UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

GENERACIÓN DE MAPAS VECTORIALES A PARTIR DE ESCANEOS DE ESPACIOS CON TECNOLOGÍA LIDAR EN DISPOSITIVOS APPLE

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN COMPUTACIÓN

MANUEL ANTONIO ROJAS ASTUDILLO

PROFESOR GUÍA:
BENJAMÍN BUSTOS CÁRDENAS

MIEMBROS DE LA COMISIÓN:
JAVIER BUSTOS JIMÉNEZ
ÉRIC TANTER

SANTIAGO DE CHILE
2023

Resumen

LzIndoorScanner es el nombre del proyecto de desarrollo de una herramienta de *software* que permita agilizar el proceso de creación de mapas vectoriales para la empresa Lazarillo. Lazarillo es una empresa que ofrece plataformas de navegación exterior e interior para usuarios. Estos productos se complementan con un servicio de creación de mapas interiores digitales personalizados de las instalaciones de las empresas con las que se trabaja. El proyecto actual es sobre este último punto y surge con el lanzamiento de RoomPlan, una API liberada recientemente por *Apple* que permite generar modelos tridimensionales a partir del uso de un sensor Lidar y la cámara de un dispositivo móvil compatible.

Utilizando esta novedad como referencia, el trabajo implica el estudio de su viabilidad como digitalizador de espacios y el potencial alcanzable a partir de los resultados que entrega. Luego, entendiendo las imprecisiones naturales esperadas de una tecnología en desarrollo, se definen etapas intermedias que permitan al usuario realizar acciones de edición y filtro de la información relevante a convertir. Así mismo, ante la necesidad de presentarse como una aplicación móvil, se espera que se tomen las consideraciones necesarias en cuanto a experiencia de usuario y usabilidad respecta.

El producto obtenido al final del trabajo logra procesar exitosamente la información generada por RoomPlan, para luego transformarla en un formato compatible con el Editor de Mapas de Lazarillo, el servicio final del flujo de usuario. Si bien se logran alcanzar los objetivos fundamentales, el proceso de conversión del espacio escaneado a un plano aún requiere depuración para casos de lugares más complejos.

Aún así, el sistema es evaluado positivamente por miembros del equipo de operaciones y el CEO de la misma empresa, tanto por lo intuitivo que es el uso de la misma aplicación, como por el valor que podría aportar una vez se corrijan los detalles pendientes en el dibujo del plano final.

Even with these unreliable wings, I'm sure we can fly

Agradecimientos

A Jorge, René y todo el equipo de Lazarillo por darme la oportunidad de trabajar en un entretenido proyecto con ellos en esta importante etapa de mi formación.

A mi profesor guía Benjamín Bustos, quien siempre estuvo atento a mis inquietudes y supo orientarme incluso en momentos donde el problema yacía fuera de lo académico.

A mi pareja Valentina, quien con su compañía y amor ha hecho que este trabajo sea mucho más llevadero y menos agobiante de lo que podría haber sido.

A mis amigos de la universidad, quienes fueron un pilar fundamental en toda esta etapa. Con ellos me divertí y crecí. Me acompañaron, me motivaron y me sostuvieron. Los respeto, los admiro profundamente, y los quiero. Sin ustedes, no sé si lo lograba.

A mi familia, que se alegran y se sensibilizan con mis logros y mis tropiezos, y han provisto de su ayuda y disposición en más ocasiones de las que he merecido. Espero de corazón que esto les alegre porque también son parte de esto.

A mi Tata y a mi Pita. Nada de esto hubiese sido posible sin ustedes. En cada momento en que me senté a trabajar en esto, lo hice pensando en ustedes y en el motivo de orgullo que quiero darles. Son los mejores abuelos que cualquiera podría haber tenido. Los adoro.

A mis hermanos Vicente y Joaquín, que han tenido que lidiar y aceptar los sacrificios que ha significado mi dedicación, y que me han entendido en todo este proceso. No ha sido fácil para ellos. Espero que puedan disfrutar de lo que significa terminar este ciclo.

Y a mi mamá. La persona que siempre dio todo por que yo llegara a este punto, y cuyo sacrificio y valentía no podré nunca dimensionar. Te mereces esto y mucho más. Esto es por ti.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Capítulo 1

Introducción

En la actualidad, distintos movimientos han alzado la voz en pos de reivindicar grupos de personas históricamente dejadas de lado. La accesibilidad y la inclusión han dejado de ser consideraciones secundarias y el uso de tecnologías de información se ha vuelto fundamental en este proceso de empoderamiento. Según la Organización Panamericana de la Salud (oficina regional para las Américas de la Organización Mundial de la Salud), "se estima que casi el 12 % de la población de América Latina y el Caribe vive con al menos una discapacidad, lo que representa alrededor de 66 millones de personas" [43].

Debido a lo anterior, se dificulta ignorar las problemáticas a las que estas mismas personas están expuestas diariamente y no trabajar para atenuarlas. En el presente, caracterizado por la constante presencia de la tecnología en nuestras interacciones diarias, la capacidad de acceder a información seleccionada y actualizada en tiempo real puede impactar significativamente en su autonomía y calidad de vida, especialmente en quienes que cuentan con algún tipo de discapacidad.

Bajo esta premisa, Lazarillo ha surgido como un referente en lo que a accesibilidad refiere. Lazarillo [30] es una empresa que se dedica a asistir a la accesibilidad universal desde la parte de la información, poniendo especial énfasis en personas en situación de discapacidad. A través de su aplicación GPS homónima, Lazarillo ofrece una experiencia de navegación exterior e interior con facilidades para personas con discapacidades visuales o de movilidad. Ella brinda información en tiempo real sobre la ruta por la cual el usuario transita, entregando orientación con respecto al entorno y notificando lugares de interés cercanos como: paraderos de transporte público, vías de accesibilidad para personas con discapacidades motoras, tiendas, lugares de comida, centros de salud, bancos y cajeros automáticos, entre otros.

El servicio de navegación interior representa uno de los principales desafíos que Lazarillo debe enfrentar para brindar una experiencia completa. La disponibilidad de este servicio es un aspecto fundamental de la propuesta con que se vende la empresa. La generación de mapas interiores, precisos y detallados, es un proceso que no está libre de dificultades. Si bien, Lazarillo ya dispone de la capacidad para llevar a cabo esta etapa, el proceso actual es lento y manual, requiriendo tiempo y recursos considerables. Lazarillo solicita a las empresas que recurren a ellos los planos del espacio que se desea digitalizar, y un equipo se encarga de dibujarlo a mano en una herramienta de visualización de mapas. La dependencia de planos

proporcionados por estas empresas se topa frecuentemente con desafíos de disponibilidad, actualización y calidad. Además, este proceso no solo ralentiza la expansión del servicio ofrecido por Lazarillo, sino que también aumenta los costos operativos y limita la eficiencia de la empresa.

En este trabajo se busca abordar y superar estos desafíos críticos relacionados a la generación de mapas interiores e, indirectamente, con la capacidad de crecimiento de la empresa. Se propuso desarrollar una solución innovadora que agilice y simplifique el proceso de digitalización de espacios, reduzca la dependencia de los planos desde las empresas asociadas y el tiempo de ejecución de la tarea y, por lo tanto, aumente la eficiencia del proceso en general.

La propuesta del trabajo se basa en la tecnología Lidar, cuya disponibilidad y accesibilidad aumentó recientemente tras su implementación en algunos de los últimos modelos de dispositivos de la empresa *Apple*. Esta tecnología consiste en un método de determinación de distancias a través de la emisión de un láser. De la mano con esto, *Apple* ha estado liberando una serie de *APIs* y Kits de desarrollo que entregan cada vez más oportunidades a los programadores que quieren aprovechar esta novedad.

1.1. Contexto y desafíos

Lazarillo ha demostrado su capacidad de ofrecer un servicio de navegación en espacios interiores y ha logrado llevar a cabo la digitalización de dichos espacios por medio de un proceso manual. Sin embargo, esta labor requiere que el equipo ejecute tareas propensas a errores de precisión, burocracia y demandantes de tiempo. Por lo tanto, la necesidad de obtener planos actualizados y precisos se topa con problemas de tiempo y disponibilidad. En muchos casos, el equipo de Lazarillo encargado de este aspecto (llamado "Equipo de operaciones" o simplemente "operaciones" de ahora en adelante) se ve retrasado por circunstancias ajenas a sus capacidades.

El trabajo de esta memoria consiste en evaluar las herramientas que provee *Apple* para el uso de Lidar en sus dispositivos y desarrollar una aplicación móvil que haga uso de ellas. La aplicación debe ser capaz de producir una colección de datos que represente el espacio escaneado por medio del sensor Lidar, visualizar una reproducción preliminar que el usuario pueda usar para verificar los resultados de la captura y ofrecer herramientas de edición que permitan realizar ajustes de limpieza del modelo generado. Además, debe ser capaz de conectarse con el servicio de base de datos que Lazarillo usa actualmente en sus otros productos para así contar con la información de los lugares reales que se disponen y de publicar información en una servicio relevante en producción. Finalmente, los usuarios objetivos de la aplicación son los miembros del equipo de operaciones y por tanto su funcionalidad y usabilidad debe ser aprobada por el mismo equipo.

1.2. Solución propuesta y objetivos

Se busca que la solución logre agilizar el proceso, reducir los recursos y el tiempo que éste toma. En lugar de depender de los planos proporcionados por los clientes, se desea contar con una herramienta que sirva de alternativa al proceso actual -de responsabilidad compartida entre cliente y Lazarillo- por uno que sólo dependa de este último.

1.2.1. Objetivo general

La finalidad de este trabajo busca cumplir el siguiente objetivo general:

- Desarrollar un sistema que permita digitalizar espacios interiores, mediante la captura del entorno y la transformación de ese resultado a mapas vectoriales.

1.2.2. Objetivos específicos

Para lograr con el anterior objetivo general, se espera que esta herramienta cumpla con los siguientes objetivos específicos:

1. Capturar conjunto de datos que representen el escaneo de una habitación.
2. Identificar componentes del espacio y sus características, dado el conjunto de datos obtenido.
3. Generar mapas vectoriales en un formato relevante para *OpenMapTiles*[1] con soporte *indoor*, a partir de las componentes identificadas.
4. Guardar y asociar los mapas vectoriales a un *Place*[2], para así poder enlazar planos distintos en un mismo mapa.
5. Generar y agregar *tags* [3] [42] a componentes del mapa.
6. Evaluar la viabilidad de encontrar coincidencias entre un modelo ya guardado y un escaneo en curso para determinar la posición del usuario.
7. Evaluar la usabilidad de la herramienta.

1.3. Organización del contenido

En las próximas secciones, se introducen los diversos conceptos y herramientas que guiarán el desarrollo de esta solución. También se discuten las consideraciones específicas planteadas

[1] Ver en Sección 2.10
[2] Nombre de la entidad en la base de datos que se utiliza para identificar un lugar específico
[3] Ver en Sección 2.15.1

por el equipo de Lazarillo, y se aborda la estructura de la aplicación y las decisiones de diseño del proceso de creación.

Este proyecto, que inició en medio de la incertidumbre y la novedad tecnológica que representa trabajar bajo tecnologías nuevas, se ha convertido en un esfuerzo por aprender y utilizar la tecnología Lidar para llevar a cabo la generación de mapas vectoriales de manera funcional y más eficiente. El proceso de desarrollo ha significado una experiencia importante en resolver desafíos técnicos nuevos y ha permitido alcanzar descubrimientos importantes que impulsarán el crecimiento de Lazarillo y de tecnologías proveedoras de accesibilidad y autonomía para usuarios.

Capítulo 2

Estado del arte y definiciones

En el presente capítulo se comentarán las herramientas y tecnologías investigadas para llevar a cabo el trabajo. Se incluyen herramientas efectivamente utilizadas y que forman parte de la solución planteada, como también aquellas que sólo sirvieron de referencia en la definición del producto final.

2.1. Swift

Swift es un lenguaje de programación desarrollado por la empresa *Apple* -en conjunto con la comunidad *Open-source* para desarrollar aplicaciones de *iOS*, *MacOS*, *Apple TV* y *Apple Watch* [11]. Es un lenguaje multiparadigma, fuertemente tipado, definido por su comunidad como "seguro, rápido y expresivo" [23]. Surge como reemplazo a *Objective-C*, el lenguaje en el que hasta entonces se desarrollaban todas las aplicaciones para *iOS* y *OS X*[1], pero ofreciendo la posibilidad de coexistir con código de este. Estos dos son los únicos lenguajes que *Apple* permite para desarrollar aplicaciones en sus entornos.

2.2. MVVM

El patrón de arquitectura de software *Model-View-ViewModel* (MVVM) es un patrón que ayuda a mantener una separación limpia entre la lógica interna de la aplicación y la definición de sus interfaces de usuario. Esto facilita varios asuntos de desarrollo y facilita las tareas de *testing*, mantención y escalamiento, así como también incentiva la reutilización de componentes y el trabajo simultáneo de desarrolladores [38].

Se distinguen tres componentes fundamentales en el patrón: el modelo, la vista, y el modelo de vista.

1. El modelo refiere a las clases no visuales que encapsulan la información de la aplicación,

[1]Sistemas operativos desarrollados por *Apple* utilizados exclusivamente en su propio *hardware*.

la transferencia de la misma y la lógica.

2. La vista es responsable de definir la estructura, disposición y apariencia de lo que el usuario verá en pantalla.
3. El modelo de la vista implementa propiedades y comandos con que la vista se relaciona con la lógica interna. Se preocupa ,por ejemplo, de notificarle cambios en la información relevante disponible.

En principio, se puede considerar similar al patrón *MVC* (Modelo-Vista-Controlador). MVVM, en comparación con MVC, facilita el enlace bidireccional de datos, simplificando la sincronización entre la lógica de presentación y la interfaz de usuario. Además, permite la reutilización eficiente del ViewModel en diversas Vistas, lo que ahorra tiempo y recursos de desarrollo. Es importante indicar que la tarea de realizar pruebas unitarias se vuelve más sencilla debido a la clara separación de responsabilidades, mejorando también la mantenibilidad al reducir la probabilidad de efectos secundarios no deseados durante las actualizaciones de la interfaz de usuario.

2.3. JavaScript

JavaScript es un lenguaje de programación ligero, interpretado, compilado, dinámico, y con soporte para programación funcional. Es particularmente conocido por su uso en el desarrollo de páginas y aplicaciones web, aunque también es importante en el desarrollo de intérpretes [39].

Su capacidad para ejecutarse en el navegador del usuario lo convierte en un componente esencial para agregar comportamientos y funcionalidades a las páginas web. Además, su naturaleza interpretada y dinámica significa que los desarrolladores pueden escribir y probar código rápidamente, lo que lo hace versátil y adecuado para proyectos ágiles.

2.4. SceneKit

SceneKit es una interfaz de programación de aplicaciones (*API*) de gráficos en 3D de *Apple*. Es un *framework* escrito en *Objective-C* (y por transición, compatible con *Swift*) diseñado para ser de alto nivel y capaz de proveer de herramientas de "alto rendimiento" y uso "intuitivo" [10].

Se distingue de otras *APIs* como *OpenGl* y *Metal*[2] por ser de alto nivel: requiere sólo descripciones superficiales de escenas y de las acciones que el desarrollador espera que estos objetos realicen. Es decir, en pocas líneas de código *SceneKit* permite agregar animaciones, simular físicas, efectos de partículas, renderizar realísticamente el contenido entregado y soportar interacciones del usuario, sin necesidad de profundizar en la lógica detrás de estas acciones.

[2]Otra API de gráficos 3D de *Apple*

Funciona manteniendo un grafo de escena. Cada elemento de esta escena se agrega como nodo hijo a un nodo raíz, pudiendo representar cámaras, iluminación, o los mismos objetos geométricos de la escena. Desde este punto del informe, en algunas ocasiones se referirá a los elementos de una habitación escaneada como nodos de la escena tridimensional.

2.5. ARKit y RealityKit

ARKit y *RealityKit* son *frameworks* desarrollados por *Apple* para trabajar en realidad aumentada (AR[3]).

ARKit es una plataforma que puede capturar y procesar el movimiento del dispositivo y la escena de la cámara para construir experiencias de realidad aumentada [2].

RealityKit es una plataforma que implementa renderizado y simulación en 3D. Para esto, aprovecha la captura de información de *ARKit*. También es posible importar *assets* propios, agregar audio, animar objetos, incorporar física a la escena, responder a la interacción del usuario y cooperar con otros *frameworks* del *ARKit*[7].

2.6. RoomPlan

RoomPlan es una de las últimas *APIs* impulsadas por *Apple*. Se basa en las herramientas que otorgan *ARKit* para capturar información, y en las de *RealityKit* para renderizarla. Utiliza la cámara del dispositivo y un escáner Lidar (incorporado en las últimas versiones de *iPhone* y *iPad*) para crear un plano en 3 dimensiones de una habitación[8].

RoomPlan es capaz de recrear y reconocer paredes, ventanas, puertas y aberturas generales en la habitación que escanea. Así también puede distinguir una serie de objetos misceláneos que pueden ser encontrados en interiores como es el caso de mesas, sillas, camas, refrigeradores, chimeneas, etc[9].

2.6.1. Clases y estructuras definidas en RoomPlan

CapturedRoom

La estructura `CapturedRoom` representa una encapsulación del resultado del proceso de escaneo (Ver ejemplo en anexo A.2). Las propiedades que posee esta estructura son:

1. *doors*: arreglo de tipo *Surface*

[3] *Augmented Reality* o Realidad Virtual refiere a las experiencias de usuario que incorporan hacia su vista en vivo elementos en 2 o 3 dimensiones desde la cámara de un dispositivo, de manera tal que dichos elementos aparenten habitar en el mundo real.

2. *windows*: arreglo de tipo *Surface*
3. *walls*: arreglo de tipo *Surface*
4. *openings*: arreglo de tipo *Surface*
5. *objects*: arreglo de tipo *Object*
6. *Confidence*: nivel de certidumbre de la clasificación asignada al objeto. Valor puede ser *high*, *medium* o *low*.

Surface

Área bidimensional de una habitación que el *framework* identifica como una superficie. Sus propiedades importantes son *category*, *confidence*, *transform* y *dimensiones*. *category* corresponde a una enumeración [4], e indica qué tipo de superficie RoomPlan predice que es. Las posibilidades contempladas son los casos *wall*, *window*, *door* y *opening* [3].

Object

Área tridimensional de una habitación que el *framework* identifica como un objeto misceláneo que no corresponde a una de las categorías contempladas como *Surface*. Refiere a objetos de menor relevancia que los anteriores, puesto que no definen el espacio de forma tan significativa como ellos. Sus propiedades importantes son las mismas que están definidas para *Surface*, aunque ahora los posibles valores de *category* son *bathtub*, *bed*, *chair*, *dishwasher*, *fireplace*, *oven*, *refrigerator*, *sink*, *sofa*, *stairs*, *storage*, *stove*, *table*, *television*, *toilet*, y *washerDryer* [6].

2.7. Lidar

Nace del acrónimo en inglés LiDAR para "*Light Detection and Ranging*", y es el nombre con que se llama a la técnica de sensores remotos que utiliza luz en forma de láser de pulso infrarrojo para medir distancias, las que se definen según el tiempo que demoran los pulsos en rebotar y volver a la fuente de luz[40]. Para realizar el mapeo de un lugar se requiere un instrumento Lidar, que consiste de dicho láser sumado a un escáner, como mínimo.

Su principal uso actual es en el área de la topografía; área de estudio que consiste justamente en la digitalización de terrenos y generación de mapas. En rigor, Lidar puede ser utilizado en cualquier situación donde se deseen medir superficies, y tiene aplicaciones también en arqueología, meteorología, planificación urbana y minería, por nombrar algunas[18].

Estos sensores se volvieron populares con la aparición de los autos eléctricos. En ese entonces, y aún más antes, el uso de esta tecnología no era masivo dado su alto precio[13].

[4]Tipo de dato definido por el desarrollador, que tiene un conjunto fijo de posibles valores relacionados bajo algún criterio

Recién desde el año 2018 se registra la aparición de estos sensores a precios considerablemente menores, que permiten que otras empresas puedan estudiar la posibilidad de incluirlos en sus dispositivos.

2.8. OpenStreetMap

OpenStreetMap (OSM) es un proyecto de código abierto que proporciona información geográfica, enfocada en mapas para sitios web y aplicaciones móviles[41]. Esta plataforma se ha convertido en la base de datos fundamental para muchos proyectos de mapas de código abierto y goza de una comunidad activa de colaboradores y usuarios. Su enfoque en la accesibilidad y la colaboración abierta ha hecho de *OpenStreetMap* una fuente invaluable de datos geoespaciales precisos y actualizados, utilizados en una amplia variedad de aplicaciones, desde navegación hasta análisis geográficos. Su naturaleza de código abierto fomenta la contribución de datos por parte de usuarios de todo el mundo, lo que lo convierte en un recurso global para la representación de información geográfica.

2.9. VectorTiles

Los *VectorTiles* son paquetes de información que contienen datos geográfico en un formato especializado para la representación de mapas web de celda con estilos.

Poseen diversas ventajas con respecto a una representación completa y no dividida de la información de un mapa. Por ejemplo, la transferencia de información entre clientes se reduce a sólo la información que fue efectivamente solicitada por el usuario, pues según lo que este desee visualizar -según la ubicación y el nivel de zoom aplicado en la vista-, se envían y procesan únicamente los paquetes que corresponden. Una reducción en la transferencia de información tiene consecuencias positivas en la velocidad de carga, la que a su vez se traduce en permitir disponer de mapas con mejores resoluciones y en interacciones de usuario más fluidas.

Una segunda ventaja refiere a la estilización de mapas. El formato permite ajustar dinámicamente los elementos de la apariencia de los mapas, como si de capas se tratase, sin necesidad de volver a descargar la información fundamental.

Por último, el acceso a las propiedades de la información del mapa también es más rápido y fluido con respecto a tener una única gran e indivisible estructura de información que contiene todos los datos del mapa[37].

2.10. OpenMapTiles

OpenMapTiles es un proyecto extensible de código abierto que provee un conjunto de herramientas para trabajar con mapas personalizados.

Sirve de plantilla para implementaciones personalizadas y utiliza la información públicamente disponible de *OpenStreetMap* para compartimentalizarla en *VectorTiles* de cada rincón del planeta, lo que permite una representación eficiente y flexible de mapas en una variedad de plataformas. Puede ser desplegado en sitios web utilizando visualizadores de *JavaScript*, o en aplicaciones móviles en *Android* y en *iOS*. Al igual que *OSM*, su naturaleza de código abierto y extensible lo convierte en una opción popular para proyectos que requieren mapas personalizados y geolocalización.

2.11. Leaflet

Leaflet es una librería de código abierto escrita en JavaScript que ofrece una solución versátil y sencilla para la implementación de mapas interactivos en aplicaciones web y móviles. Esta librería, heredera de OpenMapTiles, destaca por su facilidad de uso y su capacidad para funcionar en una amplia variedad de plataformas, incluyendo dispositivos de escritorio y móviles [36].

Leaflet se destaca por su diseño modular y su extensibilidad a través de plugins. Esto permite a los desarrolladores personalizar y ampliar fácilmente las funcionalidades de los mapas, adaptándolos a las necesidades específicas de sus proyectos. Además, *Leaflet* cuenta con una comunidad activa que contribuye con una amplia gama de *plugins* adicionales, lo que enriquece aún más su conjunto de herramientas.

2.12. GeoJSON

GeoJSON es un formato de archivo que representa datos geoespaciales, basado en el más genérico *JavaScript Object Notation* (*JSON*). Es capaz de codificar una variedad de estructuras de datos geográficos, utilizando un sistema de referencia de coordenadas geográficas y unidades de grados decimales [16].

Las geometrías válidas dentro del estándar de *GeoJSON* son:

- *Points*: Par de coordenadas x e y.
- *MultiPoint*: Arreglo de coodenadas de puntos.
- *LineString*; Arreglo de posiciones x e y que componen una línea.
- *MultiLineString*: Arreglo de coordenadas de *LineStrings*.
- *Polygons*: Representación de la curva que define una superficie. Puede ser definido con sacados (hoyos).
- *MultiPolygon*: Arreglo de coordenadas de polígonos.

Un ejemplo de *GeoJSON* válido se muestra en el anexo A.1, y corresponde a un "*LineString*".

2.13. Firebase

Firebase es un conjunto de plataformas de desarrollo de aplicaciones y servicios en la nube, desarrollada por *Google* [19]. Dentro de estos servicios y soluciones ofrecidos, se encuentran productos que apoyan en lo que refiere a compilación, lanzamiento, analítica, interacción remota, bases de datos en tiempo real, almacenamiento en la nube y autenticación, entre otros.

Para este trabajo, dentro de los servicios provistos por *Firebase* se utilizarán aquellos relacionados con la autenticación de usuarios y administración de la base de datos.

2.13.1. Firebase Realtime Database

Firebase Realtime Database es el nombre del servicio de base de datos que ofrece *Firebase*. Es una base de datos *NoSQL* con algunas optimizaciones y funcionalidades extra comparada con una base de datos relacional. Está alojada en la nube, que almacena los datos en formato *JSON*, y ofrece sincronización en tiempo real con cada cliente conectado, incluso entre plataformas distintas (como entre *Android* y *Apple*) [21].

2.13.2. Firebase Authentication

Firebase Authentication proporciona servicios de *backend*, *SDK* y bibliotecas de interfaces de usuario ya elaboradas para autenticar usuarios en una aplicación. Admite autenticación mediante contraseñas, números de teléfono, *Google*, *Facebook* y *Twitter*, entre otros. Se integra estrechamente con otros servicios de *Firebase* y aprovecha estándares como *OAuth 2.0* y *OpenID Connect* [20].

2.14. Lazarillo APP

LazarilloAPP es la aplicación principal de Lazarillo. Ofrece asistencia de navegación en exteriores y en interiores. En este último caso es necesario contar con los planos respectivos, que debiesen ser otorgados por las empresas interesadas en ofrecer este servicio.

Cuando una empresa establece un contrato de trabajo con Lazarillo para digitalizar sus espacios, el procedimiento es el siguiente:

1. Se solicitan los planos en formato *PDF* o como imágenes.
2. Se verifican que los datos recibidos sean los correctos (y actualizados).
3. Se realiza una visita a terreno, o se solicita que cliente envíe fotos del espacio para corroborar la información.
4. Se crea el mapa.

2.15. Editor de mapas de Lazarillo

El *Editor de mapas* de Lazarillo es una aplicación web propia de la empresa, desarrollada en el lenguaje *JavaScript*, para crear y editar los mapas vectoriales que se utilizan en todos los proyectos. Para acceder a ella se le debe proveer del identificador del *Place* cuyos mapas se quieren revisar o editar. Permite agregar y borrar polígonos, líneas y nodos, entre otras cosas. Se compone de un proyecto de *front-end* y un proyecto de *back-end*, siendo este último el proyecto con el cual será necesario comunicarse para poder agregar mapas nuevos.

Es este editor desde donde se deben enviar los mapas nuevos para que sean visibles en los otros proyectos de Lazarillo, como la aplicación principal "LazarilloApp". Esto se hace desde una opción llamada "Sincronizar", la cual podría considerarse como el último paso para completar un flujo de usuario de la aplicación de este trabajo.

2.15.1. *Tags*

Los *tags*, o etiquetas, refieren a una caracterización que se utiliza en *OpenStreetMaps* para definir cómo se representará una figura en el mapa. Lazarillo por su parte, agregó en su versión etiquetas adicionales de para elementos con un estilo personalizado.

Esta caracterización corresponde a un estructura de tipo diccionario con una serie de pares llave-valor, que el *frontend* lee a la hora de definir la configuración y la estética de la representación del elemento en la capa del mapa que se está visualizando.

Estas etiquetas se utilizan para dar al *frontend* información sobre el elemento asociado. De ellas se extrae:

- El ID del Piso del Lugar en el que debe ser dibujado: Es el identificador generado por *Firebase* para cada piso existente.
- El nivel del Piso del Lugar: utilizado como nombre corto para que el usuario pueda reconocerlo de manera amigable (y no con un ID irreconocible).
- El color de fondo del polígono: En el Editor de Mapas se define una serie de estilos para diferenciar espacios según la usabilidad o disponibilidad del lugar. Algunos ejemplos son "Área principal", "Baños" o "Zona restringida". Si bien estos nombres son más útiles para Lazarillo que para los usuarios -ya que estos últimos sólo verían la diferencia de color en los mapas-, si ayudan a que uno pueda inferir el significado de cada zona.
- Ícono representativo: Algunas zonas o nodos pueden asociarse a un ícono que indique el rol que tiene en el lugar en el que se enmarca. Una tienda comercial puede tener un ícono que indique "*Shopping*", o una entrada al recinto puede tener un ícono de "Puerta".

Por ahora, la creación de estas etiquetas sólo se produce en el servicio de *backend* de manera automática al crear un elemento. Algunas de ellas son inferidas por datos que se

envían desde el *front* al crear un elemento, como el caso del `floorId` y `level` (del Piso). Otras son asignadas por defecto, como las etiquetas que definen el color de relleno del polígono. Y otras sólo pueden ser agregadas después de haber creado un elemento, como en el caso de las etiquetas para nodos. Notar que siempre es posible editarlas o agregar otras nuevas.

2.16. Portal Lazarillo

Portal Lazarillo es la plataforma que utiliza la empresa para la administración de todo lo relacionado a la información desplegada en mapas, eventos y lugares. Difiere del Editor de mapas en que este sólo contempla herramientas de carga y edición de las capas de los mapas personalizados. Es necesario utilizar el Portal para poder configurar un lugar (su creación, la creación de pisos internos, y la visibilidad de los mapas de sus pisos), y sólo después de eso será posible la carga de un mapa.

Una vez configurado un Lugar, es posible cargar mapas desde el editor sincronizando los cambios de este último.

Capítulo 3

Diseño de Vistas

En el siguiente capítulo, se entra en detalle sobre el trabajo de diseño y creación de interfaces gráficas. Para esto se mencionan las distintas alternativas evaluadas para la realización del trabajo, se justifica la elección de aquellas que fueron consideradas relevantes, se explican las expectativas de su implementación y se concluye con el estado final alcanzado.

Para lograr los objetivos establecidos, lo que se debe desarrollar es un *parser* que reciba un modelo 3D de un espacio interior -generado por el kit de desarrollo de *Apple* -utilizando un dispositivo con *Lidar* y cámara (como *iPhones* o *iPads*), y genere un mapa en formato *VectorTile* utilizando alguna herramienta del *framework OpenMapTiles*. Se debe asegurar que la aplicación de escaneo entregue una descripción cualitativa del espacio en cuestión y los elementos presentes en este, indicando las medidas de sus dimensiones y que permita asociar escáneres distintos a un lugar físico, en la base de datos de Lazarillo.

3.1. Algunas consideraciones y requerimientos

Para efectos de definir un plano de la habitación a escanear, sólo se considerará -al menos preliminarmente- para las delimitaciones de la habitación (como muros o puertas) las dimensiones de largo y altura, y no ancho; esto será obviado de aquí en adelante. Así también serán omitidos los muebles y objetos varios (refiriendo a instancias de la estructura *Object* de *RoomPlan*), dado que para esta etapa del trabajo no son relevantes en la confección del plano.

Además, durante las pruebas que se han hecho de la *API*, se ha notado que esta tiene problemas para detectar superficies no rectangulares, por lo que también se reducen las formas posibles de una superficie que se tendrán en cuenta.

Se utilizará el patrón arquitectural de diseño de software *MVVM*, que permite diferenciar y distribuir en distintos componentes las responsabilidades de la lógica de la aplicación y de las interfaces gráficas. Este patrón es especialmente compatible con proyectos en *Swift* debido a la separación que este mismo lenguaje provee entre la definición de interfaces de usuario con la lógica a la que se atañen y la designación de archivos en el proyecto (de tipo

.storyboard) para definir cada vista unívocamente.

Por último, por parte de Lazarillo se hace especial énfasis en la necesidad de contar con un control de acceso de usuarios a la herramienta. Esta podrá realizar acciones de carácter destructivo sobre información sensible, por lo que sólo personal de Lazarillo debiese poder utilizar todas sus funcionalidades.

3.1.1. Sobre modelo de datos

En el modelo de datos utilizado por Lazarillo, existen 2 entidades (Instituciones y lugares) fundamentales para el desarrollo de este trabajo, de las cuales se extraen 3 tipos de identidades que se tienen que identificar y *parsear*: *Instituciones* (`Institution`), *Lugares* (`Place`) y *Pisos* (`innerFloor`). La relaciones entre ellas son: Instituciones y Lugares de 1:M, y Lugares-Pisos también de 1:M.

Existen dos atributos importantes en lo que a este trabajo compete, en las entidades de tipo de lugar: `parent_institution` e `innerFloors`. El primero es un atributo de tipo *String* que contiene el ID de la Instutución a la que está asociado. Y el segundo es de tipo diccionario donde las llaves son el identificador de cada Piso del Lugar, y el valor de la llave es el *JSON* que describe tal Piso.

También se debe notar que Lazarillo define como entidades de tipo Lugar a varias ubicaciones con distinto nivel de relevancia. Tanto un centro comercial completo como un singular cajero automático cumplen con ser un Lugar. Por lo mismo, no todos las entidades de Lugar poseen el atributo `innerFloors`. Esta será una restricción para los lugares a los que se les desea agregar un mapa: el atributo `innerFloors` debe ser no vacío y poseer al menos 1 elemento.

3.1.2. Relación con el editor de mapas

Para acceder al editor de mapas, es necesario conocer el identificador del *Place* al que se le quiere agregar un mapa nuevo. Se tendrá que realizar una consulta a la base de datos en Firebase y así obtener todas las entidades existentes de tipo Institucion y Lugar. Se pretende que el usuario tenga que seleccionar una Institución, y que posteriormente la lista de lugares se reduzca únicamente a aquellos que estén asociados a dicha institución. Para obtener los Pisos es directo; se *parsea* el ya mencionado atributo `innerFloors`.

3.2. Etapa de escaneo

La primera tarea a resolver corresponde a poder utilizar la *API* RoomPlan para escanear una habitación, y poder generar un archivo que represente el resultado del escaneo. La importancia de esto último radica en la posibilidad de poder generar un resultado almacenable y del que posteriormente se pueda extraer información. El formato en el que se encuentre este

resultado no es especialmente relevante aún, puesto que se definirá en detalle más adelante en el *pipeline* según sean los requisitos de las herramientas por utilizar y según sea la cantidad de información que se espera transmitir.

3.2.1. Utilización de RoomPlan

A modo de tutorial, *Apple* provee de un archivo con los códigos de un proyecto de ejemplo de uso de la *API* de *RoomPlan*[1]. Este proyecto incluye principalmente 4 archivos de códigos, y 3 archivos de diseño de interfaces gráficos. Este último tipo de archivos es un formato que el *IDE* de desarrollo de *Swift* soporta como alternativa para visualizar y definir interfaces gráficas de la aplicación.

El proyecto de ejemplo se puede compilar y abrir como una aplicación sólo conectando un dispositivo al computador. Una vez ejecutada, la única opción disponible es la de realizar un escaneo. La vista en la que el escaneo se realiza, la aplicación ofrece una serie de sugerencias en pantalla para solicitar al usuario una acción específica con la cámara (acercarse, alejarse, enfocar cierto punto, etc). Mientras se realiza el escaneo, en la pantalla se visualiza una miniatura de lo que se ha digitalizado hasta el momento. Una vez que se da por finalizada esta parte, la siguiente vista muestra el resultado final y ofrece la opción de descartar los cambios o exportarlos por medio de alguna aplicación que soporte servicio de mensajería. El formato en el que por defecto se exportan los resultados es el formato *.USDZ*.

Para efectos del presente trabajo, se decide utilizar este proyecto como base. Está disponible para uso libre en la web de *Apple* y ya tiene incorporada la funcionalidad de escaneo. Se descartará la opción de exportación de la forma en que está en el ejemplo, y se presentarán las opciones de "Editar (resultados)" y "Guardar en Local", que permitirán las acciones de transicionar a la Vista de Edición, y guardar en el dispositivo con un nombre personalizado respectivamente. Finalmente, la vista resultante al concluir el proceso de escaneo es la presentada en la figura 3.1.

[1]Son los archivos con los que se construyó el ejemplo de uso de la *API* presentado en la *WWDC 2022* [4]

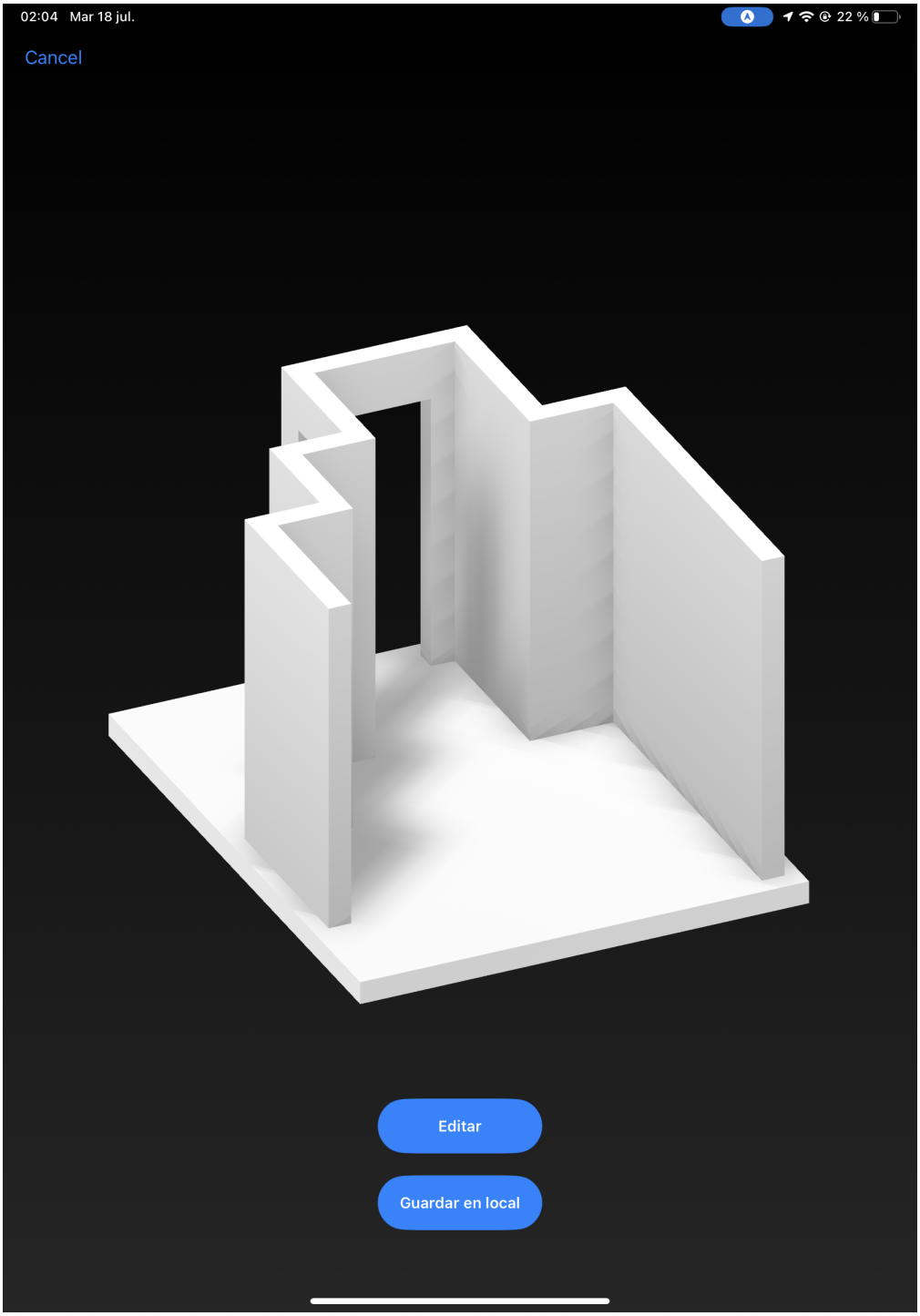

Figura 3.1: Presentación del modelo obtenido

3.2.2. Formatos de archivos utilizados

La primera dificultad a resolver corresponde a definir el formato en el que se quiere exportar la información recolectada. El formato por defecto (*.USDZ*) es un formato con poco soporte, salvo por *software* especializado para animación en 3D. Así, se vuelve necesario

evaluar otra alternativa de exportación, siendo *GeoJSON* el formato escogido inicialmente. Los mapas basados en *OpenStreetMap* permiten agregar capas de datos en formato *GeoJSON* sobre los *tiles* ya existentes con relativa simpleza, contando con una amplia documentación al respecto.

La representación del escaneo completo se almacena en una instancia de la estructura `CapturedRoom` (Ver ejemplo en anexo A.2). Para poder construir un *GeoJSON*, será necesario identificar en dicha instancia los puntos que definen los extremos de las figuras de las superficies de la escena escaneada, extraer los valores y agregarlos al archivo que se exportará. `CapturedRoom` indica la posición del centro de la superficie directamente como propiedad, pero los valores de traslación, rotación y escalamiento se deben inferir de la propiedad `transform`; una matriz de dimensión 4×4 con toda la información geométrica que define a cada cuerpo.

3.3. Etapa de edición

Se decide agregar una nueva vista que llamaremos "Etapa de Edición". Conecta a la vista de *RoomPlan* con el proceso de exportación y permita revisar y modificar la escena antes de transferir los datos a la herramienta de visualización de mapas. El objetivo de esta etapa es aprovechar la posibilidad que ofrece *RealityKit* para recrear escenas tridimensionales y darle la oportunidad al usuario de corroborar que lo digitalizado es consistente con la habitación que escanearon o de ajustar las dimensiones de los elementos de la escena -y hasta remover algunos- en caso contrario.

El primer desafío a resolver consiste en lograr transferir una representación de la escena de la primera vista hacia esta nueva vista de edición. RoomPlan no considera herramientas de edición o manipulación del resultado del escaneo por lo que este objeto deberá transformarse a algún tipo de dato o estructura compatible con las librerías de edición relevantes de *Apple*. Para esto se utilizará la clase "JSONEncoder()". Con esta clase es posible codificar la escena en una instancia de la clase "NSData", que corresponde a un *wrapper* de un *byte buffer* transferible y decodificable.

El siguiente desafío tiene que ver con decidir cuál *framework* de *Swift* utilizar para volver a recrear la escena. *Apple* cuenta con los *frameworks RealityKit*, *MetalKit* y *SceneKit*.

RealityKit es el más reciente de los 3, siendo liberado recién en el año 2019, por lo que está en crecimiento y posee características interesantes para la experiencia de desarrollo, con un gran potencial de rendimiento [7]. *Metal* es un *framework* de bajo nivel -que es incluso la base de *RealityKit* y *SceneKit* entre otros-, con herramientas aún más avanzadas para optimizar el rendimiento de aplicaciones avanzadas en 3D. *MetalKit* es un módulo disponible para *Swift* con el que es posible construir aplicaciones de *Metal* con mayor rapidez y facilidad [5].

SceneKit en cambio, es un *framework* de alto nivel, de mayor antigüedad que los otros dos mencionados (fue liberado en 2012), compatible con *ARKit*, y además compatible con varios formatos (incluído *.USDZ*) [10]. Se decide utilizar este *framework* para la recreación de la digitalización por dos razones: sólo se busca construir una escena donde las figuras

serán mayoritariamente hexaedros, siendo innecesario recurrir a herramientas de bajo nivel para esto, y la antigüedad de *SceneKit* sugiere que la documentación esté depurada por más tiempo, y que la cantidad de preguntas-respuestas disponibles en foros y comunidades *online* de programadores sea mayor.

El resultado de la vista se puede observar en los Figuras 3.2 y 3.3. Además, se incluye una captura de la ventana que despliega la descripción de la escena visualizada en la Figura 3.4.

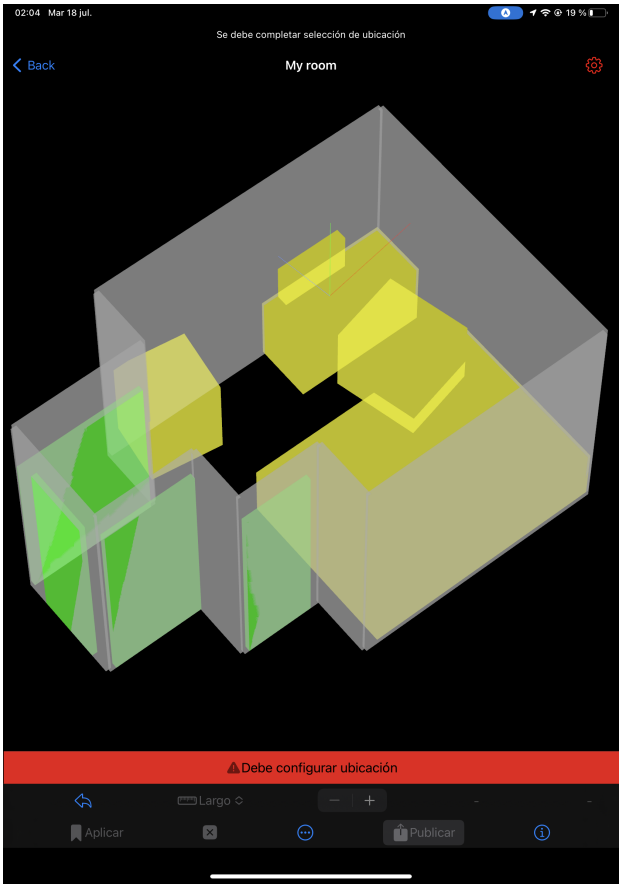

Figura 3.2: Presentación de alerta para configurar ubicación

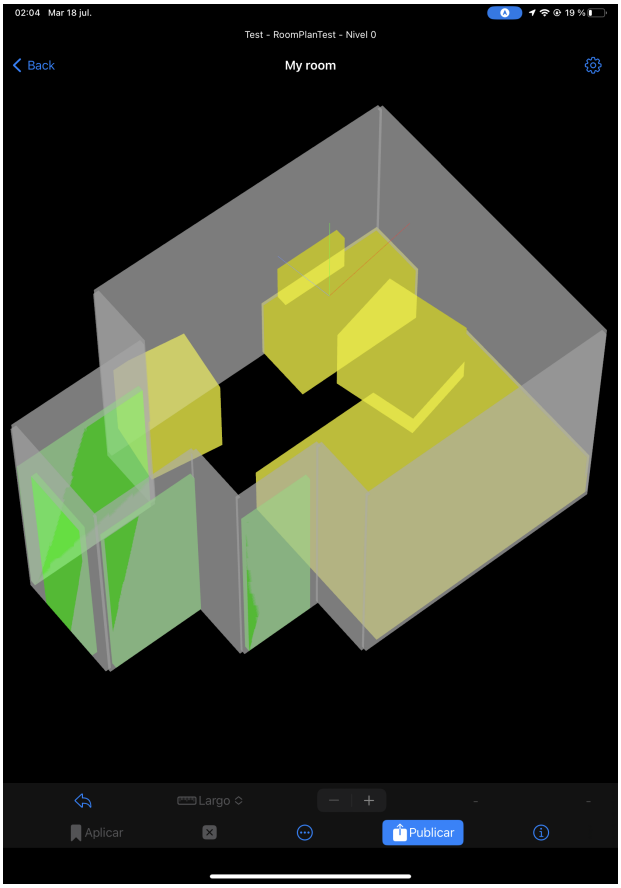

Figura 3.3: Vista de edición después de terminar configuraciones

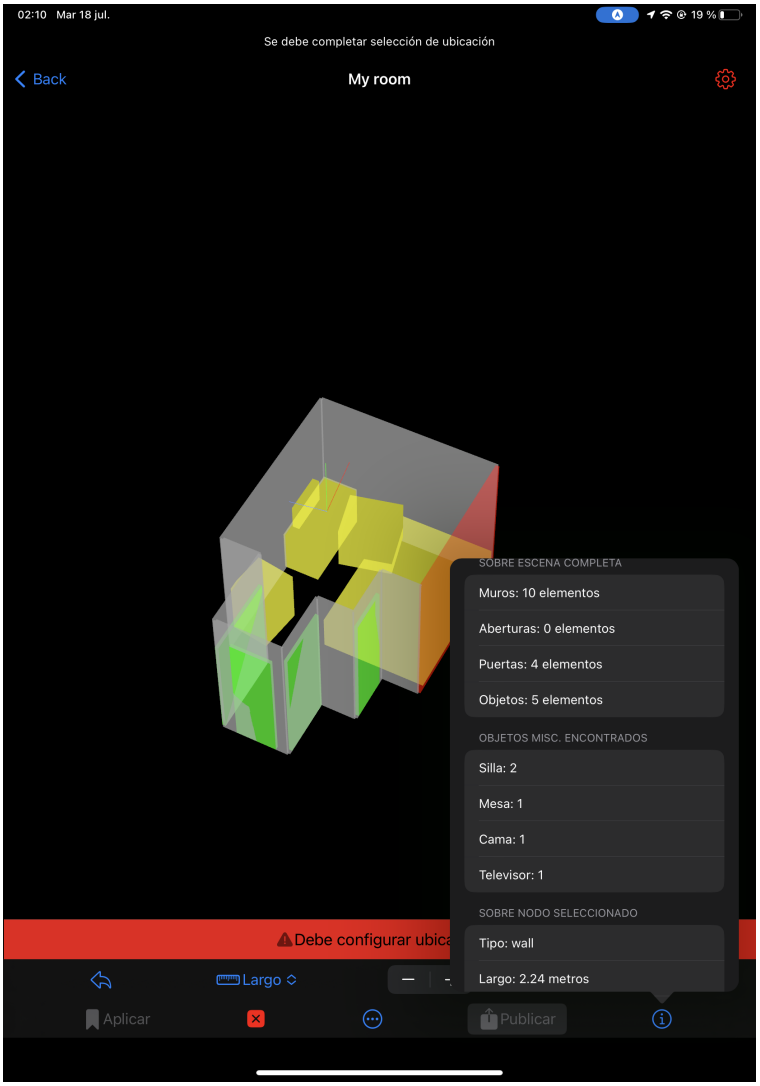

Figura 3.4: *Popover* con la descripción de los componentes de la escena, más una descripción cuantitativa de un nodo seleccionado

3.3.1. Acciones deseadas en la vista

Selección

El objetivo de esta vista es poder observar la digitalización de la habitación escaneada y realizar ajustes rápidos a los objetos de la escena que presenten discordancias evidentes al ojo del operador de la aplicación. Así, es imprescindible que al recrear la escena se puedan reconocer y trabajar de forma independiente los elementos fundamentales de la misma entre ellos. Por esta razón, se espera que si el usuario hace click en alguna parte de la escena, el elemento de ella que fue presionado debe destacarse de alguna forma, dando a entender que cualquier otra acción será aplicada única y exclusivamente en dicho elemento.

Visualización de información

Al seleccionar un nodo, la vista debe mostrar las medidas de las dimensiones de este. Así el usuario puede apoyarse de esta información para tomar cualquier otra acción sobre la escena recreada y comparar el efecto de cualquier acción de edición que se haya aplicado con la versión original de la misma.

Además, existe otro botón que despliega una ventana tipo *popover* que muestra información general sobre la escena y el nodo seleccionado, si es que hay. Allí se indica la cantidad de elementos presentes de cada categoría, las subcategorías encontradas para los elementos de la categoría principal *objects* y las dimensiones y categoría específicas del nodo seleccionado, si es que hay.

Filtrado de nodos

Siguiendo con la idea de distinguir elementos dentro de escena, se notará también que entre esos mismos elementos existirán distintos niveles de relevancia para cada uno. Se considerarán siempre fundamentales los muros de la habitación, pues son ellos quienes definirán el plano que se espera generar. Después de ellos, por ejemplo, podría considerarse que lo que le sigue en importancia son puertas y aberturas, pues indican entradas y salidas que también puede ser deseable señalar en los mapas. Por último, se encuentran las ventanas y objetos misceláneos. Dentro de estos últimos existen sub-categorías que, bajo algunas circunstancias, puede ser deseable agregarlas al mapa del lugar (como mesas grandes) pues inciden en la disposición de la habitación.

Existe más de un alternativa para sortear el problema de discriminar objetos por relevancia: se puede ignorar directamente dicha diferencia y construir todos y cada uno de los elementos escaneados por *RoomPlan*, se puede consultar al usuario qué categorías de objetos prefiere visualizar o directamente se pueden ignorar los objetos de ciertas categorías durante la construcción de la vista según lo que el desarrollador estime que es menos relevante. La primera opción le otorga el poder al usuario para definir *in situ* qué elementos quedarían digitalizados en el plano, pero podría considerarse molesto en casos donde en el espacio escaneado existen muchos muebles poco relevantes, como podría ser el caso de las sillas en una oficina. Por último, en cuanto al trabajo extra de programación que implicaría, se tendrían que agregar nuevas funciones para digitalizar los elementos no fundamentales, pues no tienen una geometría virtualmente plana como es el caso de las superficies.

La segunda opción también implica el mismo trabajo de programación en cuanto a las geometrías que habría que considerar digitalizar y también una componente de interfaz donde el usuario pueda escoger qué categorías espera traspasar al mapa final. Con el fin de proveer de información que ayude al usuario a entender el impacto de elegir estas categorías, se puede indicar también información cuantitativa sobre la habitación escaneada dentro de esta componente visual como la cantidad de elementos de cada clase. Esta opción pareciera ser la más completa y cómoda, por lo que se espera que por lo menos en el final del trabajo se pueda depurar la vista para alcanzar este nivel de usabilidad.

Finalmente, la última opción ofrece menos libertad para el usuario al no darle opción de

elegir qué categorías considerar. Sin embargo, durante las etapas preliminares de desarrollo esta simpleza ofrece que el trabajo no se estanque en funcionalidades poco esenciales dentro de lo que se espera de un desempeño correcto de la aplicación y que así se puedan entregar prototipos con mayor rapidez, que preferentemente incluyan más funcionalidades sencillas, en lugar de pocas funcionalidades de mayor complejidad. Esto es también útil cuando se trata de analizar globalmente la aplicación, pues con más funcionalidades hay más oportunidades de observar interacciones posibles entre ellas y simular flujos de uso más completos, lo que se transmite en más retroalimentación temprana, la que será necesaria a la hora de depurar las primeras versiones del producto.

Teniendo en cuenta que justamente es la comunicación fluida entre el programador y el cliente y la retroalimentación temprana que este último puede entregar lo que se espera alcanzar con una metodología ágil de trabajo, se decide que es esta última opción la que se implementará en primera instancia. A medida que el desarrollador implemente otras funciones o que usuarios muestren insatisfacción con las limitaciones actuales, se analizará tomar otras decisiones.

Eliminación de nodos

En cualquier caso, el usuario debiese tener como grado mínimo de libertad de trabajo el poder decidir que algún elemento específico -independiente de su categoría- no sea transmitido al plano del mapa. Para esto, se implementará un botón de borrado de nodos cuya acción debiese ser posible de deshacer mediante otra opción, en caso de que el usuario de arrepienta. Entonces, el botón sólo se activará al momento de seleccionar un nodo. Así mismo, luego de haber realizado este cambio, será posible para el usuario confirmar el cambio realizado, eliminando la posibilidad de deshacer la acción y así no transferir el nodo al plano por generar.

Ajuste de dimensiones de nodos

Dada la naturaleza de *RoomPlan* (está en su primera versión liberada al público al momento de realizar este proyecto, es fuertemente dependiente de factores del ambiente de escaneo como la iluminación del mismo o la cantidad de objetos menores que se interponen entre el objeto principal y el sensor), es esperable que los resultados sean por lo menos perfectibles.

En este caso se refiere a uno de los dos acercamientos que se contemplarán para ajustar en detalle los resultados obtenidos. La posibilidad de visualizar la habitación como una escena en tres dimensiones, sugiere que sería útil para el usuario aprovechar esta característica a la hora de definir el nivel de satisfacción con respecto a los resultados obtenidos. Es decir, observar la habitación como un conjunto de modelos 3D entrega una perspectiva distinta a la que ofrecerá posteriormente observarla sólo como un plano de dos dimensiones.

A través de *SceneKit*, la librería que permitirá recrear el resultado de RoomPlan, es posible extender o reducir el escalamiento de un nodo de la escena a través de alguna de sus dimensiones. Esta opción nace de la realización de pruebas preliminares de un proyecto de ejemplo que existe de RoomPlan, donde algunos de los objetos escaneados quedaban

cortos de largo o ancho. En lo que refiere a los muros de la habitación, es probable que esta funcionalidad no tenga tanto sentido, ya que salvo en escáneres incompletos, o en murallas menos tradicionales (como algún divisor de ambientes), modificar las dimensiones de una de ellas implica alterar las coordenadas de los vértices que la definen y así desconectarla de las murallas contiguas a ella. Esto último es perjudicial para la generación del polígono del plano.

Esta funcionalidad considera modificar objetos en cualquiera de sus 3 dimensiones, aunque cabe mencionar que en algunos casos su utilización no tiene sentido. Por ejemplo, técnicamente es posible modificar el grosor de los objetos que representan murallas, aún cuando la dimensión del ancho mostrada en la pantalla sólo corresponde a un valor por defecto con el motivo únicamente funcional de evitar que estas murallas luzcan como una lámina. Además, en esta vista es posible modificar la altura de estos elementos, aún cuando la alteración de esos valores no tendrá efecto alguno en los resultados finales del proceso, puesto que es la dimensión que se ignora al pasar a un plano.

Confirmar cambios realizados

El objetivo de esta funcionalidad es proveer al usuario de un medio por el cual pueda deshacer las acciones realizadas. La idea es que el hecho de poder confirmar algún cambio, sugiere que aquellos cambios que no se han confirmado, no son definitivos y por tanto es posible volver a un estado anterior a ellos. Entonces, este requisito implica dos acciones distintas y opuestas: "Descartar cambios" y "Confirmar/Aplicar cambios".

Al cargar esta vista y dibujar los modelos, cada nodo dibujado se guarda como hijo de un nodo raíz del grafo de escena. Todos estos hijos se encuentran en una lista de nodos como propiedad de este nodo raíz. Al mismo tiempo que se crean los nodos a dibujar y se guarda cada uno en la lista de nodos hijos del nodo raíz, se van también agregando a una nueva lista llamada `backUpChildNodes`, que corresponde a una colección de los nodos con sus modificaciones definitivas. Esta segunda lista se utiliza para llevar a cabo la acción "Descartar cambios". Lo que se hace allí es remover todos los nodos hijos del nodo raíz (es decir, eliminar de la escena todos los nodos dibujados) y luego agregar todos los nodos de `backUpChildNodes` como hijos.

La acción "Confirmar cambios" por su parte, hace la misma operación en el otro sentido. Reemplaza el arreglo `backUpChildNodes` por un arreglo de los nodos dibujados en la escena, incluyendo únicamente los nodos que no han sido eliminados y sus modificaciones de escalamiento. Así, para cuando se decida accionar con "Descartar cambios" nuevamente, retomará los elementos desde el estado en el que estas alteraciones ya fueron aplicadas, reafirmando que son irreversibles (desde esta vista).

Renombre del escaneo guardado

Esta opción sólo está disponible para modelos que han sido guardados en local previamente; sólo tiene sentido "cambiar el nombre" cuando un nombre ya ha sido asignado. Refiere

a cambiar el nombre de referencia del archivo que genera el modelo que está cargado en la vista en dicho momento.

Ver mapa del lugar

Refiere a poder ver cuál es el estado actual del mapa. Es importante que el usuario de la aplicación pueda observar qué planos ya existen actualmente en el mapa asociado al lugar seleccionado. Esto le sirve al usuario para saber qué es lo que ya está dibujado en el mapa del lugar, y así por ejemplo, tener precaución en lo que se va a dibujar y en qué coordenadas se centrará.

Exportar escena actual

Esta acción corresponde al objetivo principal de este trabajo: convertir el modelo generado en el escaneo en el dibujo del lugar como plano en el visualizador de mapas de Lazarillo.

Se hablará en el Capítulo 4 sobre cómo se resuelve este problema.

3.3.2. Decisiones de diseño y desarrollo

Apple provee de una amplia serie de elementos de interfaz por defecto que facilita las etapas iniciales de desarrollo al ahorrar tiempo en definir una interfaz personalizada. Posee configuraciones base de estilos, que se ensamblan con relativa facilidad -estéticamente hablando-, y que compatibilizan con cualquiera que sea el tema (claro u oscuro) del dispositivo.

Dentro de las decisiones que sí se tomaron, se tiene que se seleccionó un estilo que destaca más para aquellos botones que realizan las acciones más invasivas ("Aplicar cambios", "Borrar elementos", "Publicar"), mientras que para el resto, su apariencia es menos llamativa.

Otras decisiones han sido:

- Se decide que el fondo sobre el que se dibuja el modelo escaneado sea de color negro independiente de si el tema es oscuro o claro.
- Se visibiliza el eje de coordenadas XYZ sobre el que se orientan los nodos.
- Se simplifica el nombre de los botones con el fin de que quepan de manera cómoda dentro de la resolución de un *iPhone* (considerando que el desarrollo de realizó en un *iPad*, y hubo que evaluar cómo se veía en otros dispositivos).
- Se despliega un mensaje de advertencia que indique al usuario que debe configurar la ubicación del mapa que se quiere generar.

3.4. Etapa de publicación

Esta sección refiere a la etapa de carga de los elementos obtenidos por *RoomPlan* hacia el Editor de Mapas de Lazarillo y a las decisiones de diseño consideradas para la vista encargada.

3.4.1. Situaciones a considerar

Manejo de solicitudes

Uno de los problemas que se tuvo que resolver fue el manejo de las solicitudes de tipo POST al Editor de Mapas.

Sucede que, en primera instancia, para agregar elementos a un lugar en el Editor de Mapas, el *backend* de este proyecto sólo contempla un único *endpoint* de "creación". Y este *endpoint* sólo permite la creación de un único elemento a la vez. Para el caso de esta aplicación, eso significa tener que enviar una *request* por cada objeto, entrada y colección de muros[2] que se intenten publicar.

Uno de las inconveniencias de esto es que se realizarán varias conexiones con el *backend*, donde el éxito o error de ellas son independientes entre sí. Cada conexión, a nivel de código, corresponde a la ejecución de una función asíncrona que realiza la solicitud y se queda pendiente de esperar la respuesta del servidor. Si alguna solicitud fallase, las solicitudes previas o incluso posteriores podrían ser publicadas de igual manera y entonces en el editor de mapas aparecerían sólo algunos de los elementos. Puede darse el caso aquí en que hay dos elementos que necesitan del otro para entender qué significa cada uno (como el caso de una puerta con una pared) y si sólo se publicase uno de ellos, sería difícil entender el espacio que se quiere reflejar.

Retroalimentación para el usuario

Otro problema, esta vez relacionado a la experiencia de usuario, tiene que ver con el comportamiento definido para dar retroalimentación al usuario del éxito o fracaso de la acción de "Publicar". Se debe tomar en cuenta que, al ser la acción más importante y a la vez la más destructiva de la aplicación, es fundamental indicarle al usuario si la acción tuvo éxito o no. Pero como cada acción es independiente con respecto a otra acción, asíncrona, dependiente del *status* del servidor del *back* y de la conexión a internet, entonces la condición para desplegar un mensaje de éxito o fracaso no es tan directa como sólo esperar que se ejecute un trozo de código, ni tampoco es amigable desplegar una alerta por cada respuesta (sería forzar al usuario a cerrar una cantidad absurda de alertas que aparecerían prácticamente al mismo tiempo).

[2]Se habla de "colección de muros" porque en teoría, se espera que se defina un único polígono a partir de todos los muros que componen el lugar; entonces el elemento de la solicitud que se publicaría sería ese polígono. En la práctica, es posible que la aplicación no logre detectar el espacio interior como un único espacio, y por tanto se crearían tantos polígonos como sean necesarios para utilizar todos los muros detectados.

Esto se abordará utilizando dos estructuras de tipo Cola que almacenan las *requests* que se quieren enviar y una variable nombrada `currentRequest` para distinguir la solicitud en curso. La primera de ellas se denomina `pendingQueue` y contiene todas las *requests* pendientes de realizar. La segunda de ellas nombra `doneQueue`, se inicializa como vacía y se utilizará para agregar cada solicitud que obtiene una respuesta desde el *backend* (exitosa o no). Entonces, se considerará que la acción de publicar ha concluído una vez que la Cola `pendingQueue` se encuentre vacía y el valor de `currentRequest` nulo. Una vez llegado a este estado, se despliega la alerta de que la publicación se ha realizado.

Para informar al usuario del éxito o error de cada solicitud individualmente, cada una se listará en pantalla acompañada de un elemento gráficos que indique el estado de ella. Además, se mostrará una barra de carga de la publicación en general que ayude al usuario a observar el progreso general de toda la escena. Un ejemplo de esto se ilustra en la Figura 3.5

3.4.2. Fin de la etapa

Se considera que la acción de publicación de la escena ha finalizado una vez que para cada objeto, entrada y polígono existente en la escena, se ha enviado una solicitud y se ha recibido una respuesta a esa solicitud. Cuando la última respuesta se recibe, se despliega una alerta para el usuario que indica que "Algo ha salido mal" en caso de que alguna solicitud no haya sido exitosa. En caso contrario, la alerta indica que todo salió bien,e invita al usuario a abrir el Editor de Mapas para observar el resultado de la publicación. De todas formas, si el usuario cierra la alerta, se agrega un botón a la vista para abrir el Editor, como se observa en la Figura 3.6).

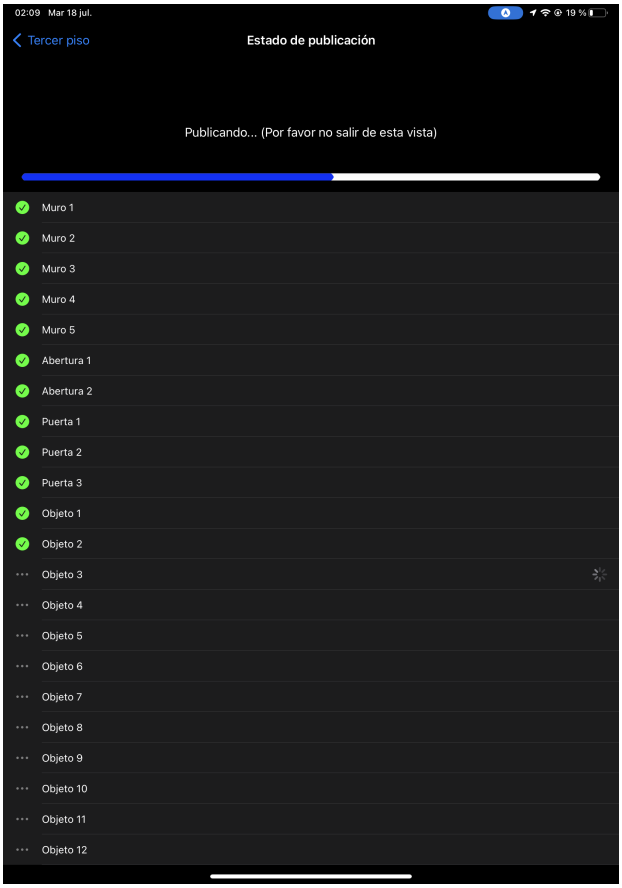

Figura 3.5: Lista de solicitudes con su respectivo estado en vista de publicación

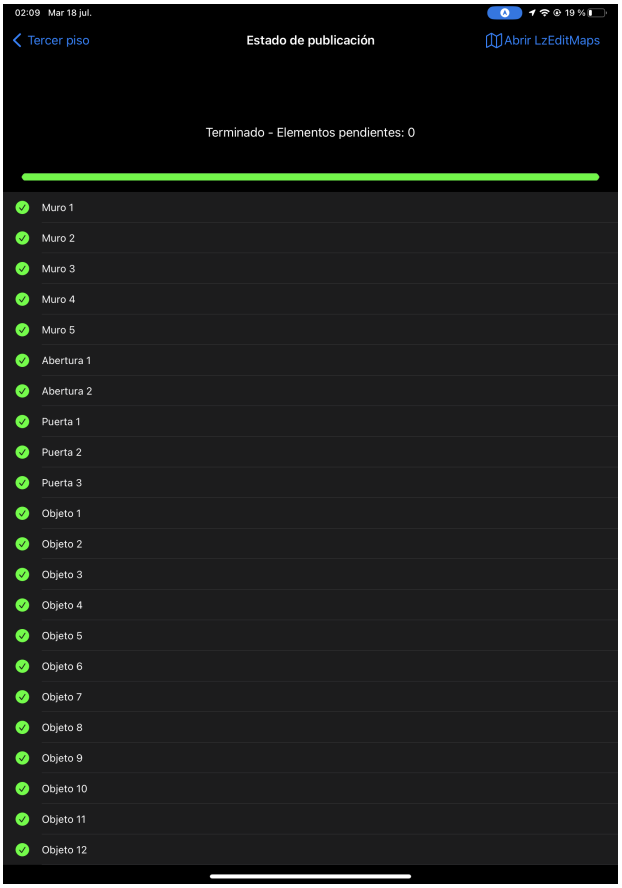

Figura 3.6: Vista de publicación posterior al envío exitoso de todos los elementos

3.5. Otras vistas

Durante todo el periodo de desarrollo de este trabajo, son varias las posibles funcionalidades que han sido evaluadas para ser agregadas a la aplicación. La decisión y orden de su implementación ha radicado en si son consideradas necesarias, deseables o irrelevantes para el usuario.

Por ejemplo, la vista de edición es la primera nueva vista que se agregó luego de tener completado lo que refiere al uso de *RoomPlan*. Esto se hizo porque era importante encontrar una manera de reproducir el modelo escaneado, comprender qué tanto nivel de edición sería posible alcanzar y poder definir una dirección de trabajo alternativa junto con la empresa en caso de encontrar limitaciones para llevar a cabo el plan original. Con esto, se tuvo que elegir cuáles serían las acciones que estarían disponibles en esta vista.

3.5.1. Vista de Login

Es necesario que sólo usuarios validados por Lazarillo puedan hacer uso de la aplicación, dado que esta conecta con el editor de mapas. El editor es el encargado de crear los mapas utilizados tanto en portal como en LazarilloAPP, por lo que su mal uso tiene consecuencias directas en la experiencia de cada uno de los clientes de la empresa.

Considerando que se pretende que los usuarios válidos sean los mismos usuarios que utilizan las otras aplicaciones de Lazarillo, se debe utilizar el servicio de autenticación de *Firebase*. La librería llamada *FirebaseAuth*, es de uso rápido y provee métodos que permiten personalizar la experiencia de autenticación en distintos niveles de profundidad. Si bien en primera instancia se creó un formulario desde 0, que sólo lee y *parsea* lo que el usuario ingresa en los campos de texto para nombre de usuario y contraseña, este medio presentaba el problema de que no permitía el ingreso mediante la autenticación de otros servicios como *Google*, *Facebook* o *Apple*.

Luego de revisar cómo se manejaba el acceso para las cuentas de esos servicios, se observa que para ellas no está definido un usuario ni contraseña. Dado que las direcciones de correo de la empresa Lazarillo son cuentas de *Gmail*, y que la gran mayoría utiliza tal cuenta para autenticarse en las aplicaciones de Lazarillo, se decide utilizar la extensión *FirebaseAuthUI*. Esta extensión permite agregar un elemento de interfaz que implementa los botones para autenticarse por medio de servicios externos y que incluye tanto el diseño de cada botón para cada servicio como la lógica de autenticación que estos utilizan.

Por último, se agrega el título de la aplicación en el centro de la pantalla, siendo este el único otro elemento que se despliega en esta vista (Ver resultado de la vista en anexo 3.7).

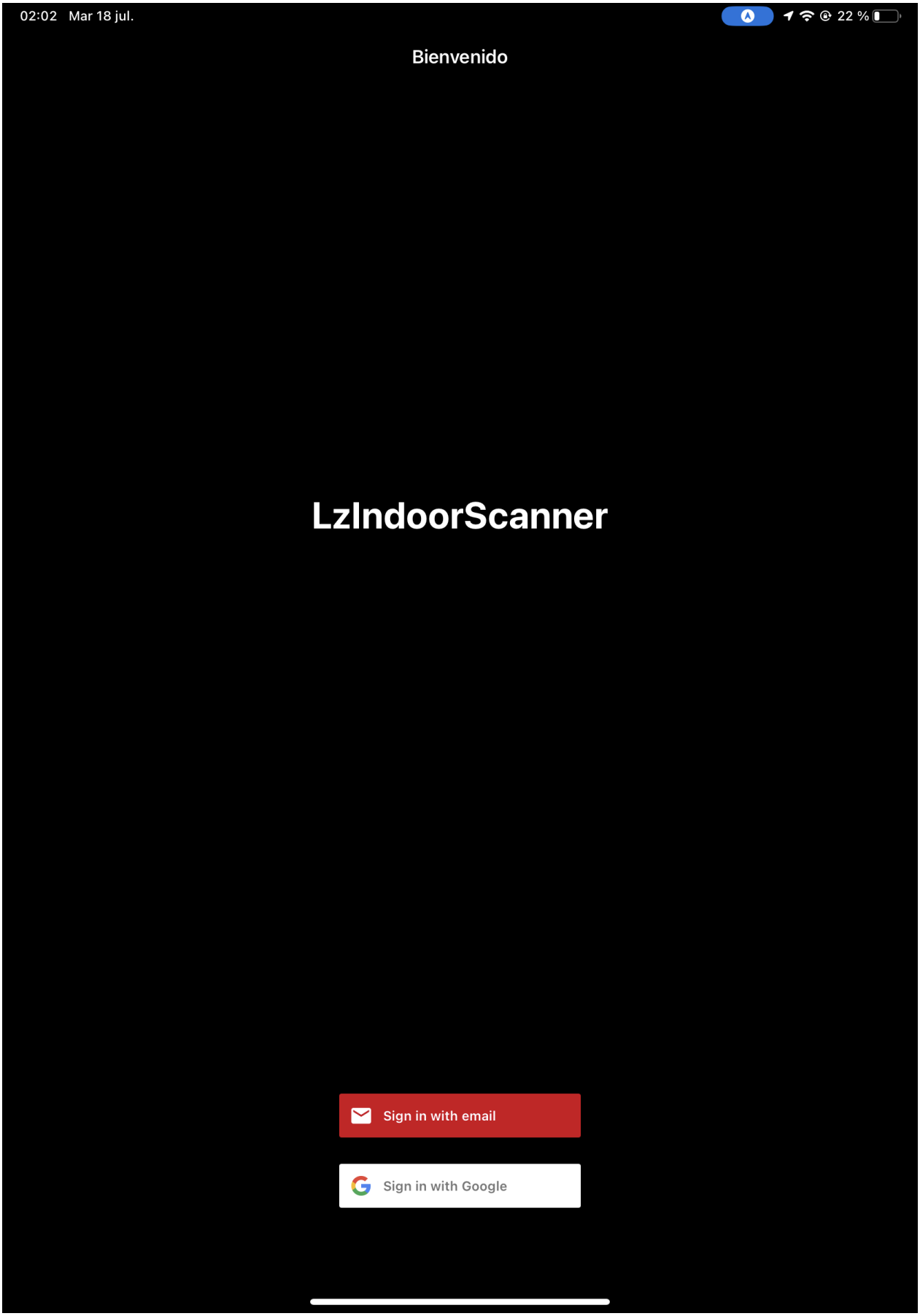

Figura 3.7: Vista de *Login*

3.5.2. Menú principal

Esta vista nace con la creación del proyecto base que dio inicio al trabajo actual, y refiere a la vista desde donde el usuario -ya autenticado- puede elegir entre iniciar un escaneo nuevo, cargar y editar un escaneo ya realizado, o eliminar un escaneo cargado.

Con la liberación de *RoomPlan*, *Apple* liberó un proyecto de *template* que permite conocer cómo se podría utilizar el *framework*. Es a partir de ese *template* que se ha construido la aplicación, aunque son pocos los elementos que permanecen en el estado actual. De aquel proyecto demo, se ha conservado:

- La transición hacia la vista de escaneo.
- El comportamiento al dar por terminado el escaneo y poner en primer plano la miniatura del modelo generado. Eso sí, las acciones disponibles en este punto ya son diferentes.
- La distribución base de elementos en la vista inicial. Antes se proveía de instrucciones de uso al usuario, y disponía de un botón para iniciar el escaneo. Ahora, se agregaron más botones para otras acciones, se tradujeron las instrucciones al español y se agregó más información relevante del uso de la aplicación (Ver resultado de la vista en anexo 3.8).

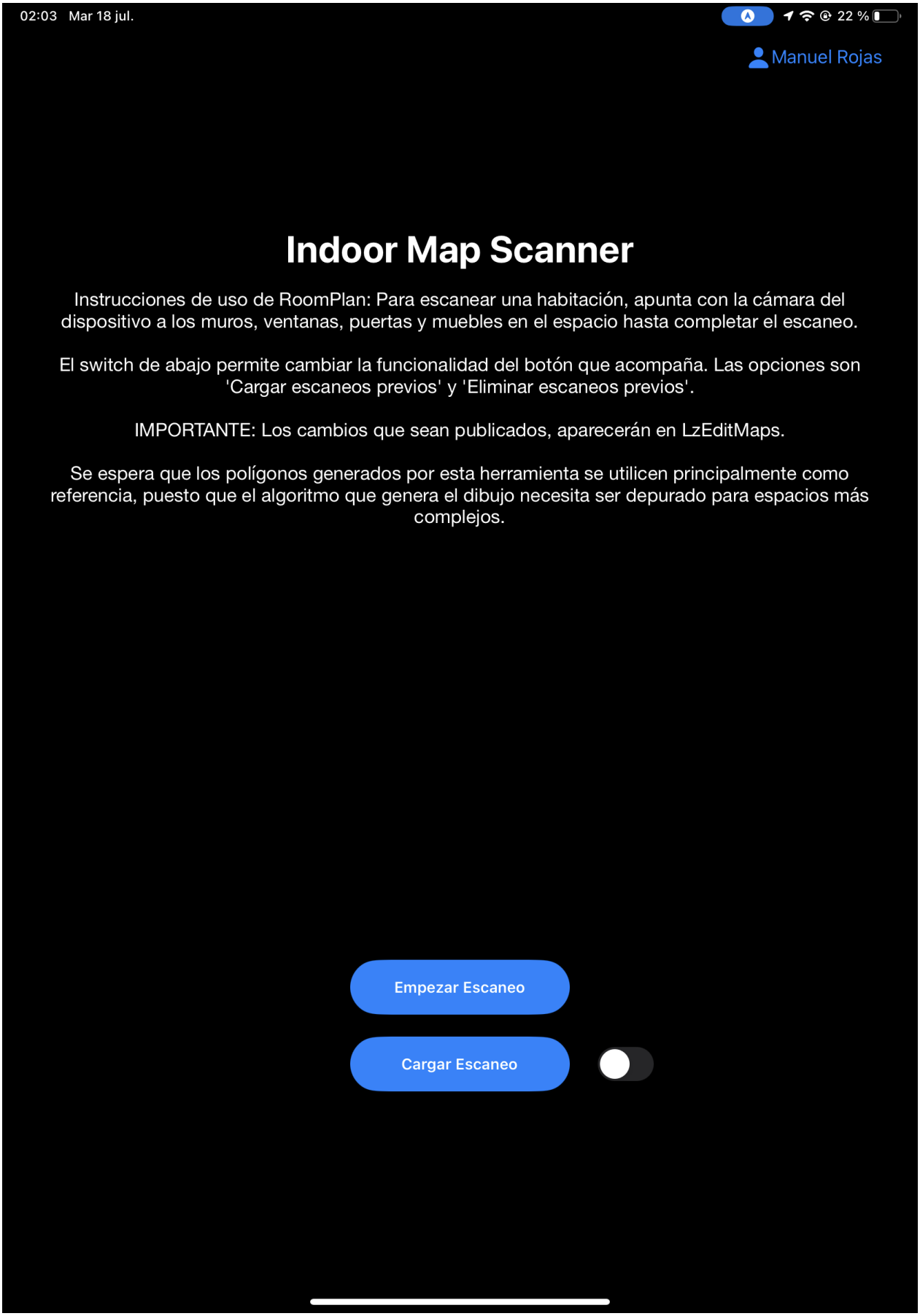

Figura 3.8: Vista de Menú principal

3.5.3. Vista de configuraciones

Para todo el resto de acciones que correspondían a una categoría diferente a las acciones de la vista de edición (es decir, que no se consideran exactamente como un acto de editar la escena) o que no lograron formar parte de la interfaz de la vista de edición por temas de espacio en la pantalla, se decide crear esta vista de configuraciones generales.

Por ahora, contempla una sección fundamental para la aplicación, y otra que es un "deseable" para la experiencia del usuario. Se pretende también que las nuevas funcionalidades y opciones que se desarrollen se agreguen a esta vista.

Filtro de categorías

La primera funcionalidad desarrollada corresponde a filtrar los elementos visibles en la escena según la categoría a la que fueron asignados por *RoomPlan*. Esto es particularmente útil para espacios que cuentan con una gran cantidad de mobiliario y donde sólo se quiere obtener la distribución del espacio a través de sus muros y entradas. Entonces, en lugar de eliminar cada silla, mesa o pantalla encontrada, se podría de-seleccionar la categoría "Objetos" y así ocultarlos todos de una vez.

Esto se logró mediante el uso de *radio buttons*, incluyendo una opción para seleccionar o de-seleccionar todos las categorías. Eso sí, se decidió dejar permanentemente activa la selección de la categoría "Muros", porque se considera como fundamental y básica para el objetivo principal del trabajo; exportar sólo objetos y entradas sin una referencia al espacio escaneado no tiene sentido. Asi mismo, ni siquiera la opción de "Todas las categorías" modifica sobre la selección de "Muros". Al instalar la aplicación por primera vez, la opción por defecto es que todo esté seleccionado. El usuario no tendría necesidad de interactuar con estas funciones.

Configuración del lugar

El segundo uso que tiene esta vista consiste en la asignación de un Lugar para el mapa que se desea generar. El usuario tendrá que seleccionar una Institución, Lugar, Piso, y ubicación geográfica aproximada del espacio escaneado. Todos ellos son de carácter obligatorio. Cuando no se ha seleccionado alguno de estos valores, se decide destacar con rojo el color de la celda de la entidad que cuya selección está pendiente.

Aquí se hace uso de una sub-vista donde se listan las entidades existentes para cada tipo de dato, y se agrega un campo de texto donde se filtra la lista según los valores cuyo nombre contiene lo ingresado por el usuario. Dado que la selección de 2 de las 3 entidades depende de la selección de otra, la selección de ellas se activa sólo cuando la entidad más independiente ya ha sido configurada. Entonces, el orden de acción del usuario y el comportamiento de esta sección de configuraciones es el siguiente:

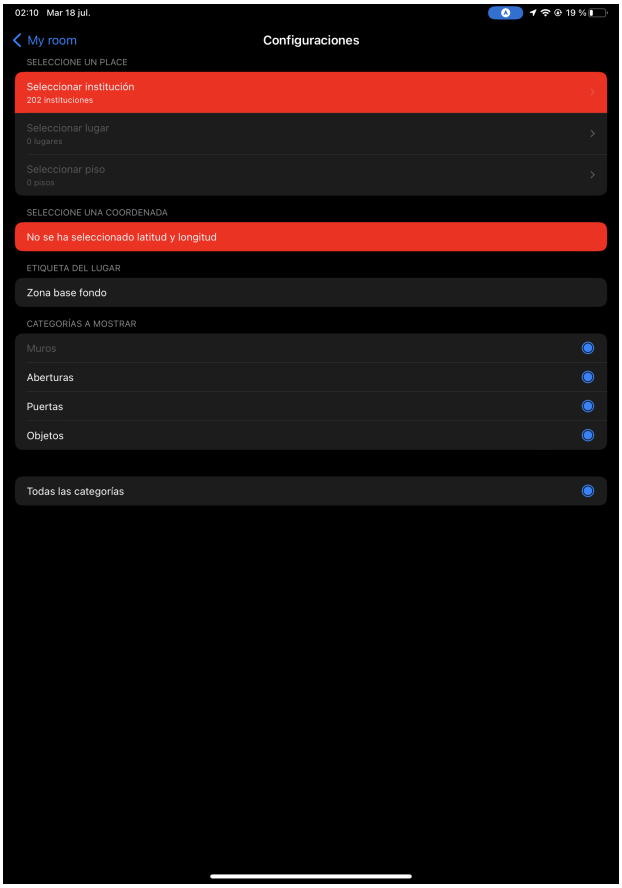

Figura 3.9: Estado inicial de la vista de configuraciones, con selecciones pendientes por realizar

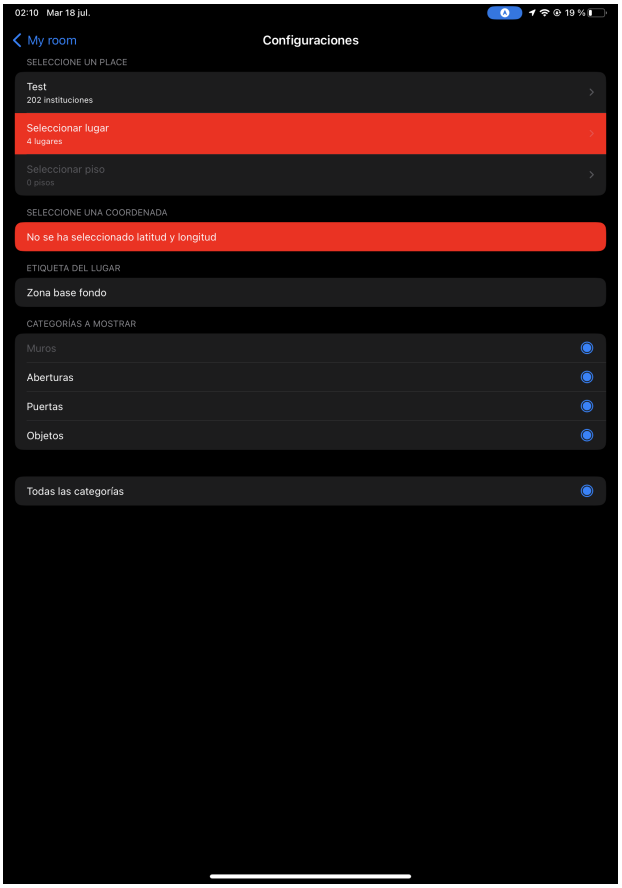

Figura 3.10: Con Institución seleccionada, se indica que se debe continuar a la selección de un Lugar

1. Se destaca con rojo la celda de selección de Institución, indicando que el usuario debe escoger una entidad de este tipo para continuar (Ver Figura 3.9).
2. Al elegir un valor, la celda de Institución deja de estar en rojo, y se destaca en ahora la celda de selección de Lugar (Ver Figura 3.10).
3. Al dar click en la celda de lugar, se muestran sólo los lugares que tienen como Institución asociada a la Institución seleccionada en el paso anterior. Además, sólo se muestran lugares que contengan al menos 1 Piso interior.
4. Al seleccionar un Lugar, el fondo de la celda de Lugar deja de estar en rojo, para destacar en el mismo color ahora la celda de selección de Piso. Además, se hace disponible para la aplicación la ubicación geográfica del Lugar seleccionado, que puede ser usada más adelante en la selección de ubicación (Ver Figura 3.11).
5. Al dar click en la celda de piso, se muestra la lista de Pisos que se extraen del Lugar seleccionado antes. Una vez hecho esto la celda deja de estar en rojo.

Para completar esta configuración se debe finalizar con la selección de la ubicación geográfica. El usuario tiene 3 alternativas para esto:

1. Ingresar manualmente una latitud y longitud a utilizar. Está siempre disponible como opción.
2. Utilizar ubicación del dispositivo según GPS. Está disponible sólo si es que el usuario permitió que la aplicación tenga acceso a la ubicación.
3. Utilizar ubicación del `Place` seleccionado, mencionada en los pasos anteriores. Refiere a la ubicación geográfica de la dirección asignada en la base de datos al `Place`. Aparecerá como disponible sólo después de seleccionar `Place`.

La selección de estas opciones se presentan como aparecen en la Figura 3.12

Una vez hecho esto, la celda deja de destacarse en rojo y el usuario habría completado las configuraciones básicas para poder acceder a la acción de "Publicar" en el Editor de Mapas. La vista final quedaría como en la Figura 3.13.

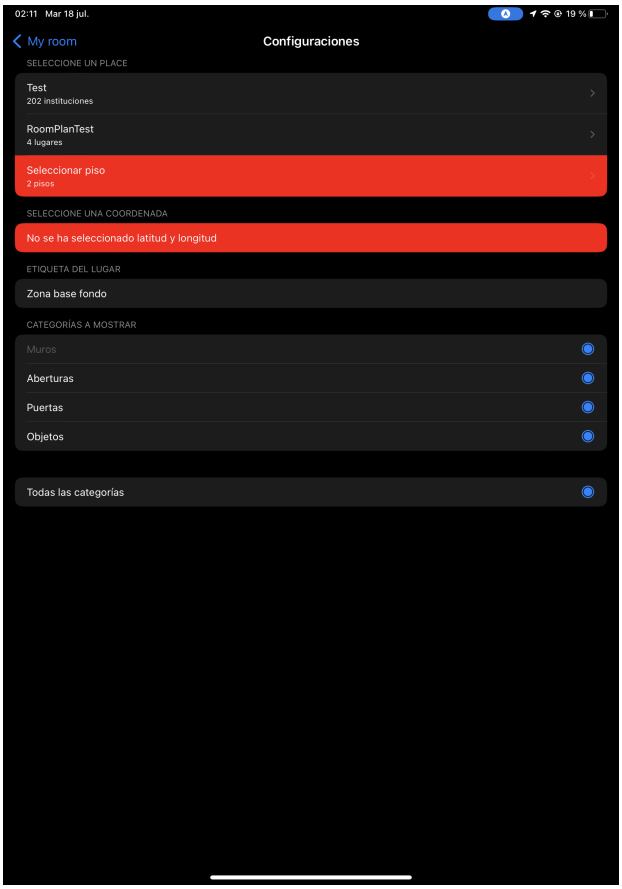

Figura 3.11: Configuraciones donde el piso y la ubicación aún están pendientes

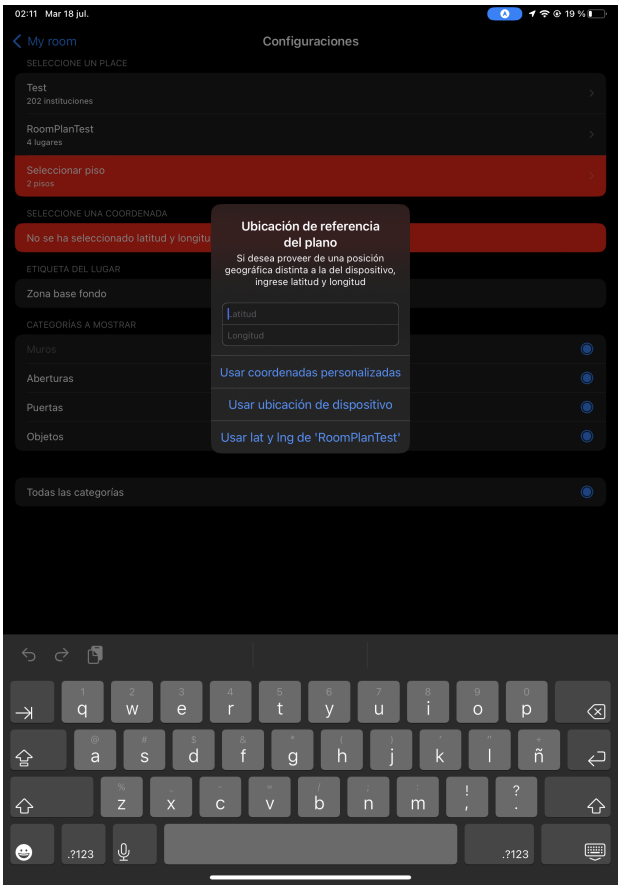

Figura 3.12: Opciones para selección de latitud y longitud

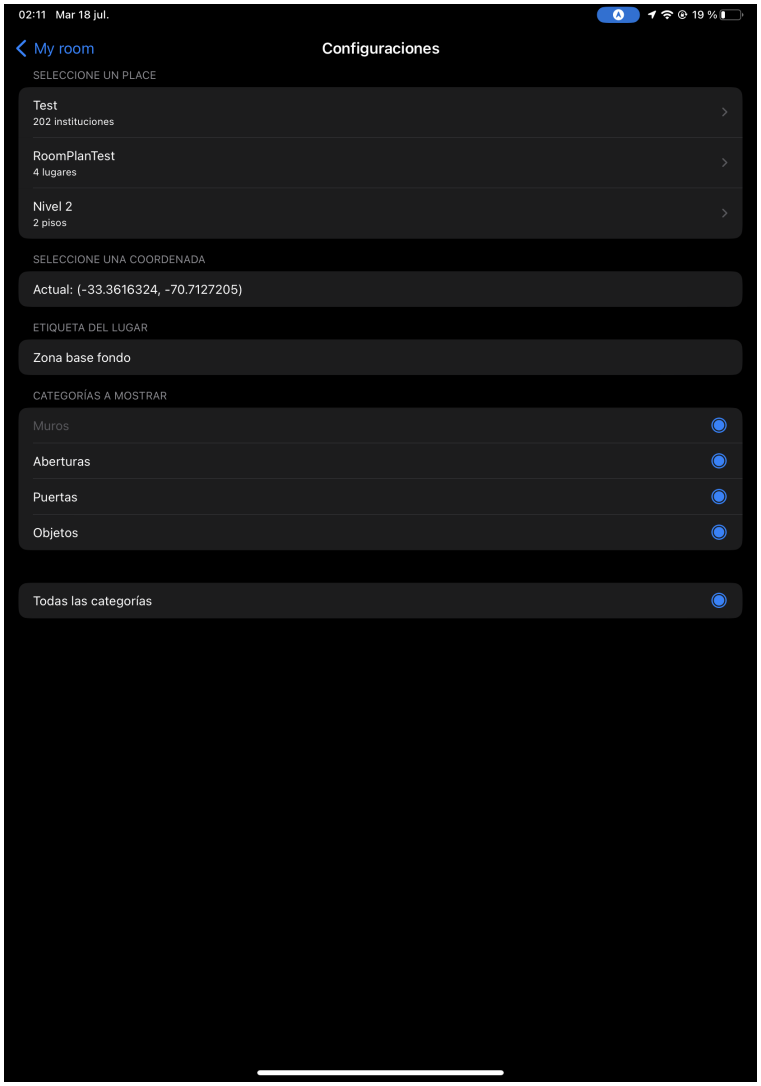

Figura 3.13: Configuraciones completadas

3.5.4. Editor de mapas

En esta ocasión, por "Vista de editor de mapas", se refiere a la vista del navegador del dispositivo (*Safari*) que se despliega y abre la URL[3] del Editor de mapas. Esta vista es accesible en 2 instancias:

1. Mediante la acción *Ver mapa del lugar*, implementada dentro de la Vista de Edición. Solo estará disponible después de terminar de configurar la ubicación.
2. Al finalizar la publicación de elementos. Independiente del éxito de la tarea, se ofrecerá la opción de abrir esta vista.

[3] *https://myvectormaps.lazarillo.app/{`placeId`}* , donde `placeId` es el identificador asignado por *Firebase* a la entidad Lugar seleccionada

Capítulo 4

Generación y exportación del plano

La siguiente sección refiere a la lógica desarrollada para generar el dibujo de un espacio interior a partir del resultado entregado por *RoomPlan*. Más específicamente, cómo se utilizará el conjunto de muros del resultado para dibujar los polígonos (1 o más) que representen el plano del lugar en el Editor de Mapas de Lazarillo.

4.1. Estado actual

4.1.1. Sobre el Editor de mapas

Hasta el momento en que este trabajo inició su desarrollo, como se adelantó en la sección de "Etapa de Publicación", el único *endpoint* para la creación de elementos que dispone el servicio de *backend* del Editor de mapas solo puede agregar un ítem a la vez. Hasta entonces no había necesidad de más, puesto que la única manera de crear un elemento era a través del servicio de *frontend* del Editor, donde el usuario seleccionaba una de las 3 opciones de creación (polígono, línea o nodo) y luego procedía a dibujar con el cursor en el mapa desplegado la figura que deseaba. En el momento en que el usuario daba click en un botón que indicaba "Finalizar", el elemento se guardaba inmediatamente. No es posible y no tiene sentido pensar en dibujar otro elemento en simultáneo con este procedimiento.

Una segunda limitación, que no se había mencionado hasta ahora, se relaciona con el objetivo de agregar etiquetas a un lugar. Como se mencionó en la entrada de "Etiquetas" en "Estado del Arte", las etiquetas se agregan automáticamente y solo se editan aquellas ya asignadas por defecto; no se deciden en el momento de la creación de un elemento y por tanto el *endpoint* no contempla recibir información sobre ellas. Para esto, no hubo otra alternativa que construir un *endpoint* nuevo que permita incluir en el cuerpo de la solicitud un diccionario de las etiquetas con las que se inicializará un elemento del mapa. Esto se realizó duplicando el *endpoint* de creación original, y agregando la instrucción que extrae el parámetro `tags` del *JSON* recibido, reemplazando luego el valor por defecto que se tenía (Ver ejemplo de *JSON* aceptable en anexo A.3).

4.1.2. Consideraciones técnicas

Para la creación de un polígono el *backend* recibe una solicitud de tipo *POST* donde el contenido del cuerpo es en formato *JSON*. Originalmente se esperaba utilizar el formato *GeoJSON* (que es más común encontrar en el estándar de *OpenStreetMaps*), pero se encontró con que este servicio utiliza una versión con unas pequeñas diferencias que lo adaptan a necesidades propias de Lazarillo. De hecho, el *backend* por debajo lo convierte a un *GeoJSON*. En la Tabla 4.1 se indican los atributos que requiere el JSON aceptable.

Tabla 4.1: Atributos del *JSON* aceptado por el *backend*

Llave	**Descripción del valor**	**Tipo**
type	'polygon', 'polyline', 'node'	String
properties	Diccionario con la llave 'type' y su valor	{String: String}
feature	Diccionario con la llave 'properties' y su valor	{String: Dict}
_latlngs	Lista de diccionarios con llaves 'lat' y 'lng' y valores de tipo Double	[{String: Double}]
placeKey	ID del Lugar, como String	String
tags	Diccionario con dos pares: - Llave 'k' y como valor el nombre de la llave de la etiqueta para el Editor - Llave 'v' y como valor el nombre del valor para tal etiqueta	{String: Dict}
layer	Diccionario con la llave '*layer*' y como valor un diccionario con las llaves '*feature*', '*_latlngs*', '*placeKey*' y '*tags*'	{*String*: Dict}

El gran desafío corresponde a generar el valor para el atributo `_latlngs`. Este atributo describe una lista de pares latitud-longitud, y es lo que el *framework* de mapas lee para dibujar un polígono. Cada par representa un vértice de dicho polígono, el cual se dibuja en el visualizador siguiendo los mismos puntos en el orden en que aparecen en la lista. A diferencia de un *GeoJSON*, en este caso, la lista de pares latitud-longitud no está obligada a que el último elemento sea igual al primero; al ver la etiqueta "polygon", el *backend* se encarga de cerrar los vértices.

Con esto, se puede ver que para obtener el resultado deseado que satisface el objetivo de dibujar el lugar escaneado como un mapa, se debe lograr obtener los vértices de cada pared y ordenarlos según la contigüidad entre ellos.

4.2. Inconvenientes con resultados de RoomPlan

Durante el escaneo, lo que hace RoomPlan es generar un valor de tipo `CapturedRoom`, que es un tipo de dato diseñado para este mismo *framework* que almacena toda la información obtenida del espacio. En particular, devuelve listas con las representaciones de los elementos

detectados, almacenados en listas y separados por la categoría inferida. No existe una conexión entre elementos de la misma categoría, ni un motivo para el orden en que cada elemento aparece en la lista de su categoría.

Otra situación a tener en cuenta es el hecho de que cada valor de medición se representa por números de tipo *Double*. Con ellos, hay que tener en consideración que se pueden producir errores de precisión que pueden afectar en mayor o menor medida según el algoritmo que se utilice. En el caso de dos muros contiguos, para prácticamente todos los casos el punto en el plano en que ambos coinciden son idénticos para varios decimales de precisión. Esto otorga un nivel de certidumbre a la hora de elegir un umbral en el que se considerarán idénticos, pero que aún así no permitirá determinar una igualdad estricta confiable.

4.3. Variedad de espacios a considerar

Un problema en si mismo es la gran variedad de distribuciones de espacios que se pueden encontrar, las que ni siquiera son excluyentes entre sí. Estos ejemplos pueden corresponder a espacios reales o a un modelo incompleto que no fue capaz de captar el entorno en su totalidad.

Muros en común

Refiere un lugar como en la Figura 4.1 en donde una misma pared forma parte del contorno de dos habitaciones contiguas. Además, ha sido relativamente común que en estos casos haya más de un muro común. Es problemático para algoritmos que generan la lista de vértices a partir de revisar la lista de muros secuencialmente. O por lo menos, para una revisión secuencial en donde un elemento que ya fue considerado para un polígono, no se revisa más después.

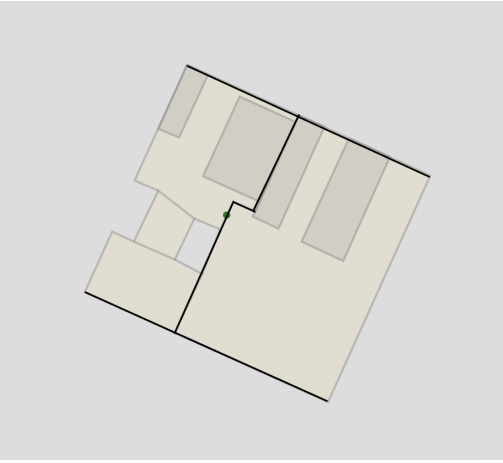

Figura 4.1: Ejemplo de espacio que presenta muros compartidos por habitaciones distintas

Divisor de ambientes

Aquí se señala como posiblemente problemática una situación en que existe un pared que sólo separa internamente dos ambientes dentro de un mismo espacio. Es similar al caso anterior, pues podría considerarse como una pared en común, pero como este muro extra no separa dos polígonos, habría que poner atención en cómo se dibuja el polígono al rededor de este elemento. Un ejemplo de esto se muestra en la Figura 4.2

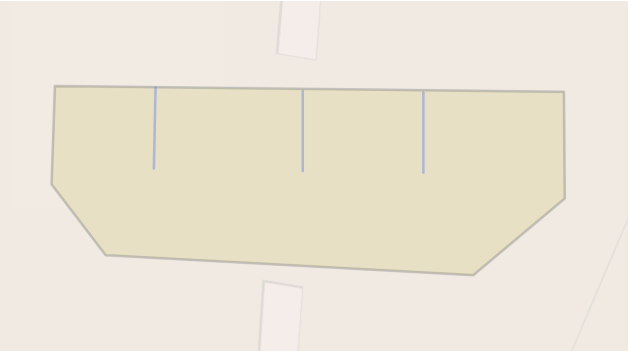

Figura 4.2: Ejemplo de espacio que presenta muros que dividen ambientes, pero no habitaciones completas

Espacio semiabierto

No es de extrañar que en ocasiones se intenten escanear lugares que no sean completamente cerrados. Un espacio semiabierto refiere a un lugar como el de la Figura 4.3 donde el contorno de él no cierra completamente, pero que de todas formas todos los muros están conectados entre sí. Dicho de otra forma, sólo existe una única zona abierta en el lugar. A priori, no es un espacio difícil de abordar, pero sí se debe tener en cuenta al plantear una solución.

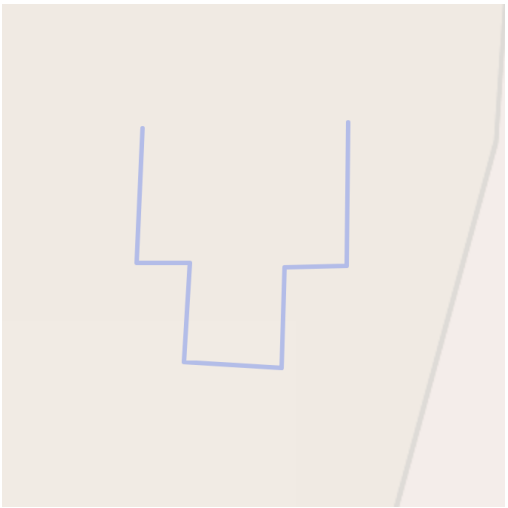

Figura 4.3: Ejemplo de espacio que cuenta con una única sección sin completar. Puede ser debido al diseño real de la habitación, o un error durante el escaneo

Espacio con varias secciones abiertas

Se habla de un espacio que contiene varias secciones disjuntas, como el de la Figura 4.4. Esto podría darse porque el contorno del espacio es realmente así o porque RoomPlan aproximó las dimensiones o posiciones de los muros encontrados de manera que no coinciden (o no cierran) algunos de sus vértices. Es posiblemente el caso más complejo de abordar porque si es un producto de un error en la detección de muros, entonces aumenta la probabilidad de que el mapa generado no sea satisfactorio para el usuario. También es difícil de abordar ya que estas aberturas (no confundir con la categoría detectada por RoomPlan, que refiere a entradas interiores) podrían corresponder a una sección de muros relativamente compleja en el espacio real.

Este caso se ha encontrado principalmente en espacios que son extensos en al menos una dimensión (como *halls* largos) y en espacios definidos por esquinas con varios recovecos.

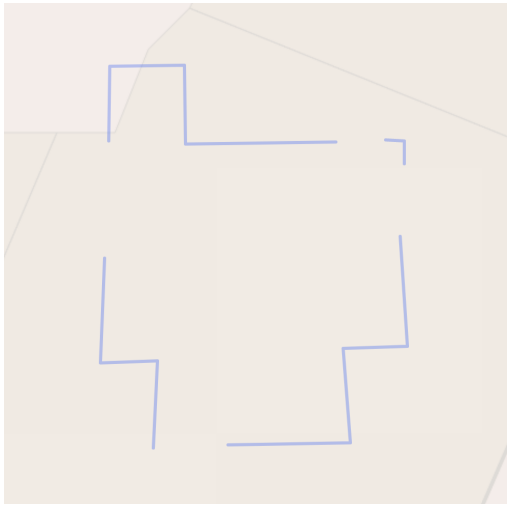

Figura 4.4: Ejemplo de espacio que presenta múltiples ausencias de conexiones entre murallas

4.4. Alternativas evaluadas para la generación de la lista de vértices ordenada

4.4.1. Envoltura convexa

Se puede definir como el recubrimiento o cobertura que encierra a todos los elementos de un conjunto, sin vértices cóncavos, de puntos, polígonos, círculos ni objetos 3D [15].

Es un algoritmo fundamental en geometría computacional. Sin embargo, no es posible plantearlo como solución absoluta puesto que eliminaría toda concavidad existente en el contorno real del espacio. Una habitación con forma de L, se convertiría bruscamente en algo más similar a un triángulo.

Esto es lamentable, ya que en varios casos podría determinar con gran precisión el contorno

de un espacio escaneado con muchos vértices, pero sólo si es que no tiene esquinas cóncavas. Se podría aplicar para espacios donde no se reconocen estas esquinas, pero en el tiempo en que se determina que se cumplen dichas condiciones, se podría ejecutar un algoritmo general que abarque más espacios.

4.4.2. Modelar contorno como grafo

Este acercamiento también generaría confianza dada la gran cantidad de algoritmos de probados que existen para abordar problemas. Sin embargo, el problema radica en la caracterización de los nodos del hipotético grafo.

Lo que se intentó diseñar fue un grafo cuyos nodos fuesen las esquinas de un espacio y cuyas aristas fuesen los muros que van de esquina a esquina. El primer problema de esto es caracterizar cada esquina. Como se mencionó antes, uno de los inconvenientes de los datos que se obtienen es que dos muros contiguos no comparten necesariamente el mismo vértice debido a problemas de precisión con el tipo de dato en el que se representan. Entonces, al definir un nodo del grafo como un par *Double-Double* y analizar las aristas existentes, se tendría que verificar todo el tiempo si cada nodo que se revisa es suficientemente cercano para ser considerado igual a otro nodo. Pero al definir un rango en el que se afirmaría que los bordes de dos muros forman una esquina, se corre el riesgo de que otro muro tenga un borde muy cercano a tal punto de manera que también sea considerado como parte de esa unión, sin necesariamente ser ese el caso.

Y aún más, tampoco resuelve el problema de los espacios con varias secciones abiertas. Aunque es una estrategia que sí podría ayudar cuando se tienen espacios donde varias paredes son paredes en común de distintos ambientes. Pero aún está pendiente encontrar un algoritmo concreto que sea útil para el caso.

4.4.3. *Greedy*

La tercera opción evaluada, es un acercamiento mas avaro para resolver el problema. Consiste en tomar un primer elemento que represente a un muro, guardarlo en una nueva lista, elegir uno de sus bordes, y buscar en todos los demás muros otro borde suficientemente cercano (para creer que en la práctica es el mismo punto). Al encontrarlo, se guarda como el segundo elemento de la lista nueva mencionada y se repite el proceso hasta que no se encuentren muros. De esta forma, se logra generar una lista ordenada de muros según sus vértices, a falta de un poco de procesamiento para adaptarse a lo que requiere el *back-end*.

A priori tampoco resuelve el problema de un espacio con varios espacios abiertos. Pero se puede complementar con una estrategia que solucione parcialmente la pérdida de información del resto de nodos. Los nuevos pasos serían:

1. Crear una nueva lista vacía que se utilizará para almacenar las listas obtenidas en la descripción recién mencionada. En otras palabras, una lista de listas de muros o lista de listas de polígonos.

2. Cada vez que el proceso llegue al punto en que no logra encontrar un muro, se agrega la lista de muros generada a la lista superior del paso anterior.
3. Se repite el proceso hasta que no quede ningún muro sin leer.

Con esto, se espera que se forme un polígono por cada conjunto de muros contiguos y que la suma de estos conjuntos tenga la forma del espacio completo, similar a un rompecabezas.

Se entiende que esta estrategia no representa una solución definitiva, pero si una solución que le permite al usuario comprender un nivel de avance en el trabajo, y de todas maneras le permitiría observar la forma completa del espacio escaneado, a falta de depuración. Esta depuración incluso se podría delegar al Editor de Mapas.

Por todo esto, y dado que se espera que para un escaneo estándar es optimista pensar que el modelo generado no presente alguna de las incongruencias mencionadas, se decide que ésta será la primera estrategia a utilizar. No tiene problemas con las esquinas cóncavas y ofrece una solución parcial para las secciones disjuntas. En cualquier caso se realizarán pruebas y se evaluarán con usuarios los resultados obtenidos para determinar si es posible alcanzar un resultado más cercano a la disposición real del espacio realizando sólo ajustes menores o si es necesario buscar otra estrategia desde cero para abordar el dibujo del plano (como podría ser utilizar algoritmos distintos para casos distintos).

4.5. Procedimiento para muros

A continuación se hablará en detalle sobre cómo se convierte paso a paso un objeto de tipo `CapturedRoom` a una lista de vértices ordenados por contigüidad que definen un plano.

4.5.1. Clases y estructuras relevantes

En todo este flujo, se hace uso de una serie de clases, estructuras y enumerables que se utilizan para asociar, definir y agregar semántica a conjuntos de datos que se representarán agrupados entre sí.

- Se crea la enumeración `DefaultObjects` para definir un conjunto fijo de nuevos valores que sirvan como equivalencias para cada categoría de RoomPlan, y definir también propiedades y comportamiento asociados estrictamente a alguno de estos valores específicos.
- Se crea la estructura `Vertex`[1] para definir un vértice de un futuro polígono. Sus atributos son dos elementos de tipo *Float* que representan un par de coordenadas (x, y) o latitud-longitud. También se le implementa un método personalizado para definir la igualdad entre dos `Vertex`, la cual será verdadera sí y sólo sí la distancia entre la primera y la segunda coordenada es menor a un valor muy pequeño[2].

[1] Se utilizará una *Struct* porque no se requiere ni herencia ni mutación de instancias.
[2] El umbral utilizado por ahora es 0,0001, y no ha presentado problemas en ninguna prueba de la aplicación.

- Se crea la clase `RequestObject`[3] para encapsular la información necesaria que se necesita transmitir desde la Vista de Edición. Sus atributos refieren al identificador local del objeto que se desea publicar, el tipo del objeto (con las propiedades internas que este tiene), la lista de vértices que define el contorno, el *data buffer* que será enviado como el cuerpo de la solicitud y el estado de la misma solicitud. A partir de esto es posible que en la Vista de Publicación se pueda manejar la *request*, *setear* un nuevo estado si se dan las condiciones y dar retroalimentación al usuario de toda esta información.

4.5.2. Creación del modelo en SceneKit

Al momento de cargar la vista y todas las componentes gráficas, se ejecuta la función `sceneSetUp()` que configura los elementos de la escena. Esta configuración revisa un diccionario llamado `categoriesToDisplay` cuyas llaves son los nombres de los categorías de RoomPlan y sus valores son Booleanos que indican si tal categoría está seleccionada como visible. Entonces, `sceneSetUp` revisa cada valor del diccionario y ejecuta el método `drawCapturedRoomElements(objectType)`, que se encarga de extraer la lista elementos de categoría `objectType` desde la instancia de *CapturedRoom* para dibujarla como nodos de escena por medio de la librería *SceneKit* (Ver código con que se realiza esta acción en anexo A.4). Si la vista sólo es desplegada y no cargada (es decir, se vuelve desde la vista de Configuraciones por ejemplo), se revisa de igual manera `categoriesToDisplay` para eliminar todo nodo cargado que esté en una categoría que podría ya no estar seleccionada como visible o para agregar los nodos de una categoría que antes no estaba visible y el usuario recién configuró que ahora sí.

Una vez revisadas las categorías visibles y agregados los nodos correspondientes al nodo raíz del grafo de escena, se crea un arreglo de respaldo `backUpSceneViewNodes` que contenga exactamente los mismos nodos que los que tiene el nodo raíz. Este nuevo arreglo tiene como función servir de respaldo para confirmar o deshacer acciones de edición por parte del usuario; es una versión definitiva y segura de los nodos con únicamente los cambios que han sido confirmados.

4.5.3. Preparación de datos

Al momento de dar click en "Publicar", inicia el procesamiento de los datos a partir de los nodos generados en la sección anterior. Primero se inicializa una estructura vacía de tipo Cola (llamada `requestQueue`) para contener objetos de tipo *RequestObject*. A continuación se revisan todas las categorías de `DefaultObjects` saltándose aquellas que no están marcadas como visibles en `categoriesToDisplay` y se genera la colección de arreglos de coordenadas para todos los elementos de cada categoría visible. Sin pérdida de generalidad, en el caso de los muros esta colección se genera siguiendo estas instrucciones:

1. Filtrar todos los nodos de `backUpSceneViewNodes` que pertenezcan a la categoría en cuestión (Muro, por ejemplo) y guardarlo en una lista `nodes`, y asegurarse que `nodes`

[3]Se utilizará una *Class* porque se requerirá modificar los valores de las instancias de este tipo

no esté vacía para poder continuar. En este caso, `nodes` contiene una lista con los muros de la escena.

2. Crear una nueva lista `coordinatesArray` a partir de la realización de una operación de mapa sobre `nodes`. Esta operación debe convertir nodos de *SceneKit* en un par de `Vertex` que representen los bordes del muro respectivo. Pero como los bordes de cada muro no son explícitos en los atributos del nodo, se deben inferir utilizando los valores de las transformaciones de rotación, escalación y traslación desde el centro arbitrario de la escena. Se puede ver la representación de este paso como código en el anexo A.5.
3. Ordenar elementos de `coordinatesArray` por contigüidad en una nueva lista denominada `sortedPairsCollection`. Esta lista contiene listas de polígonos que definen un contorno (o una lista de puertas o una de objetos, para otras categorías). La explicación del algoritmo para este paso se encuentra más adelante.
4. Revisar para cada lista contenida en `sortedPairsCollection` que el último muro conecte con el primero. Para esto, se compara el vértice destino del último muro con el vértice origen del primer muro y si no son iguales bajo el criterio definido para un `Vertex`, entonces se agrega un nuevo que tenga como origen el destino el último muro y como destino el origen del primero. Básicamente, un nuevo muro que conecta ambos extremos.
5. Mapear cada sublista de `sortedPairsCollection` a una lista conservando sólo el vértice origen de cada muro (acércandose así el resultado al formato que exige el Editor de Mapas). Esto se guarda en una nueva lista `sortedCoordinatesCollection`.
6. Se mapea cada elemento de `sortedCoordinatesCollection` a un `RequestObject` y se retorna.

Para cada elemento de esta lista obtenida, ya es posible generar una *request*. El cuerpo de ella se construye utilizando la función mostrada en el anexo A.6. Utilizando el Piso, Lugar, y ubicación seleccionada en configuraciones, es posible transformar las coordenadas de los vértices desde metros a latitud-longitud. Luego se obtienen las etiquetas asignadas en las propiedades de cada categoría en `DefaultObjects` y se tiene toda la información que el *backend* necesitará leer.

Finalmente, el objeto `RequestObject` completamente configurado se encola en `requestQueue`. Una vez hecho esto para todas las categorías, se envía esta cola a la Vista de Publicación, que se encargará de realizar la conexión.

4.5.4. Ordenar muros por continuidad

Corresponde a la idea descrita por el acercamiento *greedy* discutido anteriormente. El algoritmo recibe como *input* una lista de pares de `Vertex` y entrega un *output* del mismo tipo (Ver código en anexo A.7).

1. Se inicializa una lista vacía (llamada `polygon`) donde se agregarán los muros (como pares de vértices) que definen un polígono.

2. Se inicializa una lista vacía (llamada `polygonCollection`) donde se agregarán listas que definen un polígono.
3. Se verifica que existan nodos suficientes para realizar la operación y se agrega el primer nodo del input a `polygon`.
4. Se lee el vértice destino del último muro agregado a `polygon` y se busca en el input otro muro que tenga algún borde igual[4] a ese vértice. Cuando se encuentra, se agrega tal muro a `polygon`. Si el vértice de la igualdad es el vértice destino del nuevo muro, entonces se invierte el par de manera de seguir una continuidad.
5. Se repite el proceso con el nuevo vértice agregado. Si al repetir el proceso con el muro nuevo se cumple que ningún muro es contiguo a él, se agrega la lista `polygon` a `polygonCollection`, se vacía la lista `polygon` y se repite el proceso tomando el siguiente vértice del input que no se ha leído para iniciar un nuevo polígono.
6. Si se vacía la lista del *input*, el algoritmo termina y retorna `polygonCollection`.

Sea n la cantidad de elementos contenidos en el *input*. Este algoritmo toma un elemento y revisa los otros $n-1$ del *input* en el peor caso. Luego, al encontrar ese elemento, revisa los siguientes $n-2$ elementos, y así sucesivamente hasta vaciar la lista. Esto equivale a realizar una cantidad de revisiones igual a

$$1 + 2 + 3 + ... + n - 2 + n - 1 = (\sum_{i=1}^{n} i) - n = \frac{n^2 - n}{2} \tag{4.1}$$

Considerando además que cada revisión es de orden constante, la cantidad total de operaciones del algoritmo es $\Theta(n^2)$.

4.5.5. Procedimiento para objetos y entradas

Hasta ahora se ha abordado la lógica utilizada para la creación de muros, que ha requerido especial dedicación debido a que es el núcleo del trabajo y a la dificultad que conlleva ordenarlos. Ahora, se hablará de la lógica para objetos y entradas.

Objetos (Categoría de RoomPlan)

La categoría "Objetos" de RoomPlan es diferente a las categorías de Muros, Ventanas, Puertas y Aperturas. Estas últimas, corresponden a instancias de `CapturedRoom.Surface`, a diferencia de Objetos que tiene su propio tipo `CapturedRoom.Object`. Además, posee una subcategoría que corresponde a una predicción de la misma *API* que abarca múltiples opciones menos genéricas; principalmente mobiliario. El objeto gráfico con el que RoomPlan[5]

[4]Se refiere a la igualdad aproximada de *Floats* definida para objetos de tipo `Vertex`

[5]Al menos en su primera versión, que es la utilizada en este trabajo

los representa en su modelo es un hexaedro siempre. Por lo tanto, para cada objeto el único problema a resolver es la obtención de los 4 puntos que definen la proyección planar del hexaedro.

Si bien los pasos a seguir para dibujar un objeto son similares a los pasos con que se dibujan los muros, la dificultad es considerablemente menor ya que no es necesario preocuparse por la diversidad de geometrías que pueden definir una habitación.

El algoritmo para obtener `RequestObjects` de categoría `Object` es:

1. Filtrar elementos de la escena que pertenezcan a la categoría, guardar en una lista `nodes` y asegurar que la lista obtenida no está vacía para poder continuar.
2. Mapear `nodes` en una nueva lista `coordinatesArray`. La operación del mapeo para cada nodo consiste en obtener los 4 vértices de la proyección plana del hexaedro utilizando las transformaciones geométricas existentes en el nodo, y ordenarlos utilizando el algoritmo de Envoltura Convexa[6]. Luego, crear un `RequestObject` que contenga los 4 vértices ordenados y el identificador local.
3. Retornar `coordinatesArray`.

Entradas (Categorías *Door* y *Opening* de RoomPlan)

Dado que estos elementos se representarán como nodos singulares en el mapa que se desea generar, sólo se necesita definir el punto geográfico en el que serán anotados.

El paso a paso esta vez será:

1. Filtrar elementos de la escena que pertenezcan a la categoría, guardar en una lista `nodes` y asegurar que la lista obtenida no está vacía para poder continuar.
2. Mapear `nodes` en una nueva lista `coordinatesArray`. La operación de mapeo será crear un `Vertex` a partir de las coordenadas en el plano (ignorando la componente vertical) del centro del nodo gráfico dibujado y luego crear un `RequestObject` que contenga tal vértice, incluyendo también el identificador local.
3. Retornar `coordinatesArray`.

[6]En esta ocasión, no hay que preocuparse de vértices cóncavos que puedan desaparecer

Capítulo 5

Evaluación

En el siguiente capítulo se abordarán las diferentes estrategias y métricas utilizadas para la evaluación del trabajo realizado.

5.1. Encuestas de usabilidad

Dentro de los objetivos establecidos se menciona que el diseño del sistema debe resultar en una interfaz amigable y una experiencia cómoda para el usuario. Si bien estos criterios pueden ser percibidos como ambiguos, es posible capturar la percepción de los usuarios generada por el sistema al momento de su utilización para así validar o replantear las decisiones de diseño tomadas a lo largo del desarrollo.

En el caso de esta aplicación se está hablando de la creación de una herramienta de trabajo para el grupo de operaciones de Lazarillo, principalmente. Por lo tanto es imperante recibir retroalimentación que permita definir las directrices del trabajo futuro del sistema.

Para lograr captar los puntos clave de la experiencia de los usuarios, se hará uso de dos cuestionarios de usabilidad basados en ejemplos construidos por organizaciones dedicadas a la investigación sobre metodologías y técnicas de evaluación de experiencia de usuario.

Con algunos *testers* se tuvo que coordinar un lugar y fecha para reunirse y realizar pruebas con la aplicación. Se llevaron a cabo en el *iPad* de desarrollo, ya que ellos no contaban con un dispositivo con el sensor Lidar. Para otros usuarios, se subió una versión Beta 1,2,4 a la *AppStore* de *Apple*, a la que podían acceder mediante una invitación. En total se evaluó la experiencia de 5 personas, todos futuros clientes reales del sistema. Se construyó un formulario por medio de *Google Forms* para esta ocasión, dada la familiaridad que suelen tener las personas con él y por los recursos gráficos que utiliza en la entrega las respuestas.

5.1.1. *Single Ease Question* (SEQ)

Single Ease Question es una formato de pregunta que escala en 7 puntos la dificultad percibida por el usuario a la hora de llevar a cabo una tarea [24].

Se decidió que existen dos tareas principales reconocibles como flujos de usuario: escanear un espacio interior, y convertir un modelo escaneado a un mapa. Además, se le dio un espacio para explicar su respuesta, y una pregunta extra para evaluar la experiencia en términos generales.

Tarea 1

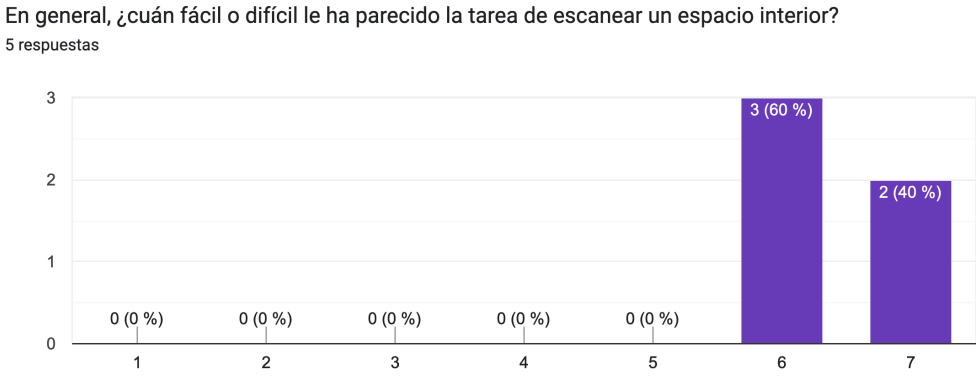

Figura 5.1: Opiniones sobre tarea 1

Acompañando la Figura 5.1, se recibieron las siguientes opiniones:

- "*App muy intuitiva. Sencilla de utilizar*"
- "*Intuitivo, la interacción fluye fácilmente*"
- "*Gracias a la aplicación, escanear un lugar es tan fácil como mover tu celular al rededor del espacio que quieres escanear por unos pequeños minutos*"
- "*Fácil, porque se necesitan pocos clicks para iniciar, terminar y editar*"
- "*Creo que fue fácil e intuitivo, pero haría un breve instructivo de como utilizar la aplicación y que el resultado sea el más óptimo posible*"

Tarea 2

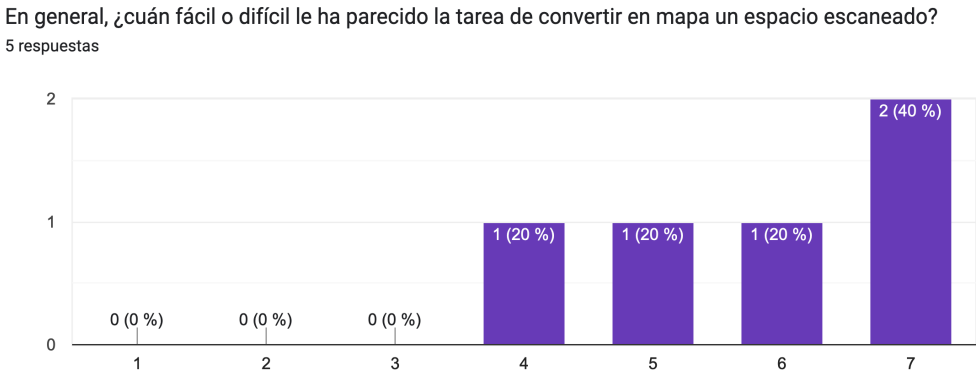

Figura 5.2: Opiniones sobre tarea 2

Acompañando la Figura 5.2, se recibieron las siguientes opiniones:

- "*No hubo ninguna complicación en convertir lo escaneado en plano 2D*"
- "*Al publicar no se aplicaron los últimos cambios ya que no se había guardado, pero no dio aviso. El plano del lugar inicialmente se construyo en varios polígonos independientes que no encajaban, se tuvo que limpiar los muros interiores para que mejorar el resultado*"
- "*Porque no hay muchos pasos que completar entre escanear y mirar el mapa*"
- "*Son pocos pasos adicionales y es bastante sencillo a comparado lo que hay que hacer actualmente para crear un mapa.*"
- "*Fue una tarea simple.*"

Completo

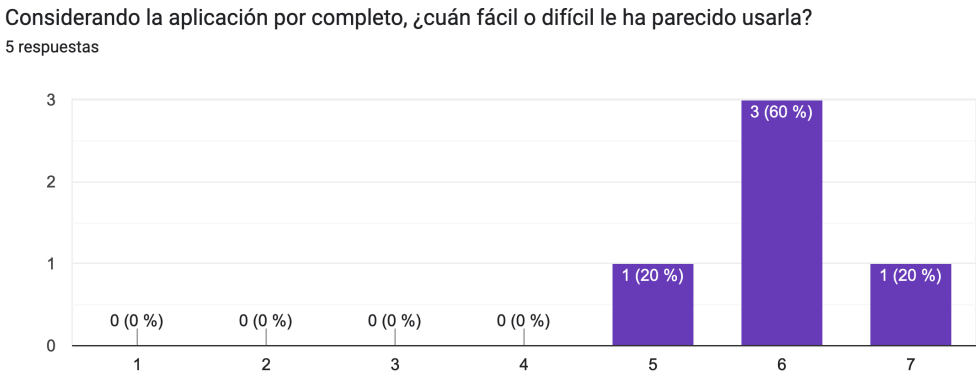

Figura 5.3: Opiniones sobre aplicación completa

Acompañando la Figura 5.3, las sensaciones generales expresadas fueron:

- "*En general tanto el escaneo como la interacción con los elementos escaneados es intuitiva y relativamente fácil de utilizar, lo que podría mejorar es la generación del mapa del Place. Que el usuario proporcione algunos input que puedan ayudar en la generación del mismo, como por ejemplo marcar paredes del contorno*"
- "*Una vez editando el mapa, es en su mayoría intuitiva*"
- "*Pocos clicks para lograr el objetivo*"
- "*Creo que en general es muy intuitiva, creo que las dudas pueden surgir desde el usario al utilizarla por primera vez en relación a algunos casos de uso, como por ejemplo las puertas y ventanas. Pero en general después de haberla utilizado una vez, sería mucho más simple en una segunda instancia.*"

Esto permite ver que en la gran mayoría de los casos, las tareas fueron consideradas relativamente sencillas de llevar a cabo. En el caso de la tarea de publicación de un mapa, se debe tener cuidado y corregir los aspectos discutidos durantes las sesiones de prueba y los recibidos en los comentarios del formulario. Los comentarios recibidos en la pregunta sobre la aplicación completa reafirman esta idea evidenciando que los usuarios se sienten satisfechos hasta ahora con el producto y su potencial.

5.1.2. *System Usability Scale* (SUS)

System Usability Scale provee de un cuestionario rápido y directo para medir aspectos fundamentales de la usabilidad de un sistema. Consiste de 10 preguntas, con 5 posibles respuestas cada una que varían desde "Muy de acuerdo" a "Muy en desacuerdo". Es un

estándar ampliamente utilizado en la industria, y con un sinfín de referencias en artículos y publicaciones que validan sus resultados [17].

La interpretación de los resultados no es directa. Para calcular el puntaje final, los ítems de las Figuras 5.4, 5.6, 5.8, 5.10 y 5.12 asignan puntos del 0 al 4, donde 4 es el "Muy de acuerdo". Para los ítems de las preguntas 5.5, 5.7, 5.9, 5.11 y 5.13, se asignan 4 puntos para la respuesta "Muy en desacuerdo". Luego, la suma de los puntajes tampoco debe verse como un porcentaje de éxito [12]. Por ejemplo, un puntaje de 50/100 no implica que la usabilidad del sistema es de un nivel promedio. Diferentes sitios sugieren que un puntaje SUS promedio es al rededor de 68 [25, 44].

Pregunta 1

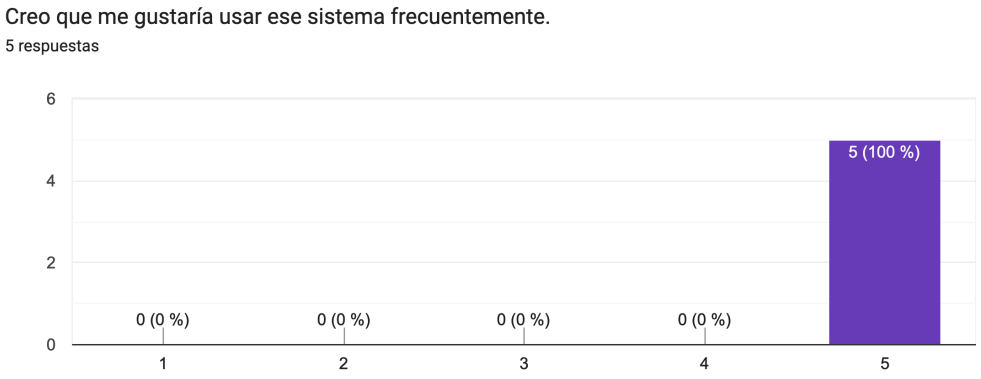

Figura 5.4: Resultados de pregunta 1 del cuestionario SUS

Pregunta 2

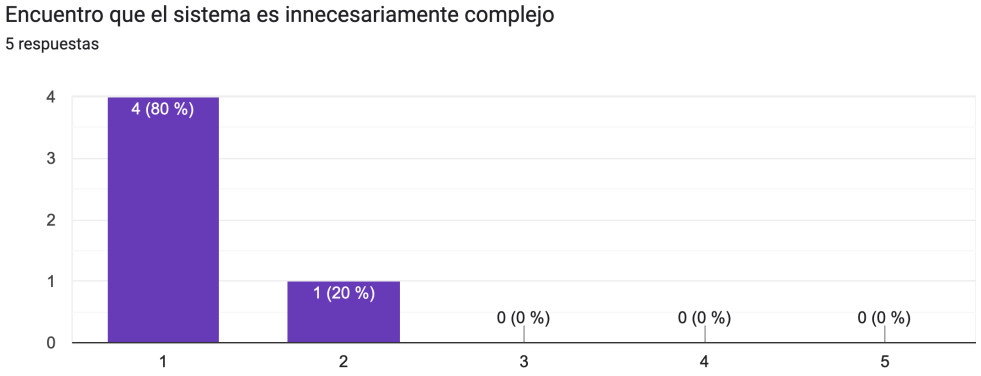

Figura 5.5: Resultados de pregunta 2 del cuestionario SUS

Pregunta 3

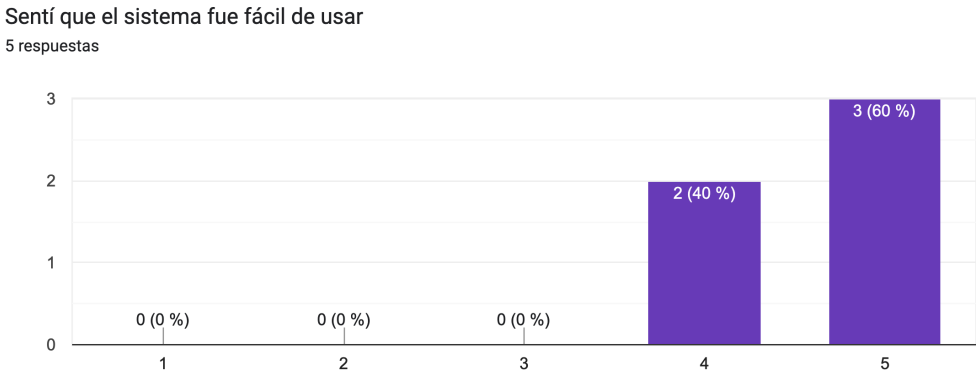

Figura 5.6: Resultados de pregunta 3 del cuestionario SUS

Pregunta 4

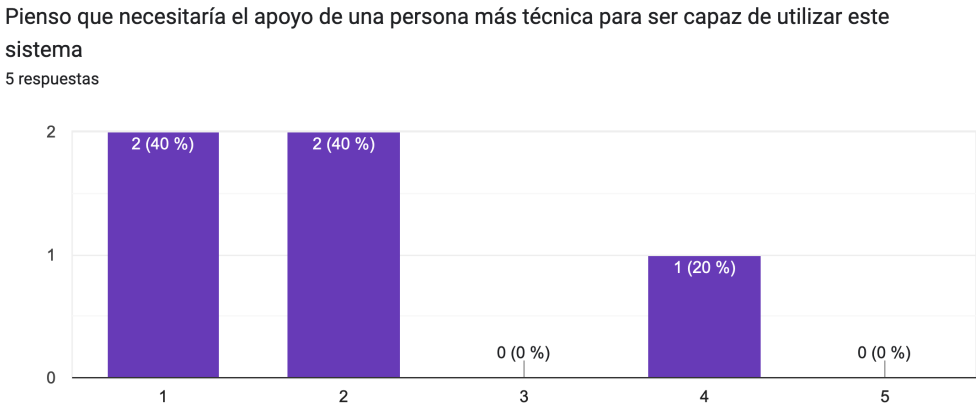

Figura 5.7: Resultados de pregunta 4 del cuestionario SUS

Pregunta 5

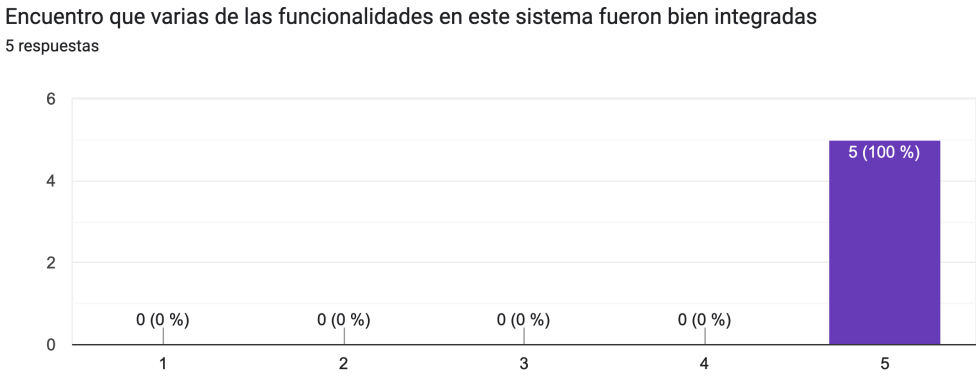

Figura 5.8: Resultados de pregunta 5 del cuestionario SUS

Pregunta 6

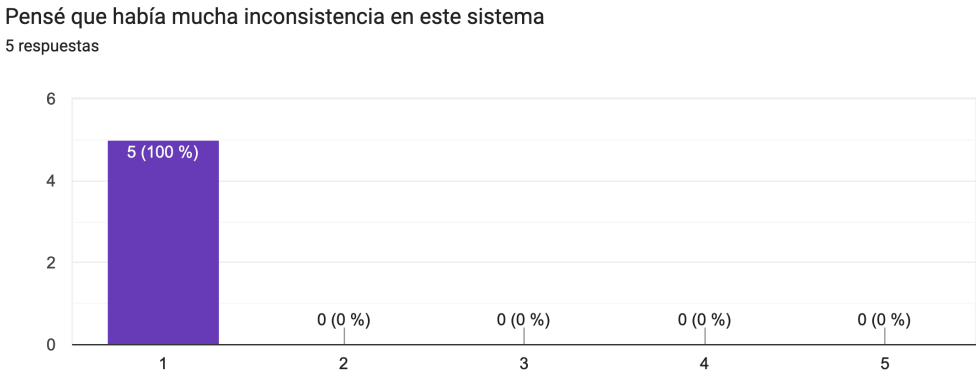

Figura 5.9: Resultados de pregunta 6 del cuestionario SUS

Pregunta 7

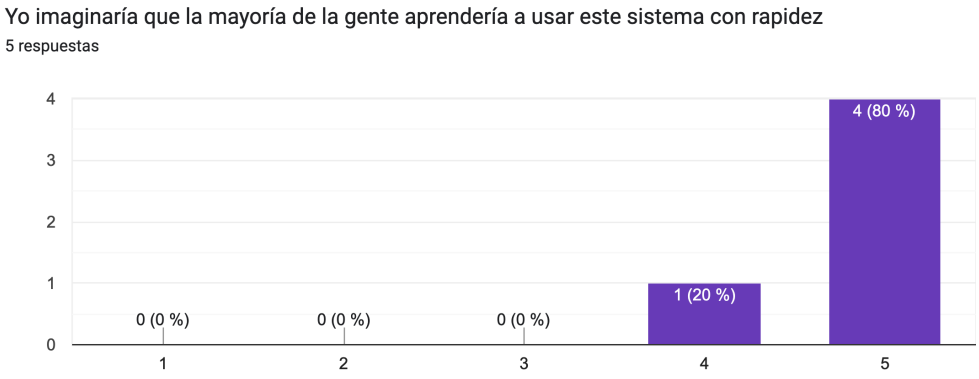

Figura 5.10: Resultados de pregunta 7 del cuestionario SUS

Pregunta 8

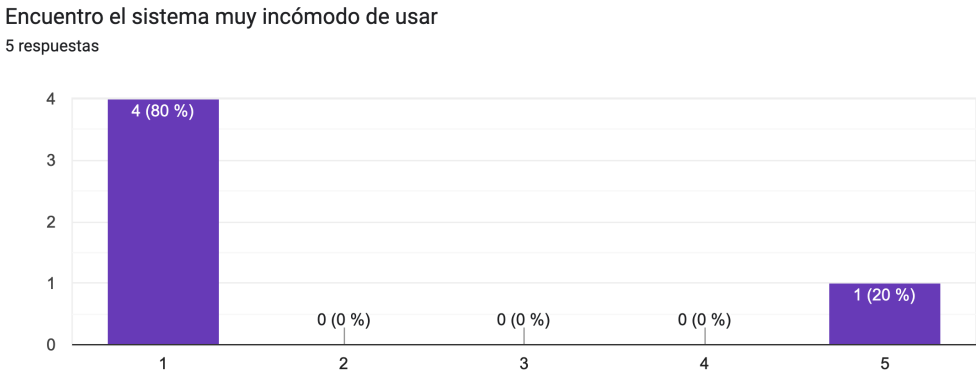

Figura 5.11: Resultados de pregunta 8 del cuestionario SUS

Pregunta 9

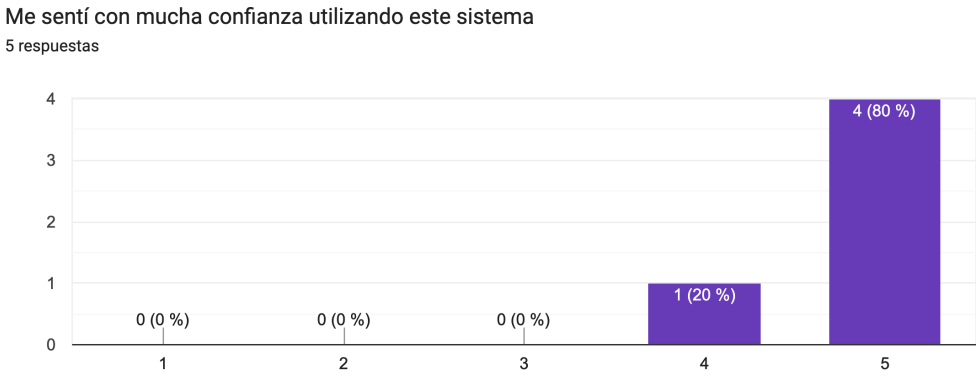

Figura 5.12: Resultados de pregunta 9 del cuestionario SUS

Pregunta 10

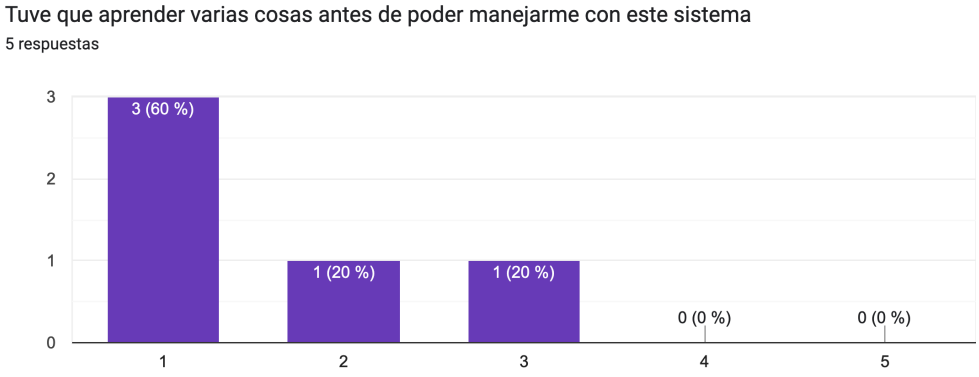

Figura 5.13: Resultados de pregunta 10 del cuestionario SUS

Resultados generales

La suma total de puntajes tiene un valor de 183. Ponderando por 2,5 y dividiendo en los 5 participantes, se llega a que el puntaje *SUS* de la aplicación es de 91,5. Este puntaje es catalogado como muy bueno según los sitios citados, mucho más cercano al valor máximo 100 que al promedio. Por lo tanto, a través de esta medición se considera que el sistema en su estado actual y en su potencial son evaluados como excelente.

5.2. Tiempo que demora el escaneo

Las sesiones de prueba en compañía de operaciones se realizaron en una nueva habilitación de un cliente de la empresa en Santiago. Este espacio ya cuenta con un mapa creado de la manera tradicional. Es decir, se solicitó el plano, se revisó y luego se dibujó a mano elemento por elemento.

Según el mismo equipo de operaciones, sólo en el caso de contar con los planos del lugar este proceso suele durar en promedio entre 2 1/2 horas a 3 horas, dependiendo de la complejidad del espacio y también de la experiencia en la tarea de quien esté encargado de dibujarlo. Si no se cuentan con los planos, pueden pasar meses sin que se pueda concluir el trabajo. Según la cantidad de subespacios contenidos dentro de el espacio general, la tarea puede durar más tiempo[1]. Además, aquellos miembros del equipo que tienen más experiencia y práctica en el dibujo también se encargan de instruir a otros miembros, realizando tutoriales y entregando *tips* hasta que alcancen un nivel más avanzado.

Figura 5.14: Mapa de sucursal de un cliente de Lazarillo, construido según el método tradicional por el equipo de operaciones

A continuación se indicarán los tiempos de las sesiones de escaneo realizadas por los miembros del equipo de operaciones presentes en la instancia de *testing*.

[1]Por ejemplo, centros comerciales.

Usuario	Tiempo (hh:mm:ss)
Usuario 1	0:41:33
Usuario 2	1:20:00 [2]
Usuario 3	0:36:34
Promedio	0:52:22

Tabla 5.1: Duración de sesiones de prueba con la aplicación

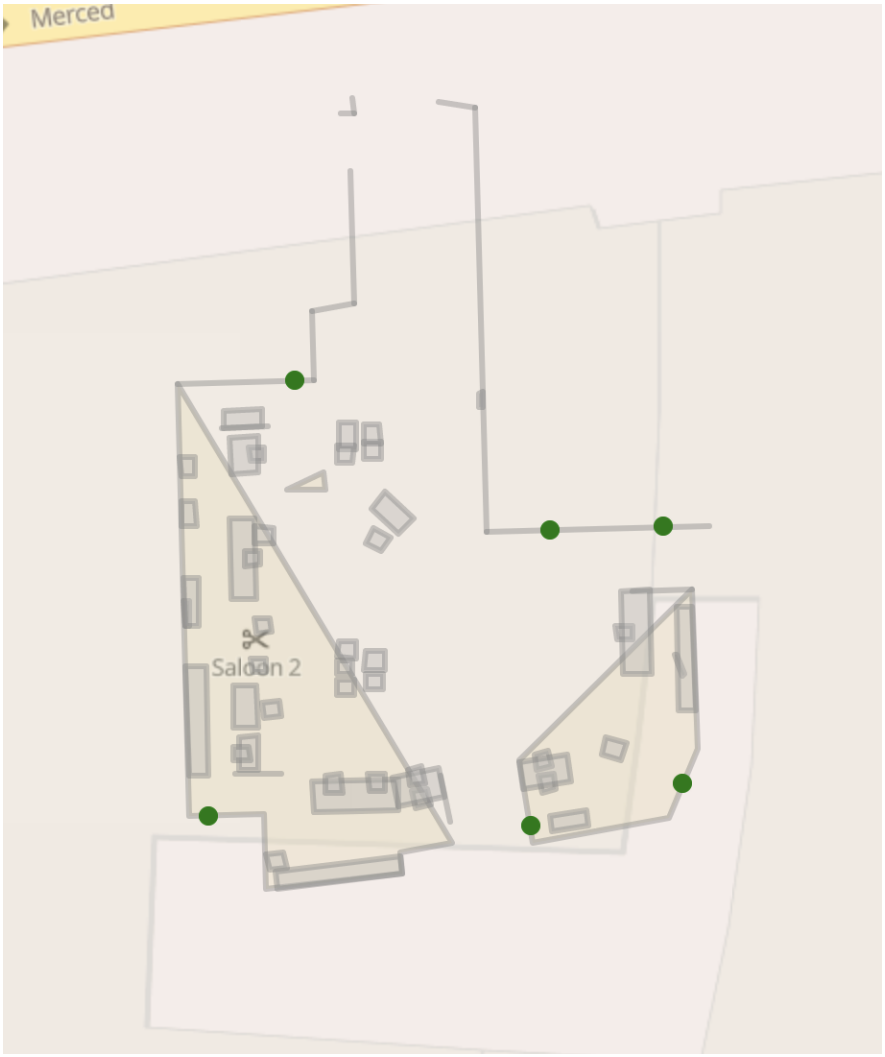

Figura 5.15: Mapa generado por el escaneo de la oficina, realizado por Usuario 1

La Figura 5.15 refiere a la misma habilitación que refiere la Figura 5.14. Además de la rotación de la primera figura, es posible observar que la forma del espacio del ambiente principal es similar al mapa actual. Sin embargo, las uniones entre muros no capturadas causaron polígonos divididos indeseados que ensucian el resultado.

Se puede ver también que los valores de tiempo de la Tabla 5.1 son varias veces menores a los tiempos del procedimiento tradicional. Y si se tomara la decisión de implementar definitivamente este sistema al proceso del equipo, el tiempo dedicado a la capacitación de otros

miembros también se reduciría o eliminaría por completo.

En cuanto al plano en sí, el dibujo del contorno presenta problemas con la conexión entre los muros, debido a la presencia de múltiples separaciones que no lograron completarse. Aunque la mayoría del mobiliario fue detectado y con un nivel de detalle mayor al plano oficial.

5.3. Precisión de resultados

La última perspectiva desde la que se espera evaluar el trabajo, refiere a la precisión de los resultados del escaneo. A diferencia de los puntos de vista anteriores, en esta ocasión no se cuenta con un umbral establecido con el cual comparar los resultados de esta evaluación. Por lo tanto, más allá de alcanzar un porcentaje de diferencia aceptable entre el valor real y el valor medido, estos valores se pueden utilizar para analizar el comportamiento de la herramienta en distintos objetos y encontrar en qué casos de uso aumenta la confiabilidad de los resultados.

A continuación se mostrarán dos comparaciones de resultados entre las mediciones tomadas personalmente con una huincha de medir y las medidas indicadas por RoomPlan, además de la diferencia relativa del último valor con el primero. El plano de las Figuras 5.16 y 5.17 corresponde al plano generado del escaneo del piso de una vivienda.

Las Tablas 5.2 y 5.3 refieren a los resultados para los muros detectados en la habitación de la prueba, mientras que las Tablas 5.4 y 5.5 refieren a los objetos detectados en el mismo espacio.

Muro	Real	RoomPlan	Δ (%)	Muro	Real	RoomPlan	Δ (%)
A	272 cm	275 cm	1.09	I	119 cm	122 cm	2.45
B	261.5 cm	265 cm	1.32	J	56 cm	56 cm	0
C	229 cm	229 cm	0	K	55 cm	52 cm	5.76
D	55 cm	52 cm	5.76	L	229 cm	229 cm	0
E	75 cm	67 cm	11.94	M	308 cm	315 cm	2.22
F	55 cm	62 cm	11.29	N	539 cm	540 cm	0.18
G	93 cm	99 cm	6.06	O	360 cm	363cm	0.82
H	96 cm	98 cm	2.04	P	310 cm	311 cm	0.32

Tabla 5.2: Comparación entre valores medidos con y sin RoomPlan para muros encontrados en la escena

Figura 5.16: Escaneo de ejemplo para comparación de medidas de muros

Medida	Valor
Número de muestras	14
Promedio Δ (%)	3.245
Desviación estándar (%)	3.879

Tabla 5.3: Métricas generales de los resultados obtenidos para muros

En la Tabla 5.3 la cantidad de muestras consideradas es 14 en lugar de 16, dado que dos de esas murallas fueron escaneadas una vez por cada lado, lo que se reflejó en entradas duplicadas.

Dimensión	Real	RoomPlan	Δ (%)	Muro	Real	RoomPlan	Δ (%)
A	50 cm	50 cm	0	F	240 cm	241 cm	0.41
B	110 cm	151 cm	27.15	G	80 cm	98 cm	18.36
C	105 cm	133 cm	21.05	H	240 cm	248 cm	3.22
D	198 cm	191 cm	3.66	I (ancho)	65 cm	67 cm	2.98
E	75 cm	63 cm	19.04	I (altura)	200 cm	200 cm	0

Tabla 5.4: Comparación entre valores medidos con y sin RoomPlan para las dimensiones de los objetos relevantes de la escena escaneada

Figura 5.17: Escaneo de ejemplo para comparación de medidas de objetos varios encontrados

Medida	Valor
Número de muestras	10
Promedio Δ (%)	9.587
Desviación estándar (%)	9.968

Tabla 5.5: Métricas generales de los resultados obtenidos para objetos

A continuación se presentan las métricas de la comparación completa entre los resultados del escaneo y las mediciones manuales en la Tabla 5.6.

Medida	Valor
Número de muestras	24
Promedio Δ (%)	5.888
Desviación estándar (%)	7.742

Tabla 5.6: Métricas generales de todos los resultados obtenidos

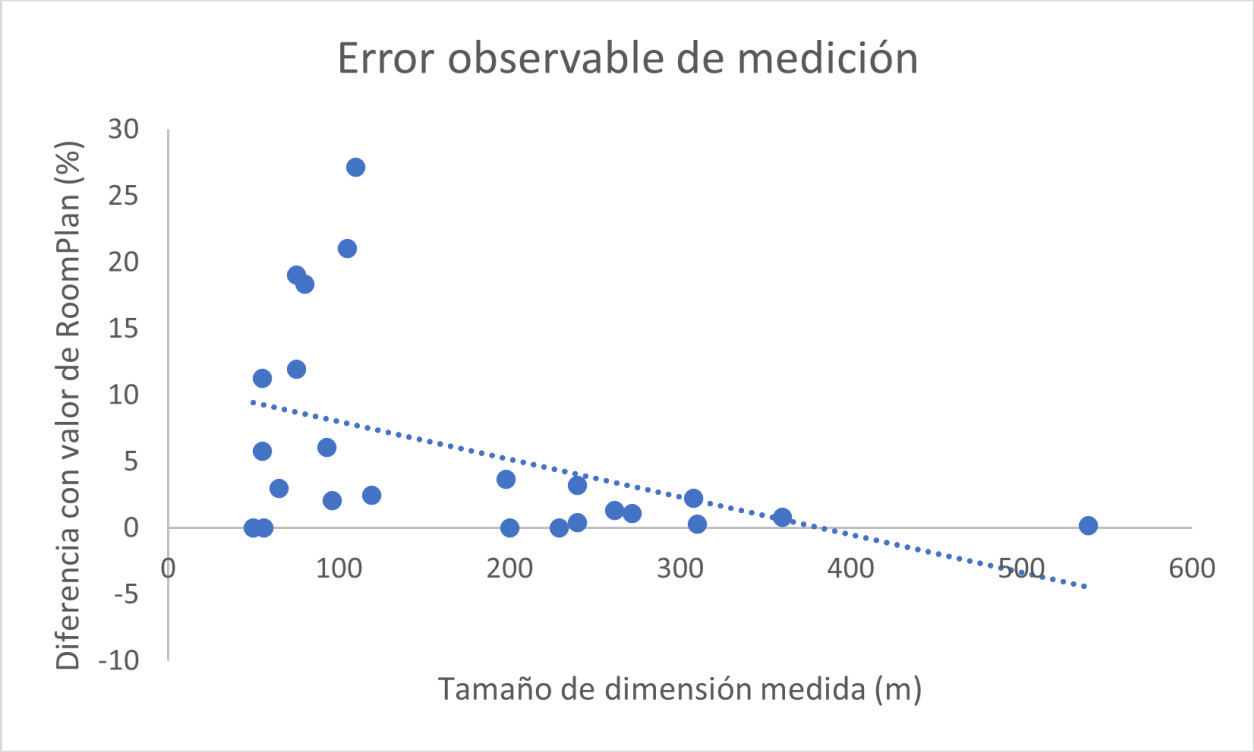

Figura 5.18: Tendencia del porcentaje de diferencia entre valor real del objeto medido con respecto al valor entregado por RoomPlan

Por último, se presenta la Figura 5.18 que consiste de un gráfico que ilustra una relación entre la magnitud del objeto escaneado y el error obtenido utilizando la aplicación. El eje x representa el tamaño o longitud de una dimensión de un ítem en centímetros, mientras que el eje y representa la diferencia porcentual entre el valor que se obtuvo utilizando *RoomPlan* con respecto al valor medido con una huincha de medir. Los puntos en el gráfico representan las 24 entradas totales entre la Tabla 5.2 y la Tabla 5.4, y la recta representa una regresión lineal ajustada a estos puntos para modelar el comportamiento subyacente.

La pendiente negativa de la línea indica que existe una tendencia del error porcentual a decrecer a medida que se intenta escanear objetos de mayor tamaño. En otras palabras, que la aplicación -o más específicamente *RoomPlan*- es más precisa en la medición de objetos cuanto más grandes estos sean. Esto podría atribuirse a que el sensor tiene un grado de error constante e independiente del objeto que se mida y que por tanto el porcentaje de este error decrece únicamente porque el valor de referencia con el que se compara aumenta. Una segunda posible explicación de esto nace del concepto de "nube de puntos" y sugiere que al ser mayor la cantidad de puntos que colisionan con la superficie del objeto grande que la cantidad que lo hacen con un objeto más pequeño, hay más información disponible para RoomPlan que este puede utilizar para la estimación. En ambos casos, se sugiere que la confiabilidad en los resultados de la digitalización de muros aumenta para espacios de mayor extensión.

Capítulo 6

Conclusión

El proceso de desarrollo de *LzIndoorScanner* ha sido una experiencia de aprendizaje constante. El camino no ha estado exento de contratiempos y en ocasiones, la frustración de no poder completar algunos de los hitos propuestos ha primado por sobre la expectativa del resultado final.

Se ha tenido la oportunidad de experimentar con un conjunto de herramientas sumamente interesantes, pero cuya contemporaneidad es una dificultad en si misma. El proceso de desarrollo incluyó enfrentarse a problemas distintos y más complejos a los experimentados en otros contextos donde las tecnologías utilizadas eran más familiares. Desde ese punto de vista, ya se puede aseverar que el trabajo ha logrado aportar valor en su autor.

Discusión de resultados

Desde el punto de vista de la empresa, se puede concluir que la gran mayoría de los objetivos planteados se han cumplido. El sistema es capaz de reconocer un espacio y generar un modelo en tres dimensiones de este a partir de RoomPlan. El modelo puede ser leído y reproducido por otras herramientas, es capaz de categorizar los elementos en la habitación y la información de la que dispone difiere a penas en un escaso porcentaje a las mediciones manuales que se hicieron, las que también cuentan con su propia imprecisión.

Luego, se ha establecido exitosamente la comunicación con la aplicación de edición de mapas y con el servicio de base de datos utilizado por Lazarillo, para así continuar con la generación de mapas vectoriales de planos; objetivo que también se ha alcanzado. Fue necesario agregar un nuevo *endpoint* para soportar la publicación conjunta de planos y etiquetas; nuevamente, tarea que se completó con éxito.

Para cerrar la iteración, una vez alcanzado el nivel de avance que permite lo fundamental de las acciones de escaneo y edición con publicación, la experiencia de uso de la aplicación fue evaluada con los mismos usuarios para los que se está desarrollando. La respuesta de ellos fue sumamente positiva, excediendo por varios puntos el umbral del valor promedio de usabilidad de las métricas expuestas, acompañada de comentarios de satisfacción y también

de expectativas por el potencial demostrado.

Cabe decir que hubo un último objetivo que no se logró cumplir. Este objetivo fue el que referían a estudiar la posibilidad de que al iniciar un escaneo en un espacio ya digitalizado, el dispositivo asociara el resultado ya existente al escaneo nuevo y de alguna manera geoposicionar al usuario. Por temas de tiempo, no se pudo evaluar de ninguna manera. Aunque con la experiencia alcanzada, sí se puede afirmar que tales funcionalidades escapan completamente del alcance de *RoomPlan*.

Sobre el nivel de precisión de los resultados, se pudo ver que el valor del error obtenido para los muros es 3 veces menor que el valor obtenido para los objetos. Cabe la posibilidad de que esto sea sólo una casualidad, pero esta situación ya había sido observada en un comienzo (lo que motivó a la creación de herramientas de edición del modelo, al fin y al cabo). Es posible que esto sea producto de que, a diferencia de como funciona con los muros, RoomPlan no restringe la geometría de los objetos según la relación que puede encontrar entre ellos. En el caso de los muros, RoomPlan está constantemente reajustando las dimensiones de cada elemento ya que entiende que hay una conexión que debe respetar y que los números deben ser tales que estos muros conecten en sus esquinas siempre. En el caso de un espacio que contenga más de un ambiente, como el de la Figura 5.16, RoomPlan entiende que existe una forma general que define todo un piso (generalmente un cuadrilátero por ejemplo) y recalculará las dimensiones de los elementos de manera de que se ensamblen dentro de esta figura contenedora.

En cuanto a lo que refiere al tiempo de realización de la tarea, las mejoras son considerables. Los usuarios demoraron casi una cuarta parte del tiempo en realizar el escaneo del lugar, con respecto al mejor caso promedio señalado para el procedimiento tradicional y esto sin tomar en cuenta el tiempo que dedica el experto y el aprendiz en la capacitación respectiva. Esto indica una mejora significativa en el tiempo que el equipo de operaciones debe dedicar a esta etapa. Y esto sólo considerando el caso en que un cliente ha otorgado los planos, lo que no siempre sucede. Existen clientes, como un servicio de metro por ejemplo, en que el periodo de entrega de planos es particularmente extenso debido principalmente a la burocracia propia de una industria con infraestructura sensible. Entonces, siempre con el consentimiento del cliente, la etapa de facilitación de estos documentos podría omitirse si se utiliza el nuevo sistema.

Se puede ver también que el resultado obtenido por RoomPlan en la sesión de prueba necesita depuración. El algoritmo actual sólo ofrece una solución parcial para poder obtener una versión estable de pruebas y es necesario evaluar otras alternativas para que la aplicación alcance la robustez necesaria para ser utilizada formalmente como herramienta de trabajo. Aún así, es posible observar que sí se obtiene la forma general del espacio de la oficina y que se asemeja al plano manual, justificando entonces el potencial y la satisfacción que los usuarios han declarado.

Trabajo pendiente y futuro

Como ya fue mencionado, los esfuerzos deben dirigirse ahora en la mejora del algoritmo que dibuja el contorno del espacio escaneado. Ya se han comenzado discusiones sobre una

alternativa: proveer al usuario de una acción que le permita unir dos muros. De esta forma sería posible "cerrar" el polígono requerido sin necesidad de cambiar el algoritmo actual y al mismo tiempo hacerlo según el criterio del usuario.

También quedó pendiente evaluar si el nivel de precisión de resultados obtenido puede considerarse como suficiente o no. No se pudo determinar cuál sería un umbral de tolerancia aceptable para mediciones de este tipo y las referencias encontradas en otros lugares refieren a áreas de estudio diferentes, como mediciones en la construcción o mediciones en laboratorio. Por ahora, se conviene que los resultados son aceptables según la opinión de los usuarios y que se establecen como *baseline* para cualquier otro método que se desarrolle posteriormente.

Otro comportamiento que quedó a medio terminar fue el manejo de errores en la subida de elementos al editor de mapas. Se mencionó que el envío de las solicitudes se maneja con distintas estructuras de tipo cola y que una de ellas se destinaría para contener a aquellas solicitudes que no pudieron completarse. Sería ideal guardar esa estructura en memoria o en caché para que el usuario pueda retomar la carga posteriormente.

Por último, quedó una extensa lista con mejoras de baja y mediana importancia sobre el diseño y la usabilidad de botones de la interfaz. Por ejemplo, se indicaba que al "Publicar" una escena, es poco intuitivo que esta acción no aplique automáticamente las modificaciones hechas en la Vista de Edición. O también dudas sobre por qué las alertas desplegadas no contienen siempre un botón de 'Aceptar"; se asumió que cerrar alertas al hacer click afuera de ellas era un comportamiento intuitivo, sin consultar con usuarios habituales de *iOS*.

Reflexiones finales

El proyecto realizado fue ambicioso en muchos sentidos. La misma tarea de trabajar con un *framework* recién liberado implica que existe un nivel de incertidumbre relativamente alto sobre si será posible alcanzar un resultado útil para la empresa. Pero en este momento, habiendo concluido el periodo formal de desarrollo, gran parte de esa incertidumbre ha desaparecido. Se ha logrado conocer el potencial del trabajo y sólo unas cuantas correcciones de menor envergadura separan a Lazarillo de poder contar con una aplicación que significará una mejora sustancial en los tiempos de digitalización de espacios.

Se ha desarrollado una herramienta de software funcional, validada por usuarios, que entrega resultados relativamente cercanos a lo esperado y que tiene espacio e ideas para mejorar. La satisfacción y las esperanzas del cliente con el potencial vislumbrado sugieren que el trabajo de ingeniería llevado a cabo puede aportar valor al nivel de las expectativas iniciales.

Bibliografía

[1] MapTiler AG. About the project — openmaptiles.org. https://openmaptiles.org/about/. [Accessed 25-Apr-2023].

[2] Apple. ARKit — Apple Developer Documentation — developer.apple.com. https://developer.apple.com/documentation/arkit. [Accessed 25-Apr-2023].

[3] Apple. CapturedRoom.Surface — Apple Developer Documentation — developer.apple.com. https://developer.apple.com/documentation/roomplan/capturedroom/surface. [Accessed 17-May-2023].

[4] Apple. Create parametric 3D room scans with RoomPlan - WWDC22 - Videos - Apple Developer — developer.apple.com. https://developer.apple.com/videos/play/wwdc2022/10127/. [Accessed 17-May-2023].

[5] Apple. MetalKit — Apple Developer Documentation — developer.apple.com. https://developer.apple.com/documentation/metalkit/. [Accessed 17-May-2023].

[6] Apple. objects — Apple Developer Documentation — developer.apple.com. https://developer.apple.com/documentation/roomplan/capturedroom/objects. [Accessed 17-May-2023].

[7] Apple. RealityKit — Apple Developer Documentation — developer.apple.com. https://developer.apple.com/documentation/realitykit/. [Accessed 25-Apr-2023].

[8] Apple. RoomPlan - Augmented Reality - Apple Developer — developer.apple.com. https://developer.apple.com/augmented-reality/roomplan/. [Accessed 25-Apr-2023].

[9] Apple. RoomPlan — Apple Developer Documentation — developer.apple.com. https://developer.apple.com/documentation/roomplan. [Accessed 25-Apr-2023].

[10] Apple. SceneKit — Apple Developer Documentation — developer.apple.com. https://developer.apple.com/documentation/scenekit/. [Accessed 25-Apr-2023].

[11] Apple. Swift - Apple (CL) — apple.com. https://www.apple.com/cl/swift/. [Accessed 25-Apr-2023].

[12] John Brooke. Sus: A quick and dirty usability scale. *Usability Eval. Ind.*, 189, 11 1995.

[13] Enrique Dans. The Incredible Shrinking LiDAR — forbes.com. https://www.forbes.com/sites/enriquedans/2020/09/11/the-incredible-shrinking-lidar/?sh=3305cd9557d6. [Accessed 25-Apr-2023].

[14] Digital.gov. About Us — Usability.gov — usability.gov. https://www.usability.gov/about-us/index.html. [Accessed 18-Jul-2023].

[15] EcuRed. Envoltura convexa - EcuRed — ecured.cu. https://www.ecured.cu/Envoltura_convexa. [Accessed 17-Jul-2023].

[16] ArcGIS Enterprise. GeoJSON—Portal for ArcGIS — Documentación de ArcGIS Enterprise — enterprise.arcgis.com. https://enterprise.arcgis.com/es/portal/latest/use/geojson.htm. [Accessed 17-May-2023].

[17] Assistant Secretary for Public Affairs. System Usability Scale (SUS) — Usability.gov — usability.gov. https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html. [Accessed 18-Jul-2023].

[18] American GeoSciences. What is Lidar and what is it used for? — americangeosciences.org. https://www.americangeosciences.org/critical-issues/faq/what-lidar-and-what-it-used. [Accessed 25-Apr-2023].

[19] Google. Firebase — Google's Mobile and Web App Development Platform — firebase.google.com. https://firebase.google.com/?hl=es-419. [Accessed 13-Jul-2023].

[20] Google. Firebase Authentication — firebase.google.com. https://firebase.google.com/docs/auth?hl=es-419. [Accessed 23-Jul-2023].

[21] Google. Firebase Realtime Database — firebase.google.com. https://firebase.google.com/docs/database?hl=es-419. [Accessed 13-Jul-2023].

[22] Stefan Hagen. RFC ft-ietf-geojson: The GeoJSON Format — datatracker.ietf.org. https://datatracker.ietf.org/doc/html/rfc7946#appendix-A. [Accessed 17-May-2023].

[23] Apple Inc. About swift. https://www.swift.org/about/. [Accessed 25-Apr-2023].

[24] PhD Jeff Sauro. 10 Things To Know About The Single Ease Question (SEQ) x2013; MeasuringU — measuringu.com. https://measuringu.com/seq10/. [Accessed 18-Jul-2023].

[25] PhD Jeff Sauro. 5 Ways to Interpret a SUS Score x2013; MeasuringU — measuringu.com. https://measuringu.com/interpret-sus-score/. [Accessed 18-Jul-2023].

[26] PhD Jeff Sauro. About x2013; MeasuringU — measuringu.com. https://measuringu.com/about/. [Accessed 18-Jul-2023].

[27] Kontakt. What is beacon and how does it work? BLE Bluetooth Beacons Technology Guide — Kontakt.io — kontakt.io. https://kontakt.io/what-is-a-beacon/#small-radio-transmitter. [Accessed 25-Apr-2023].

[28] Lazarillo. Healthcare Facilities - Lazarillo — lazarillo.app. https://lazarillo.app/healthcare/. [Accessed 02-May-2023].

[29] Lazarillo. Lazarillo - Accessible GPS — apps.apple.com. https://apps.apple.com/us/app/lazarillo-accessible-gps/id1139331874. [Accessed 02-May-2023].

[30] Lazarillo. Lazarillo App - Assistente inteligente de orientación e información — lazarillo.app. https://lazarillo.app/es/. [Accessed 02-May-2023].

[31] Lazarillo. LazarilloApp GPS Accesible - Apps en Google Play — play.google.com. https://play.google.com/store/apps/details?id=com.lazarillo&hl=es_CL&gl=US&pli=1. [Accessed 02-May-2023].

[32] Lazarillo. Museums - Lazarillo — lazarillo.app. https://lazarillo.app/museums/. [Accessed 02-May-2023].

[33] Lazarillo. Nuevo iPad Pro con Escáner LiDAR compatible con trackpad — apple.com. https://www.apple.com/cl/newsroom/2020/03/apple-unveils-new-ipad-pro-with-lidar-scanner-and-trackpad-support-in-ipados/. [Accessed 02-May-2023].

[34] Lazarillo. Shopping Centers - Lazarillo — lazarillo.app. https://lazarillo.app/shopping/. [Accessed 02-May-2023].

[35] Lazarillo. Universities - Lazarillo — lazarillo.app. https://lazarillo.app/uni/. [Accessed 02-May-2023].

[36] Leaflet. Leaflet — an open-source JavaScript library for interactive maps — leafletjs.com. https://leafletjs.com/. [Accessed 16-May-2023].

[37] MapBox. Vector tiles introduction — Tilesets — docs.mapbox.com. https://docs.mapbox.com/data/tilesets/guides/vector-tiles-introduction/. [Accessed 25-Apr-2023].

[38] michaelstonis. Model-View-ViewModel — learn.microsoft.com. https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm. [Accessed 24-Jul-2023].

[39] Mozilla. JavaScript — MDN — developer.mozilla.org. https://developer.mozilla.org/en-US/docs/Web/JavaScript. [Accessed 16-May-2023].

[40] National Oceanic and Atmospheric Administration. What is lidar? — oceanservice.noaa.gov. https://oceanservice.noaa.gov/facts/lidar.html. [Accessed 25-Apr-2023].

[41] OpenStreetMap. OpenStreetMap — openstreetmap.org. https://www.openstreetmap.org/about. [Accessed 25-Apr-2023].

[42] OpenStreetMap. Tags - OpenStreetMap Wiki — wiki.openstreetmap.org. https://wiki.openstreetmap.org/wiki/Tags. [Accessed 02-May-2023].

[43] PAHO. Disability - PAHO/WHO — Pan American Health Organization — paho.org. https://www.paho.org/en/topics/disability. [Accessed 21-08-2023].

[44] Will T. Measuring and Interpreting System Usability Scale (SUS) - UIUX Trend — uiux-trend.com. https://uiuxtrend.com/measuring-system-usability-scale-sus/. [Accessed 18-Jul-2023].

[45] La Tercera. La app que asiste a personas con discapacidad visual - La Tercera — latercera.com. https://www.latercera.com/piensa-digital/noticia/la-app-que-asiste-a-personas-con-discapacidad-visual/MLRRY76ULZFIVJFJEVEECYYWXM/#:~:text=El%20futuro%20de%20Lazarillo&text=Y%20a%20nivel%20mundial%20tienen,lentes%2C%20baja%20visin%20y%20ceguera. [Accessed 25-Apr-2023].

Anexo A

Código

A.1. Ejemplo de formato GeoJSON

```
{
    "type": "FeatureCollection",
    "features": [{
        "type": "Feature",
        "properties": {
            "prop0": "value0"
        },
        "geometry": {
        "coordinates":
            [
                [
                    1.3548867,
                    -1.8914409
                ],
                [
                    1.4080888,
                    -1.689265
                ]
            ],
            "type": "LineString"
        }
    }]
}
```

Listing A.1: Ejemplo de GeoJSON que representa una línea

A.2. Ejemplo de instancia de tipo `CapturedRoom` representada en formato JSON

```
{
   "windows":[

```

```
    ],
    "doors":[
       {
          "category":{
             "door":{
                "isOpen":false
             }
          },
          "confidence":{
             "medium":{

             }
          },
          "dimensions":[
             0.6133735179901123,
             1.9652961492538452,
             0
          ],
          "completedEdges":[

          ],
          "parentIdentifier":"61571A22-117C-490B-86BD-9C28A8203C19",
          "identifier":"1787D64D-CB38-411A-AE5D-3E03B05B4F84",
          "curve":null,
          "transform":[
             0.94577771425247192,
             0,
             0.32481476664543152,
             0,
             0,
             1,
             0,
             0,
             -0.32481476664543152,
             0,
             0.94577771425247192,
             0,
             1.9206036329269409,
             0.012520831078290939,
             -0.43156841397285461,
             1
          ]
       },
       {
          "category":{
             "door":{
                "isOpen":false
             }
          },
          "confidence":{
             "medium":{

             }
          },
          "dimensions":[
             0.63583070039749146,
```

```
            1.9495317935943604,
            0
         ],
         "completedEdges":[

         ],
         "parentIdentifier":"662CCEFF-C398-4D16-9FBA-124135575127",
         "identifier":"FDA9D014-CEA1-4F8C-919C-8F2B95206AA4",
         "curve":null,
         "transform":[
            -0.32481467723846436,
            0,
            0.9457777738571167,
            0,
            0,
            1,
            0,
            0,
            -0.9457777738571167,
            0,
            -0.32481464743614197,
            0,
            2.8489375114440918,
            0.0046386532485485077,
            0.90929132699966431,
            1
         ]
      },
      {
         "category":{
            "door":{
               "isOpen":false
            }
         },
         "confidence":{
            "medium":{

            }
         },
         "dimensions":[
            0.75128006935119629,
            1.9705510139465332,
            0
         ],
         "completedEdges":[

         ],
         "parentIdentifier":"01381FB7-5117-4480-A2ED-5553D601DA39",
         "identifier":"58CE3736-21DF-4A23-B630-EEE03297DDD6",
         "curve":null,
         "transform":[
            0.9457777738571167,
            0,
            0.32481452822685242,
            0,
            0,
```

```
            1,
            0,
            0,
            -0.3248145580291748,
            0,
            0.94577783346176147,
            0,
            2.5200705528259277,
            0.015148263424634933,
            0.32710933685302734,
            1
         ]
      }
   ],
   "walls":[
      {
         "category":{
            "wall":{

            }
         },
         "confidence":{
            "high":{

            }
         },
         "dimensions":[
            1.0504865646362305,
            2.354151725769043,
            0
         ],
         "completedEdges":[

         ],
         "parentIdentifier":null,
         "identifier":"662CCEFF-C398-4D16-9FBA-124135575127",
         "curve":null,
         "transform":[
            -0.3248145580291748,
            0,
            0.94577783346176147,
            0,
            0,
            0.99999994039535522,
            0,
            0,
            -0.94577771425247192,
            0,
            -0.32481452822685242,
            0,
            2.822481632232666,
            0.20694856345653534,
            0.9863242506980896,
            1
         ]
      },
```

```
    {
       "category":{
          "wall":{

          }
       },
       "confidence":{
          "high":{

          }
       },
       "dimensions":[
          0.93678385019302368,
          2.354151725769043,
          0
       ],
       "completedEdges":[

       ],
       "parentIdentifier":null,
       "identifier":"01381FB7-5117-4480-A2ED-5553D601DA39",
       "curve":null,
       "transform":[
          0.94577771425247192,
          0,
          0.32481449842453003,
          0,
          0,
          0.99999994039535522,
          0,
          0,
          -0.32481452822685242,
          0,
          0.94577783346176147,
          0,
          2.5500936508178711,
          0.20694856345653534,
          0.33742034435272217,
          1
       ]
    },
    {
       "category":{
          "wall":{

          }
       },
       "confidence":{
          "high":{

          }
       },
       "dimensions":[
          0.74824649095535278,
          2.354151725769043,
          0
```

```
        ],
        "completedEdges":[

        ],
        "parentIdentifier":null,
        "identifier":"61571A22-117C-490B-86BD-9C28A8203C19",
        "curve":null,
        "transform":[
           0.94577765464782715,
           0,
           0.32481473684310913,
           0,
           0,
           0.99999994039535522,
           0,
           0,
           -0.32481476664543152,
           0,
           0.94577771425247192,
           0,
           1.9230828285217285,
           0.20694856345653534,
           -0.43071696162223816,
           1
        ]
     },
     {
        "category":{
           "wall":{

           }
        },
        "confidence":{
           "high":{

           }
        },
        "dimensions":[
           0.58196032047271729,
           2.354151725769043,
           0
        ],
        "completedEdges":[

        ],
        "parentIdentifier":null,
        "identifier":"C6F57158-CD14-4200-BB4D-949AB4559398",
        "curve":null,
        "transform":[
           -0.32481434941291809,
           0,
           0.94577789306640625,
           0,
           0,
           0.99999994039535522,
           0,
```

```
            0,
            -0.94577783346176147,
            0,
            -0.3248143196105957,
            0,
            1.6637599468231201,
            0.20694856345653534,
            -0.82744032144546509,
            1
         ]
      },
      {
         "category":{
            "wall":{

            }
         },
         "confidence":{
            "high":{

            }
         },
         "dimensions":[
            0.52282488346099854,
            2.354151725769043,
            0
         ],
         "completedEdges":[

         ],
         "parentIdentifier":null,
         "identifier":"16F252D6-A255-4BF1-A184-142E3DB3B480",
         "curve":null,
         "transform":[
            -0.32481488585472107,
            0,
            0.94577771425247192,
            0,
            0,
            0.99999994039535522,
            0,
            0,
            -0.94577765464782715,
            0,
            -0.32481485605239868,
            0,
            2.1920099258422852,
            0.20694856345653534,
            -0.061958152800798416,
            1
         ]
      }
   ],
   "openings":[

   ],
```

```
    "objects":[

    ],
    "version":1
}
```

Listing A.2: `CapturedRoom` de habitación con 5 muros y 3 puertas detectadas.

A.3. JSON aceptable por el endpoint de creación del backend del Editor de mapas

```
{
    "layer": {
        "feature": {
            "properties": {
                "type": "polygon"
            }
        },
        "_latlngs": [
            {
                "lat": -33.36163932486686,
                "lng": -70.71295112371446
            },
            {
                "lat": -33.361735654467566,
                "lng": -70.71297258138658
            },
            {
                "lat": -33.36175133602039,
                "lng": -70.712867975235
            },
            {
                "lat": -33.36165276621289,
                "lng": -70.71284651756288
            }
        ],
        "placeKey": "-NV0NPYPoWJEWpggSAub",
        "tags": [
            {
                "k": "indoor",
                "v": "area"
            },
            {
                "k": "surface",
                "v": "grass"
            },
            {
                "k": "level",
                "v": 2
            },
            {
                "k": "floor_key",
                "v": "-NV0Ug5WWH25FgNYk87w"
```

```
            }
        ]
    }
}
```

Listing A.3: JSON que genera un polígono de cuatro vértices con la etiqueta “Área verde”

A.4. Método que convierte una superficie una lista de objetos de la misma categoría, en nodos de escena

```
func drawScannedSurfaces(surfaceType: DefaultObjects){
    guard let model = usdzModel else {
        return
    }
    let surfaceToDisplay: [CapturedRoom.Surface]? = surfaceType.
   getSurfacesArray(model: model)

    guard let surfaces = surfaceToDisplay else {
        return
    }
    let surfaceColor = surfaceType.defaultColor
    let surfaceLength = CGFloat(0.05)
    for i in 0..<surfaces.count {
        let surface = surfaces[i]
        let dimensions = surface.dimensions
        let newSurface = SCNBox(width: CGFloat(dimensions.x),
                                height: CGFloat(dimensions.y),
                                length: surfaceLength,
                                chamferRadius: 0)
        newSurface.firstMaterial?.diffuse.contents = surfaceColor
        newSurface.firstMaterial?.transparency = 0.5
        newSurface.firstMaterial?.lightingModel = .constant

        let newNode = SCNNode(geometry: newSurface)
        newNode.simdTransform = surface.transform
        newNode.name = surfaceType.rawValue

        self.sceneView.scene?.rootNode.addChildNode(newNode)
    }
}
```

Listing A.4: Método que convierte muros y entradas a un nodo gráfico con geometría laminar

A.5. Ejemplo de obtención de arreglo de coordenadas como `Vertex`

```
let coordinatesArray = nodes.map{node in
        let angle = node.rotation.w * node.rotation.y
```

```
        let leftCornerX = node.scale.x * node.boundingBox.min.x * cos(angle)
        let leftCornerY = node.scale.x * node.boundingBox.min.x * sin(angle)
        let rightCornerX = node.scale.x * node.boundingBox.max.x * cos(angle)
        let rightCornerY = node.scale.x * node.boundingBox.max.x * sin(angle)
        return (Vertex(x: leftCornerX - node.position.x, y: leftCornerY + node.position.z), Vertex(x: rightCornerX - node.position.x, y: rightCornerY + node.position.z))
    }
```

Listing A.5: Método que convierte muros y entradas a un nodo gráfico con geometría laminar

A.6. Generación de *body* de *request* para Editor de Mapas

```
func genericPostBody(_ requestObj: RequestObject?, object: DefaultObjects) -> RequestObject? {
    guard let floor = CR.shared.currentFloor, let place = CR.shared.currentPlace, let location = CR.shared.currentLocation , let requestObj = requestObj else {
        return nil
    }

    let lngConversionRate = 1.0/(111319.488 * cos(location.coordinate.latitude * .pi / 180.0))
    let sortedCoordinates = requestObj.coordinatesData
    let properties: [String: Any] = ["type" : object.mapType]
    let feature: [String: Any] = ["properties" : properties]
    let latLngsPairs = sortedCoordinates.map{node in
        let coordinate: [String: Double] = ["lat" : Double(node.y) * latConversionRate + location.coordinate.latitude,
                                            "lng" : Double(node.x) * lngConversionRate + location.coordinate.longitude]
        return coordinate
    }
    let tags = LMCM.getFullTags(level: floor.level, floorKey: floor.id, styleTag: object.customTags)
    let layer: [String: Any] = ["feature" : feature, "_latlngs" : latLngsPairs, "placeKey" : place.id, "tags" : tags]
    let json: [String: Any] = ["layer" : layer]
    requestObj.setData(data: json.jsonData)
    return requestObj
}
```

Listing A.6: Método que genera el cuerpo de una solicitud al *backend* en el formato deseado

A.7. Solución actual para obtener lista ordenada de muros

```
func sortNodesByContiguity(_ unsortedNodes: [(Vertex, Vertex)]) -> [[(
    Vertex, Vertex)]] {
     var nodes = unsortedNodes
     var polygon: [(Vertex, Vertex)] = []
     var polygonCollection: [[(Vertex, Vertex)]] = []
     var currentNode = nodes.first
     if currentNode == nil {
         return []
     }
     polygon.append(currentNode!)
     nodes.remove(at: 0)
     while(!nodes.isEmpty && currentNode != nil) {
         var idx = 0
         let nodesToCheck = nodes.count
         for node in nodes {
             if (currentNode!.1 == node.0) {
                 polygon.append(node)
                 currentNode = node
                 nodes.remove(at: idx)
                 break
             }
             else if (currentNode!.1 == node.1) {
                 let inversedNode = (node.1, node.0)
                 polygon.append(inversedNode)
                 currentNode! = inversedNode
                 nodes.remove(at: idx)
                 break
             }
             idx += 1
         }
         if idx >= nodesToCheck {
             polygonCollection.append(polygon)
             polygon = []
             currentNode = nodes.first
         }
     }
     if !polygon.isEmpty {
         polygonCollection.append(polygon)
     }
     return polygonCollection
}
```

Listing A.7: Algoritmo que genera polígonos con pares de vértices ordenados por contigüidad