posición del usuario mediante triangulación. Esta redundancia no solo mejorará la precisión en la localización, sino que también reducirá la probabilidad de interrupciones en la detección. A medida que se integre un mayor número de sensores en la red, la aplicación podrá mantener una mayor consistencia en el seguimiento del usuario, proporcionando así una navegación fluida y confiable en todo el campus.

En cuanto al soporte de la tecnología *UWB*, se observó que esta está presente a partir del *iPhone 11*, por lo tanto cualquier teléfono de esta marca lanzado posteriormente podrá utilizar la aplicación si se decide utilizar esta tecnología.

### 7.13. Localización de la aplicación

La localización en inglés y español se implementó utilizando el sistema de *XCode* llamado *Localizable* (29). Esta herramienta permite implementar sencillamente localización en una aplicación gracias a la estructura que usa para manejar las cadenas de texto. El archivo que se genera al activar esta herramienta se llama *Localizable*. Este archivo contiene un diccionario en los lenguajes que se hayan indicado. En el caso de esta aplicación contiene 2: Español e Inglés. A continuación se observa cómo se presenta la interfaz gráfica de esta herramienta:

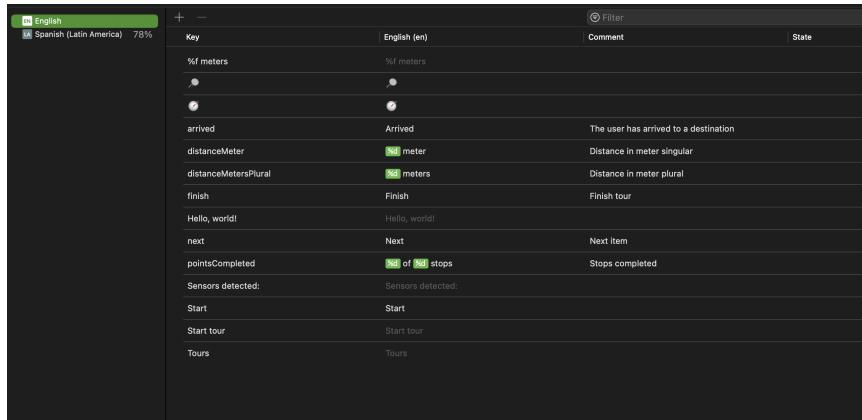


Figura 7.36: Herramienta *Localizable*

En el caso de la traducción de las cadenas de texto, se puede escoger el lenguaje deseado en el panel de la izquierda y llenar los campos requeridos.

| Key                  | Default Localization (en) | Spanish (Latin America) (es-419) | Comment                               | State |
|----------------------|---------------------------|----------------------------------|---------------------------------------|-------|
| %f meters            | %f meters                 | %f metros                        |                                       | NEW   |
| arrived              | Arrived                   | Llegaste                         | The user has arrived to a destination | NEW   |
| distanceMeter        | %d meter                  | %d metro                         | Distance in meter singular            | NEW   |
| distanceMetersPlural | %d meters                 | %d metros                        | Distance in meter plural              | NEW   |
| finish               | Finish                    | Terminar                         | Finish tour                           | NEW   |
| Hello, world!        | Hello, world!             | Hola mundo!                      |                                       | NEW   |
| next                 | Next                      | Siguiente                        | Next item                             | NEW   |
| pointsCompleted      | %d of %d stops            | %d de %d puntos                  | Stops completed                       | NEW   |
| Sensors detected:    | Sensors detected:         | Sensores detectados              |                                       | NEW   |
| Start                | Start                     | Empezar                          |                                       | NEW   |
| Start tour           | Start tour                | Empezar tour                     |                                       | NEW   |
| Tours                | Tours                     | Tours                            |                                       | NEW   |

Figura 7.37: Herramienta *Localizable* para Español

Se siguió este proceso para localizar los componentes visuales de la aplicación. Cabe resaltar que localizar la descripción de cada parada debería ser una tarea asignada al departamento encargado de proveer los puntos de interés. A continuación se observa cómo se ve esta funcionalidad den una de las vistas:

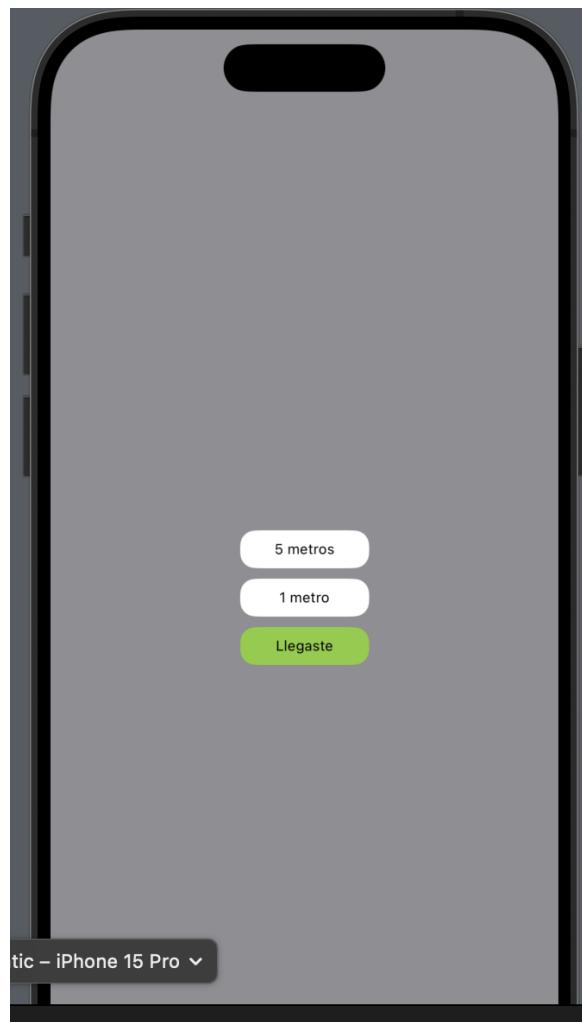


Figura 7.38: Vista con localización en Español



Figura 7.39: Vista con localización en Inglés

## 7.14. Paleta de colores para modo oscuro

La implementación del modo oscuro provee a los usuarios una interfaz más cómoda en entornos con poca iluminación (2). Para esto se ha utilizado la herramienta de *Assets* y *Color Pallets* que *XCode* ofrece. Como se puede observar en la figura 7.40, esta herramienta permite seleccionar colores para modo claro y oscuro. Luego de una selección de colores, la aplicación se ve de la siguiente manera en esta paleta de colores:

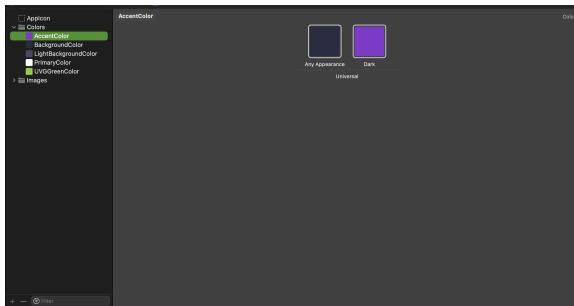


Figura 7.40: Interfaz utilizada para configurar colores en *XCode*

## 7.15. Pruebas Unitarias

Como se ha discutido previamente, las pruebas unitarias permiten validar un correcto comportamiento de la aplicación. Esto permite desarrollar código que sea legible por otras personas y también resistente a los cambios que puedan surgir en los requerimientos. A continuación se muestra una imagen de estas pruebas en *XCode*.

Figura 7.41: Vista de pruebas unitarias en *XCode*

Las pruebas unitarias fueron aplicadas a dos clases clave en el proyecto: *Tour* y *WatchSensorsUseCase*. La clase *Tour* es una entidad que encapsula la lógica de la aplicación relacionada con los recorridos, incluyendo el manejo de las paradas y el estado de finalización del tour. Implementar pruebas unitarias en esta clase permitió garantizar que la lógica de la aplicación respondiera de manera correcta a los cambios en los datos y el estado, asegurando que las funcionalidades críticas de los recorridos fueran robustas y consistentes.

Por otro lado, la clase *WatchSensorsUseCase* representa un caso de uso dedicado a la obtención de información de sensores desde un repositorio. Las pruebas unitarias en esta clase fueron esenciales para validar que la integración con los repositorios de datos funcionara según lo esperado y que se manejara adecuadamente la actualización de la información de los sensores.

La arquitectura MVVM combinada con conceptos de *Clean Architecture* facilitó el desarrollo de estas pruebas unitarias. La separación de la lógica de negocio y la presentación permitió que las pruebas se enfocaran directamente en la lógica sin interferencias en la interfaz gráfica. Esta estructura permite el desarrollo de futuras pruebas que sean escalables y útiles para la validación de esta aplicación.

Dado que este proyecto busca crear bases para una futura incorporación, se realizaron pruebas unitarias únicamente en las funcionalidades más críticas, las cuales son:

reacción al cambio de los sensores y correcto funcionamiento del progreso en un tour. Esto se puede observar en la carpeta *UVGTourTests*. Se decidió incluir inicialmente solo las clases *Tour* y *WatchSensorsUseCase* en las pruebas unitarias debido a su papel en las funcionalidades críticas del proyecto. Estas clases gestionan la lógica de los recorridos y la obtención de datos de sensores, los cuales son aspectos fundamentales para el correcto funcionamiento de la aplicación.

# CAPÍTULO 8

---

## Conclusiones

---

1. Sí fue posible desarrollar una interfaz gráfica que validase el uso de sensores para localización interna en la aplicación de recorridos virtuales en el campus de la Universidad.
2. Se logró mostrar información relevante al usuario sobre los puntos de interés utilizando la información que el sensor más cercano provee.
3. La librería *CoreLocation* permitió acceder a la dirección en la que el celular está siendo sostenido, haciendo posible el cálculo de una dirección hacia el siguiente punto de interés.
4. El uso de pruebas unitarias aseguró el correcto funcionamiento de la aplicación durante el desarrollo. Asimismo, el uso de un controlador de versiones fue implementado correctamente.
5. Se ha utilizado la guía de *Human Interface Guidelines* de *Apple* para incorporar mejoras de accesibilidad en la aplicación, como un modo oscuro y el uso de motores hapticos del celular para brindar retroalimentación al usuario.

# CAPÍTULO 9

---

## Recomendaciones

---

1. Para garantizar la funcionalidad y precisión de la aplicación, es necesario contar con una red adecuada de sensores UWB que cubra todas las áreas clave del recorrido. Dado que la precisión de la localización depende en gran medida de la cobertura proporcionada por estos sensores, se recomienda realizar un estudio previo del entorno para determinar cuántos sensores son necesarios y en qué ubicaciones estratégicas deben colocarse. Esto evitará zonas de baja precisión o puntos ciegos que puedan afectar negativamente la experiencia del usuario. Además, una red más robusta permitirá mantener la coherencia en los puntos de interés, evitando que la aplicación pierda noción de la posición del usuario.
2. Se recomienda experimentar con *Apple Indoor Maps Program* para evaluar la posibilidad de integrar ambas soluciones y ampliar las capacidades de localización de la aplicación (30). *Apple* ofrece una infraestructura diseñada para interiores, lo que podría complementar la precisión de los sensores UWB, especialmente en espacios complejos como edificios grandes o áreas con muchos puntos de interés. La combinación de ambas tecnologías podría proporcionar redundancia y mejorar la experiencia del usuario, permitiendo que la aplicación funcione eficientemente incluso en situaciones donde la cobertura de los sensores sea reducida.
3. Antes de realizar una compra significativa de sensores UWB, es aconsejable probar diferentes marcas para comparar su rendimiento, precisión y facilidad de integración. Cada fabricante puede ofrecer características específicas que se adapten mejor a las necesidades del proyecto, como opciones de configuración avanzada, soporte técnico o precios más competitivos. Estas pruebas preliminares ayudarán a tomar una decisión informada y garantizarán que la inversión en sensores cumpla con las expectativas en términos de fiabilidad, rendimiento y costo a largo plazo.
4. Para asegurar un uso correcto de la batería del celular, se recomienda reducir la frecuencia con la que se solicita la ubicación a un sensor. Esto se puede lograr utilizando los sensores del teléfono para saber si el dispositivo se ha movido

significativamente. Por ejemplo, si el usuario solo se ha movido medio metro, tal vez no sea necesario consultar su proximidad al sensor. Esto también puede ayudar a que otros dispositivos puedan consultar su distancia a los sensores.

5. Para hacer que la aplicación soporte una estructura no lineal de los sensores, se recomienda utilizar otra estructura de datos. Por ejemplo, un grafo no dirigido. Este grafo tendrá como nodos cada uno de los sensores. De esta manera sería posible crear un tour de un punto a otro, siempre y cuando estén conectados en este grafo.

---

## Bibliografía

---

- [1] V. Khorikov, *Unit Testing Principles, Practices, and Patterns*. Shelter Island, NY: Manning Publications, 2020.
- [2] A. Ryhus, “ios human interface guidelines,” <https://bootcamp.uxdesign.cc/ios-human-interface-guidelines-60c9599ad331>, 2023, accessed: 2024-09-15. [Online]. Available: <https://bootcamp.uxdesign.cc/ios-human-interface-guidelines-60c9599ad331>
- [3] AMNH, “Explorer app - american museum of natural history,” <https://www.amnh.org/plan-your-visit/explorer>, 2024, accessed: 2024-09-15.
- [4] J. C. Arguelles, “Hybrid vs. native apps: Which mobile app extends your reach?” <https://appetiser.com.au/blog/hybrid-vs-native-apps/>, 2024.
- [5] I. Apple, “Swift,” <https://developer.apple.com/swift/>, 2024. [Online]. Available: <https://developer.apple.com/swift/>
- [6] R. de Vries, “Get started with swiftui: Apple’s new ui framework,” <https://www.adjust.com/blog/get-started-with-swiftui/>, 2020. [Online]. Available: <https://www.adjust.com/blog/get-started-with-swiftui/>
- [7] A. Inc., “Realitykit - augmented reality,” <https://developer.apple.com/augmented-reality/realitykit/>, 2024, accessed: 2024-09-15.
- [8] ——, “Core location documentation,” <https://developer.apple.com/documentation/corelocation/>, 2024, accessed: 2024-09-15.
- [9] A. Anderson, “Choosing a design pattern for your swiftui app,” [https://medium.com/@alexanderson\\_16451/choosing-a-design-pattern-for-your-swiftui-app-163c06ffcd9b](https://medium.com/@alexanderson_16451/choosing-a-design-pattern-for-your-swiftui-app-163c06ffcd9b), 2024. [Online]. Available: [https://medium.com/@alexanderson\\_16451/choosing-a-design-pattern-for-your-swiftui-app-163c06ffcd9b](https://medium.com/@alexanderson_16451/choosing-a-design-pattern-for-your-swiftui-app-163c06ffcd9b)
- [10] L. Coding, “The best way to structure your ios project,” <https://levelup.gitconnected.com/the-best-way-to-struct-your-ios-project-a2daee7dcb45>, 2024, accessed: 2024-09-15.

- [11] R. C. Martin, *Clean Architecture: A Craftsman's Guide to Software Structure and Design.* Upper Saddle River, NJ: Pearson Education, 2017.
- [12] A. Naumov, “Clean architecture for swiftui,” <https://nalexn.github.io/clean-architecture-swiftui/>, 2024, accessed: 2024-09-15.
- [13] R. R. Rosnita Baharuddin, Dalbir Singh, “Usability dimensions for mobile applications-a review,” *Research Journal of Applied Sciences, Engineering and Technology*, vol. 5, no. 6, pp. 2225–2231, 2013, accessed: 2024-09-15. [Online]. Available: <https://maxwellsci.com/print/rjaset/v5-2225-2231.pdf>
- [14] F. J. Pathari, Y. Nielsen, L. I. Andersen, and G. Marentakis, “Dark vs. light mode on smartphones: Effects on eye fatigue,” in *ACHI 2024 : The Seventeenth International Conference on Advances in Computer-Human Interactions.* IARIA Press, 2024, pp. 150–154. [Online]. Available: <https://www.thinkmind.org>
- [15] S. Chacon and B. Straub, *Pro Git*, 2nd ed. Berkeley, CA: Apress, 2014. [Online]. Available: <https://git-scm.com/book/en/v2>
- [16] Atlassian. (2024) Gitflow workflow. Accessed: 2024-09-16. [Online]. Available: <https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow>
- [17] ACCELQ, “Unit testing: Quality issue identification and assurance,” <https://www.accelq.com/blog/unit-testing/>, 2024, accessed: 2024-09-15.
- [18] I. Estimote, “Why should we work with you?” <https://estimote.com/>, 2024, accessed: 2024-09-15.
- [19] Estimote, “ios estimote uwb sdk,” <https://github.com/Estimote/iOS-Estimote-UWB-SDK>, 2024, accessed: 2024-09-15.
- [20] Kontakt.io, “What is a beacon?” 2024, accessed: 2024-11-03. [Online]. Available: <https://kontakt.io/blog/what-is-a-beacon/>
- [21] GeeksforGeeks, “What is websocket and how it is different from http?” 2024, accessed: 2024-11-03. [Online]. Available: <https://www.geeksforgeeks.org/what-is-web-socket-and-how-it-is-different-from-the-http/>
- [22] S. Concepts, “ios app architecture patterns,” <https://medium.com/swift-concepts/ios-app-architecture-patterns-405931333ebe>, 2024, accessed: 2024-09-15.
- [23] S. Saraswat, “ios architecture patterns: Mvc, mvvm, mvp and viper,” <https://www.appventurez.com/blog/ios-architecture-patterns>, 2024, accessed: 2024-09-15.
- [24] Devsquad, “Low-fidelity wireframing: How to use it in your design process,” <https:////devsquad.com/blog/low-fidelity-wireframing>, 2024, accessed: 2024-09-15.
- [25] Infomaniak, “From uikit to swiftui: Understanding the transition,” <https://news.infomaniak.com/en/de-uikit-a-swiftui/>, 2024, accessed: 2024-09-15.

- [26] I. Meta Platforms, “React - a javascript library for building user interfaces,” <https://react.dev/>, 2024, accessed: 2024-09-15.
- [27] G. Olagunju, “Building a real-time location app with node.js and socket.io,” <https://blog.logrocket.com/building-real-time-location-app-node-js-socket-io/>, 2024, accessed: 2024-09-15.
- [28] K. Ali, “How to localize your ios app in swift using extensions and localization files,” <https://medium.com/@kashif00527/how-to-localize-your-ios-app-in-swift-using-extensions-and-localization-files-eb5b97b4d4b8>, 2024, accessed: 2024-09-15.
- [29] Itsuki. (2021) Localization in ios swift. Accessed: 2024-09-30. [Online]. Available: <https://medium.com/@itsuki.enjoy/localization-in-ios-swift-6b3a7735bd1a>
- [30] Apple Inc., “Apple indoor maps,” 2024, accessed: 2024-10-14. [Online]. Available: <https://register.apple.com/indoor>

## ANEXO A

---

### Código Aplicación Móvil

---

Repository aplicación móvil  
Este proyecto debe ser abierto en *XCode* para poder ejecutarse.  
<https://github.com/guillermoSb/UVGTour>

## ANEXO B

---

### Código Interfaz Gráfica Simulador de Sensores

---

Repository Frontend Simulador Sensores  
Interfaz gráfica desarrollada en *Javascript* para facilitar el uso del simulador de sensores.  
[https://github.com/guillermoSb/UVG<sub>m</sub>egaproyecto<sub>s</sub>ensoresim<sub>f</sub>ront](https://github.com/guillermoSb/UVG_megaproyecto_sensoresim_front)

## ANEXO C

---

### Código Servidor para el Simulador de Sensores

---

#### Repositorio Backend Simulador Sensores

Este proyecto es un servidor desarrollado con *NodeJS* y *WebSocket*. Debe ejecutarse al mismo tiempo que la interfaz gráfica.

[https://github.com/guillermoSb/UVG\\_megaproyecto\\_sensorsim\\_server.git](https://github.com/guillermoSb/UVG_megaproyecto_sensorsim_server.git)

## ANEXO D

---

Demo Biblioteca

---

Video Biblioteca UVG, donde se muestra el uso de la aplicación con un sensor intermedio.

<https://www.youtube.com/watch?v=-vC1dDY0a08>

## ANEXO E

---

### Demo Cafetería

---

Video Cafetería, en el cuál se puede apreciar el intento de aproximación de una ubicación utilizando la información de los sensores.

<https://www.youtube.com/watch?v=bEhkdvdn4Q4>

ANEXO F

## Código para inicializar el servidor del simulador

```
1 import express from 'express';
2 import { Server as WebSocketServer } from 'ws';
3 import http from 'http';
4 import { Sensor, SignalStrength } from './domain/sensor';
5
6 const app = express();
7 const port = 8000;
8
9 // Initialize a simple HTTP server
10 const server = http.createServer(app);
11
12 // Initialize the WebSocket server instance attached to the same HTTP server
13 const wss = new WebSocketServer({ server });
14
15 const sensors: Sensor[] = [
16     new Sensor('1', 10),
17 ];
18
19
20
21 wss.on('connection', (ws) => {
22     console.log('Client connected');
23     // Send the sensors data to the newly connected client every second
24     const interval = setInterval(() => {
25         // Send an event of type 'sensors-update' to the client
26         ws.send(JSON.stringify({ type: 'sensors-update', data: sensors }));
27     }, 2500);
28     // Clear the interval when the connection is closed
29     ws.on('close', () => {
30         console.log('Client disconnected');
31         clearInterval(interval);
32     });
33 });
34
35
```

```

32 | });
33 |
34 | ws.on('message', (message) => {
35 |     // Handle incoming messages
36 |     console.log('received: %s', message);
37 |     const data = JSON.parse(message.toString());
38 |     const { type, body } = data;
39 |     switch (type) {
40 |         case 'request-sensors':
41 |             ws.send(JSON.stringify({ type: 'sensors-update', data: sensors }));
42 |             break;
43 |         case 'update-sensor':
44 |             const { id, signalStrength } = body;
45 |             const sensor = sensors.find((s) => s.id === id);
46 |             if (sensor) {
47 |                 // parse signal strength
48 |                 enum
49 |                 // const
50 |                 parsedSignalStrength = SignalStrength[
51 |                     signalStrength as keyof
52 |                     typeof SignalStrength];
53 |                 sensor.setSignalStrength(
54 |                     signalStrength);
55 |                 ws.send(JSON.stringify({
56 |                     type: 'sensors-update',
57 |                     data: sensors }));
58 |                 // Update the sensors data
59 |
60 |             }
61 |             break;
62 |         case 'add-sensor':
63 |             const { id, signalStrength } = body;
64 |             const sensor = new Sensor(id,
65 |                 signalStrength);
66 |             sensors.push(sensor);
67 |             console.log('added sensor', sensor);
68 |             ws.send(JSON.stringify({ type: 'sensors-update', data: sensors }));
69 |             break;
70 |         case 'remove-sensor':
71 |             const { id } = body;
72 |             const sensorIndex = sensors.
73 |                 findIndex((s) => s.id === id);
74 |             if (sensorIndex !== -1) {
75 |                 sensors.splice(sensorIndex,
76 |                     1);
77 |                 ws.send(JSON.stringify({
78 |                     type: 'sensors-update',
79 |                     data: sensors }));
80 |             }
81 |             break;
82 |     }

```

```
73 }  
74     });  
75 };  
76  
77 app.get('/', (req, res) => {  
78     res.send('Hello World!');  
79 };  
80  
81 // Start the server to listen on the specified port  
82 server.listen(port, () => {  
83     console.log(`Server started on port ${port}`);  
84 });
```