UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Título: Prototipo de Asistencia Indoor dentro de un supermercado
Autores: Capecci, Eva Magalí y Mascioto, María Belén
Director: Dra. Challiol, Cecilia
Codirector: Dra. Gordillo, Silvia
Carrera: Licenciatura en Sistemas

Resumen

La mayoría de las aplicaciones móviles de asistencia indoor en supermercados son creadas ad hoc, es decir, no cuentan con un modelo asociado para definir la información de los productos o servicios que brindan. Esta es una de las motivaciones de esta tesis, y acorde a esto, se plantea un modelo de solución en el dominio de los supermercados. En el modelo propuesto, los elementos del supermercado (productos, góndolas y sectores) tienen posiciones específicas dentro del mismo. Las góndolas y los sectores son definidos con un área mientras que los productos estarán ubicados dentro de una góndola. Esto permite brindar precisión en la asistencia, por ejemplo, mostrar que un producto está en el estante dos de una góndola.

En base al modelo propuesto, se desarrolló un prototipo funcional que usa códigos QR para identificar los elementos del supermercado. El usuario es asistido para poder realizar su lista de compras, se le brindan caminos desde su posición actual hasta el siguiente elemento en su lista. Cada vez que el usuario lee códigos QR se le actualiza su posición, y en el caso de ser necesario, se recalcula su camino.

Palabras Claves

Navegación Indoor, Aplicación Móvil basada en posicionamiento, Posicionamiento, Códigos QR, PhoneGap

Trabajos Realizados

Se realizó un análisis de diferentes aplicaciones móviles de asistencia en supermercados.
Se diseñó y modeló una solución para la problemática planteada. Los elementos del supermercado se definen con posiciones específicas dentro del espacio físico, ya sea un área o un punto particular.
En base al modelo propuesto, se desarrolló un prototipo funcional usando PhoneGap (empaquetado para Android). Para identificar cada elemento del supermercado se usaron códigos QR. Se definieron puntos estratégicos conectados (espacio transitable), para a partir de estos, buscar el camino.

Conclusiones

Las tecnologías de posicionamiento en interiores existentes carecen de soporte para smartphones o de precisión suficiente. La mayoría de las aplicaciones móviles de asistencia indoor son creadas ad hoc. Esta sigue siendo un área abierta de investigación.
El modelo propuesto es flexible y extensible, esto permite poder agregar nuevas funcionalidades.
El prototipo funcional permitió observar cómo se podría llevar a cabo una posible asistencia indoor de un supermercado.

Trabajos Futuros

Analizar otro tipo de asistencia indoor, por ejemplo, beacons para evitar tanta lectura de códigos QR.
Analizar cómo se podría mejorar la asistencia a la hora de comprar productos, por ejemplo, combinado con realidad aumentada, para mostrar al usuario la góndola que ve y donde está el producto que está buscando.
El prototipo funcional por ahora está acotado a un usuario, se podría ampliar a varios usuarios y generar un front-end de gestión de datos.
Contar con back-end con datos almacenados en servidores, por ahora el prototipo tiene los datos precargados.

Fecha de la presentación: DICIEMBRE 2016

AGRADECIMIENTOS

Belén Mascioto

A mi compañera Magalí, con quien transité todo este camino en la universidad, desde el primer año hasta hoy. Excelente compañera de estudio, pero mejor aún, amiga de la vida.

A mi familia, en especial a mis padres Ana y Daniel, sin ellos no hubiese sido posible llegar hasta acá, gracias por su apoyo constante, sus consejos y su confianza.

"Todos nuestros sueños se pueden hacer realidad si tenemos el coraje de perseguirlos."

Magalí Capecci

A mis padres Marta y Oscar, que depositaron toda su confianza en mí, por el amor y apoyo incondicional desde la distancia, que me incentivaron a nunca bajar los brazos.

A mis hermanos Néstor y Gustavo, que son mi modelo a seguir, por demostrarme que los hermanos también son amigos.

A mi compañera de tesis Belén, quien confió en mí para hacer de este sueño una realidad.

A Pablo Thomas, por brindarme la llave para descubrir el mundo de la informática. Por sus consejos personales y profesionales. Por apadrinarme toda la carrera, desde el ingreso hasta hoy.

Eternamente agradecida.

Belén Mascioto y Magalí Capecci

A Cecilia y Silvia, que fueron el corazón de esta Tesina, por su compromiso y dedicación desde el momento cero, por cada consejo y palabras de motivación que nos dieron fuerzas y energías para culminar este trabajo.

Gracias.

Índice

1. INTRODUCCIÓN

1.1 Motivación

Con el avance de la tecnología móvil y los mecanismos de sensado de posicionamiento (GPS, Códigos QR, etc.) cada vez es más frecuente encontrar aplicaciones de guías móviles en diferentes dominios [Emmanouilidis et al., 2013]. Este tipo de aplicaciones tienen la particularidad de asistir al usuario en la movilidad. En entornos outdoor los servicios de mapas combinados con el GPS permiten facilitar la creación de este tipo de aplicaciones, sin embargo en entornos indoor (cerrados), como ser un supermercado, todavía es un campo de investigación. Las aplicaciones basadas en GPS son incapaces de obtener la posición en interiores debido a la dispersión y atenuación que sufre la señal del satélite en las paredes de los edificios, lo que aumenta el error de medición hasta el punto de ser ineficaz para ubicar al usuario. Por otro lado, aunque obtuvieran la posición, es necesario disponer también de un plano del edificio para obtener la posición, del que muchas veces no se dispone. Es necesario, por lo tanto, ofrecer mecanismos que permitan posicionar al usuario tanto en espacios abiertos como cerrados.

A continuación se mencionan algunos ejemplos de aplicaciones móviles indoor ya existentes, y que sirvieron como motivación para plantear esta tesis.

Los autores proponen en [López-de-Ipiña et al., 2011] una aplicación de apoyo a personas no videntes y como característica principal el sistema no se ve afectado por los patrones de compras convencionales, lo que permite que una persona con discapacidad visual utilice la misma aplicación que una persona vidente. Creemos que es un puntapié para que uno como desarrollador al diseñar cualquier tipo de sistema tenga en cuenta la accesibilidad del usuario y garantizar así que la aplicación sea para un público más amplio.

Otra problemática abordada en [Waltner et al., 2015], está relacionada a cómo las enfermedades cardiovasculares constituyen un problema de salud pública y cómo los estilos de vida saludables pueden a largo plazo disminuir o eliminar en la sociedad éstas enfermedades haciendo hincapié que la alimentación es pieza clave para la prevención o recuperación de dicha enfermedad. Los autores sostienen, en [Waltner et al., 2015], que las decisiones tomadas individualmente en cada dieta están directamente relacionadas con las decisiones que se toman al momento de realizar la compra de alimentos. La finalidad del sistema es brindar al usuario una aplicación con reconocimiento de imagen en tiempo real y junto a ella información nutricional para poder seguir así una dieta saludable. Destacamos la importancia de realizar una aplicación que no sólo brinde información al usuario y de soporte para realizar su compra, sino que tenga impacto a la hora de la toma de decisiones para contribuir en cuestiones de salud.

No sólo los beneficios son para los usuarios, también es importante el enfoque comercial. En [Bai et al., 2011] se destaca cómo la prestación de servicios personalizados a los clientes puede mejorar el rendimiento del negocio. Se explica que las organizaciones que deseen proporcionar servicios avanzados o con fines publicitarios, como en aeropuertos, hoteles y centros comerciales, a menudo proporcionan acceso gratuito a Wi-Fi para atraer clientes. En este caso, se utilizan sistemas de seguimiento pasivos basados en Wi-Fi para el seguimiento de los clientes y recopilación de información (por ejemplo, información de la ubicación y el comportamiento de compra). La información que es directamente relevante para los intereses del cliente se basa en un reconocimiento fiable de su ubicación y el contexto de la misma.

De manera similar en [Yang et al., 2008] se propone un sistema de recomendación de ubicación adaptable a las necesidades de compra de un cliente con ofertas y promociones de proveedores o vendedores dependientes de la ubicación. En [Yang et al., 2008] la idea básica es recomendar productos de acuerdo a las preferencias de los clientes. Esto contribuye al aumento de las compras y puede verse como estrategia de venta a favor del negocio.

La mayoría de las aplicaciones móviles de asistencia indoor son creadas ad hoc, es decir, no cuentan con un modelo de solución asociado para definir toda la información de los productos o servicios que brindan. Esto es una de las motivaciones de esta tesis, y acorde a esto, se plantea en la misma un modelo de solución en el dominio de los supermercados.

Por otro lado, la navegación indoor viene siendo estudiada en los últimos años por diferentes autores, por ejemplo, [Zlatanova et al., 2013] y [Jung and Lee, 2015]. En estos trabajos se analiza cómo representar el espacio indoor. En esta tesis el espacio físico del supermercado se representará usando un área que delimitará dónde están ubicados los elementos del supermercado, como pueden ser los productos, las góndolas y los sectores. Tanto las góndolas como los sectores ocuparán un área dentro del supermercado, mientras que los productos estarán ubicados dentro de una góndola particular. Esto permitirá brindar precisión en la asistencia para que un usuario encuentre, por ejemplo, un producto en el estante dos de la góndola. En esta tesis no se realizarán cálculos de caminos complejos, sino que la indicación de cómo moverse por el espacio se brindará mediante imágenes que sirvan para determinar dónde está actualmente el usuario y a dónde tiene que ir para realizar el recorrido, y así encontrar, por ejemplo, el siguiente producto a comprar.

1.2 Objetivos

El objetivo principal de esta tesina es proponer un modelo de solución para asistir al usuario dentro de un supermercado. En particular, este modelo brinda asistencia a los clientes a la hora de realizar su compra. Para lo cual, se modelaron los posibles elementos (por ejemplo: productos, góndolas y sectores) que forman parte de un supermercado. El modelo contempla la posibilidad de generar recorridos personalizados, por este motivo, el posicionamiento tanto del usuario, como de los elementos del supermercado juega un papel fundamental dentro de este modelo, y de los servicios de asistencia que se van a brindar.

Usando como base el modelo propuesto, se realizó un prototipo con los datos de un supermercado hipotético. Este prototipo fue implementado como una aplicación móvil usando *PhoneGap*[1] con plataforma *Android* [2]. Dicho prototipo permite al usuario, de manera sencilla, seleccionar productos para armar su lista de compras, obtener información y/o ubicación exacta de los mismos. También permite ubicar góndolas o sectores dentro del supermercado. A partir de la lista de compras, el prototipo brinda, por ejemplo, la posibilidad de generar recorridos para efectuar la compra según alguna preferencia seleccionada. A medida que vaya comprando productos, se le indica al usuario dónde está ubicado el siguiente producto a comprar. Todos los elementos del supermercado cuentan con un código QR[3] ("*Quick Response*") que permite determinar dónde está ubicado cada elemento. Estos códigos QR son usados por el prototipo para indicar, por ejemplo, donde está actualmente posicionado el usuario, o para indicarle si desea comprar el producto asociado al código QR leído.

[1] Página Oficial de *PhoneGap*: http://phonegap.com (Último Acceso: 12/3/2016)
[2] Página Oficial de *Android*: http://developer.android.com/index.html (Último Acceso: 31/3/2016)
[3] Código QR: http://www.codigos-qr.com/en/ (Último Acceso: 31/3/2016)

1.3 Estructura

La estructura de la tesis se describe a continuación. En el Capítulo 2, se presentan trabajos de aplicaciones relacionadas y se describe de manera general las tecnologías utilizadas en cada uno de ellos, para mostrar así diferentes tipos de asistencias en espacios cerrados.

En el Capítulo 3, se describe la problemática que se quiere resolver usando como recurso diferentes pantallas esquemáticas para ayudar al lector a entender mejor el dominio. Luego, se presenta el modelo propuesto que da solución a dicha problemática.

El prototipo implementado en base al modelo propuesto es presentado en el Capítulo 4. En este capítulo se describen las decisiones de implementación tomadas para dar solución a la problemática, por ejemplo, cómo realizar la búsqueda de caminos dentro del supermercado. Además, se describen como se usó la plataforma *PhoneGap* (empaquetada para *Android*) para lograr tener el prototipo funcional.

En el Capítulo 5, se describe un posible caso de uso del prototipo propuesto, donde se muestra cómo el usuario se puede ir moviendo por el supermercado para así ir realizando sus compras.

En el Capítulo 6, se presentan las conclusiones que se obtuvieron al final de este trabajo y se mencionan posibles trabajos futuros.

2. BACKGROUND

En este capítulo se presentan algunas aplicaciones con diferentes características, las cuales sirvieron de base para analizar diferentes servicios que pueden estar presentes tanto en el dominio de un supermercado como a nivel de asistencia dentro del mismo.

- **Indoor Navigation and Product Recognition for Blind People Assisted Shopping [López-de-Ipiña et al., 2011]**

  El proyecto *BlindShopping* propuesto en [López-de-Ipiña et al., 2011] consiste en una aplicación de apoyo a personas no videntes para que puedan adquirir autonomía al realizar sus compras. Sin la ayuda de otra persona es imposible llegar a una sección del supermercado específica y encontrar ciertos productos. A pesar de eso, una vez ahí, ellos no pueden identificar los productos y sus características (por ej., el precio, la marca, la fecha de vencimiento) para decidir cuáles son las características de cada una de las cosas y decidir si quieren comprarlo o no.

  La solución planteada en [López-de-Ipiña et al., 2011] ofrece soporte tecnológico para personas ciegas para que puedan comprar sin tener que alterar los patrones de compras convencionales, lo que permite que una persona con discapacidad visual utilice la misma aplicación que una persona vidente, diseñado para evitar que tengan accesorios adicionales que implican ir al supermercado con algo costoso y pesado, tanto en precio como en tiempo, en la instalación o mantenimientos de cada uno de estos procesos. En la Figura 2.1 se puede apreciar una prueba de uso del proyecto mencionado. Los autores usaron diferentes celulares para hacer las pruebas como se muestra en la Figura 2.2.

Figura 2.1: (Izquierda) Sistema de navegación usado, (medio) Reconocimiento de códigos UPC, (Derecha) reconocimiento de código QR [López-de-Ipiña et al., 2011].

Figura 2.2: Los celulares Motorola Mileston y Nokia 6131 usados por [López-de-Ipiña et al., 2011].

Las cuestiones que tiene en cuenta *BlindShopping* en cuanto a la organización del supermercado son las siguientes. Primero, considera que todos los productos están

agrupados en diferentes categorías (por ejemplo: *bebidas*), y estas se dividen en tipos de productos (por ejemplo: *bebidas/cola*) que a su vez están divididas en marcas (por ejemplo: *Lata de Pepsi*). Además de estas consideraciones, el área del supermercado está dividida en sectores de celdas de dos tipos: una celda que contiene las góndolas y otra que contiene los pasillos. Internamente los mapas de *BlindShopping* tienen el ID del RFID en cada una de las celdas de navegación e información de posicionamiento, por ejemplo, el tipo obtenido de una celda o la ubicación de sus celdas vecinas.

BlindShopping ofrece soporte infraestructural para cada proceso de compra dentro del supermercado, que se define en cuatro pasos de un proceso cíclico: 1) navegación de la categoría del producto 2) búsqueda del producto 3) identificación del producto y 4) selección del producto. El ciclo se detiene cuando el usuario decide ir a pagar, en cualquiera de los puntos. Consecuentemente *BlindShopping* ofrece una navegación a través de mensajes de voz guiando al usuario, para llevarlo donde el smartphone identificó que el producto está ubicado. Una vez ahí, *BlindShopping* ofrece soporte para que cada producto pueda ser reconocido a través de una identificación ya sea mediante el escaneo de códigos QR o UPC. Tener en cuenta que los códigos QR redundantes permiten el reconocimiento eficiente, incluso cuando la persona ciega pasa lentamente la cámara sobre el código que se lee.

A pesar de lo descripto anteriormente, *BlindShopping* no ofrece una solución técnica para cada compra asistida, pero ofrece una solución accesible, usable y de fácil acceso para personas ciegas. Veamos más detalles de cada uno de estos conceptos:

- *Accesible* ya que el costo de la plataforma es relativamente reducida. El supermercado debe instalar un servidor, una red Wi-Fi en el área del shopping. Además de esto, los tags del RFID deben ser distribuidos a través del piso (como se mostró en la Figura 2.1). Por otro lado, los usuarios deben contar con un smartphone, dispositivo que hoy en día es de uso masivo. La adquisición más significativa sería adquirir los lectores RFID que van a estar en cada uno de los bastones blancos de las personas ciegas que el supermercado comprará y prestará a sus clientes.

- *Utilizable* ya que la aplicación está adaptada de manera que las personas ciegas pueden ser independizadas por el uso de la misma. Esto incluye gestos o una interfaz basada en voz, que le permite a la persona poder identificar algún tipo de funcionalidad distinta a través de un dibujo en la pantalla del smartphone con el dedo o a través de algún comando de voz (como se mostró en la Figura 2.2, al dibujar una "*P*"). El resto de la interacción se realiza a través de reconocimiento y síntesis de voz. El usuario dice a qué sección del supermercado quiere ir y el sistema utilizando síntesis de voz le indica la ruta para llegar o información del producto. Una vez que el usuario se encuentra frente a la sección del producto (por ejemplo: productos diarios/leche), puede localizar productos específicos apuntando con la cámara del celular en la dirección donde se reconoce el código 2D (por ejemplo: *leche descremada Carrefour*) y es ubicado en el estante de la góndola. Estos códigos son utilizados en etiquetas de plástico, que incluyen el nombre y precio de cada producto, distribuidos en los estantes del supermercado.

- *Fácilmente gestionable*, ya que la navegación *BlindShopping* y el tipo de producto es administrada a través de un cliente web (como se puede apreciar en la Figura 2.3) ofreciendo una interface intuitiva para configurar el sistema de navegación basado en etiquetas RFID y los códigos QR asignados en cada tipo producto delante de cada una de las góndolas.

Figura 2.3: Herramienta de *BlindShopping* [López-de-Ipiña et al., 2011].

La Figura 2.4 muestra cómo está distribuida la arquitectura de solución, compuesta de los siguientes tres componentes:

1. *Navigation system* – Sistema de navegación, encargado de guiar a la persona ciega dentro del supermercado, brindándole por medio de auriculares conectados a su smartphone instrucciones de navegación verbales simples. Dicho componente combina un bastón blanco con un lector portátil RFID unido a su punta, un conjunto de etiquetas RFID distribuidas en los pasillos del supermercado (ver parte izquierda de la Figura 2.1) y una aplicación en el teléfono procesando las lecturas RFID mediante Bluetooth y la generación de órdenes verbales de navegación del usuario como resultado.
2. *Product recognition* – Reconocimiento del producto. Una vez que el usuario llega a la sección del producto que está buscando, toma el producto o apunta con la cámara del celular buscando el código QR o UPC que está en cada sección de cada góndola donde hay diferentes productos. La cámara del celular reconoce tal código y luego informa verbalmente las características de dicho producto. Notar que el código QR puede codificar hasta 4296 caracteres alfanuméricos, y la redundancia que presenta tiene una lectura exitosa, la lectura se puede realizar incluso cuando se capturan imágenes parciales.

3. *System management* – Gestión del sistema. *BlindShopping* incluye una vista web para *BlindShopping*, RFID y gestión de la infraestructura de código QR. Esto permite el registro de la colección de etiquetas RFID distribuidas en el suelo del supermercado y los códigos QR que están en cada estante de la góndola. Para todo esto, ofrece RIA (*Rich Internet Application*), una interfaz muy utilizable para cualquier navegador web sin tener algún conocimiento técnico específico.

Figura 2.4: Arquitectura distribuida de *BlindShopping* [López-de-Ipiña et al., 2011].

- **A Mobile Shopping Assistant to Support Product Domesticity in Consumer Decisions [Stamopoulos et al., 2014]**

  En [Stamopoulos et al., 2014], los autores proponen y desarrollan una aplicación móvil para alcanzar o generar interés en los usuarios acerca de productos específicos, brindando información acerca de cada producto, focalizando en el país de producción y el origen de la compañía que lo produce. Durante las compras, el usuario recibe información acerca de cada producto, escaneando los códigos de barra de los mismos con un dispositivo móvil. Además, se brinda recomendaciones de productos similares basados en el origen de cada producto. Los resultados muestran que el sistema puede afectar significativamente el proceso de decisión de los usuarios.

  El sistema adopta el modelo cliente-servidor y consiste de dos partes principales que se muestran en la Figura 2.5. La aplicación móvil se instala en un dispositivo móvil y se puede conectar a un servidor a través de una conexión de Internet HTTP. El servidor mantiene la base de datos central donde se almacena la información que relaciona compañías, productos y código de barra de productos. La base de datos central está implementada en MySQL. Para cada compañía y producto se almacenan los detalles básicos, información del origen de la compañía y lugar de elaboración de cada producto. El mismo servidor también mantiene el sistema de administración de base de datos de productos (PDBMS), en el cual solo el administrador tiene acceso. El PDMS está implementado como una aplicación web en PHP y los administradores pueden acceder remotamente y administrar el sistema desde cualquier computadora.

Figura 2.5: Principales partes del sistema [Stamopoulos et al., 2014].

La aplicación móvil también contiene una copia local de la base de datos central (como puede verse en la Figura 2.5), la cual puede ser actualizada cuando se inicia la aplicación si el usuario está conectado a Internet. La base de datos local está implementada en *SQLite*. Para los productos que están presentes en la base de datos local, la aplicación móvil puede generar reportes con los datos almacenados relevantes. La aplicación puede seguir funcionando con la copia de la base de datos cacheada localmente, cuando la conectividad a Internet no esté disponible. Sin embargo, cuando el usuario inicia la aplicación y la conectividad a Internet esté disponible la base de datos local puede ser sincronizada con la base de datos central para descargar la información actualizada.

En el caso que un usuario escanee un producto que no esté registrado en la base de datos, la aplicación le sugerirá que llene un formulario con la información del producto y que lo registre en la base de datos del dispositivo móvil. Luego, el nuevo registro es actualizado en la base de datos central en la próxima sincronización. El administrador es responsable de crear, actualizar y eliminar información de la base de datos central. También es responsable de chequear la validez y la correctitud de la base de datos del servidor y aprobar la información suministrada por los usuarios, con la opción de modificarlos antes de guardarlos en la base de datos central.

La interface de la aplicación móvil se puede apreciar en la Figura 2.6, el menú principal de la aplicación consiste en 4 botones: "*Scan*!", "*Companies*", "*Products*" y "*Update*". En esta figura se puede apreciar lo que pasa al escanear un producto, el cual puede estar o no en el sistema.

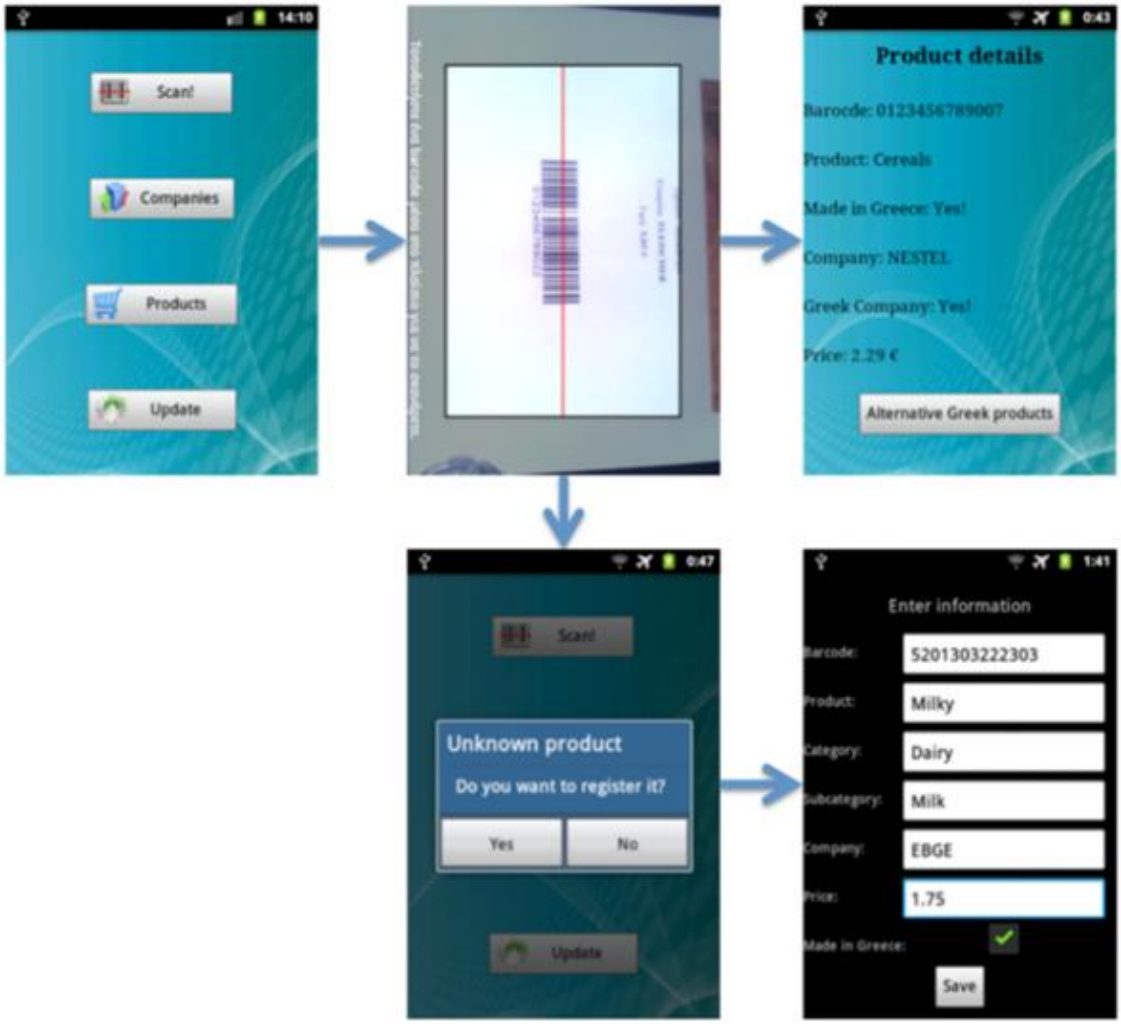


Figura 2.6: Pantallas del sistema [Stamopoulos et al., 2014].

Al seleccionar el botón "*Scan*!" (Figura 2.6) se invoca a la aplicación Android integrada "*Barcode Scanner*" y se usa la cámara del dispositivo móvil como un escáner de código de barra. Así cuando los usuarios apunten con la cámara al código de barra del producto deseado, la aplicación mostrará información acerca del mismo. En caso de que el usuario escanee un producto que no esté registrado en la base de datos local (como se puede ver en la Figura 2.6), la aplicación notificará que el producto no está registrado y preguntará si desea registrarlo. Si el usuario quiere registrarlo entonces tiene que llenar un formulario con la información del producto, insertando el nombre del producto seleccionando, una categoría y subcategoría a la cual pertenece. El usuario también selecciona el nombre de la compañía de fabricación desde la lista de compañías existentes y también provee su opinión personal, si el producto es elaborado a nivel nacional. Luego, apretando el botón "*Save*", se guarda el nuevo registro en la base de datos del dispositivo móvil. Además, el usuario puede agregar una nueva compañía en caso que no exista en la lista de compañías. Luego del escaneo y basado en el tipo de producto que puede existir o fue registrado como nuevo, si el producto no está fabricado a nivel nacional, o la compañía que lo produce no es nacional, la aplicación puede proponer al usuario cualquier producto nacional similar como alternativa que esté registrado en la base de datos.

Al seleccionar el botón "*Companies*" (de la Figura 2.6), se listan los nombres de las compañías registradas en la base de datos central, al lado de las compañías nacionales, se ha agregado la imagen de la bandera del país, para que sea más fácil para el usuario

diferenciarlos en la lista. Al seleccionar una compañía, el usuario puede navegar las categorías en las cuales la compañía tiene productos, luego a las subcategorías de productos relevantes y finalmente a los productos junto con sus detalles de cada subcategoría. También se cuenta con un campo de búsqueda en caso que el usuario quiera encontrar una compañía por nombre en lugar de buscarla en la lista.

Al seleccionar el botón "*Products*" (de la Figura 2.6) se listan todas las categorías de productos disponibles (Figura 2.7). Al seleccionar una categoría, el usuario puede navegar a sus subcategorías y a los productos como al detalle de producto de cada subcategoría. También cuentan con un campo de búsqueda en caso que el usuario quiera encontrar un producto por nombre en lugar de buscarlo en la lista.

Figura 2.7: Ejemplos de búsquedas [Stamopoulos et al., 2014].

Finalmente al seleccionar el botón "*Update*" (de la Figura 2.6), la aplicación móvil verifica las redes disponibles. En caso que el dispositivo móvil esté conectado a una red con accesos a Internet, la aplicación se comunica con el servidor que almacena la base de datos central y actualiza su base de datos local.

Al mismo tiempo, si el usuario ha ingresado manualmente algún producto o compañía en la aplicación móvil, los registros son actualizados en el servidor que almacena la base de datos central. Después de eso, el administrador del sistema puede verificar la correctitud y validez de los registros actualizados e insertarlos en las tablas primarias de la base de datos central o eliminarlas. En caso de que no haya una red disponible, la aplicación informa al usuario que el proceso de actualización no fue completado con éxito.

- **Where Am I?: Studying Users' Indoor Navigation Location Needs [Muralidharan et al., 2014]**

  La principal contribución del paper [Muralidharan et al., 2014] es identificar la precisión de la ubicación de los usuarios para navegar en ambientes indoor. Los autores presentan un caso de estudio en dos edificios de la ciudad de *Singapur* para determinar el nivel de precisión de seguimiento de la ubicación que los usuarios típicos necesitan para navegar desde su ubicación actual a una sala específica dentro del edificio. Para esta tarea, considerada análoga al problema de navegación a un local en un shopping, los autores encontraron que los usuarios necesitan su ubicación actual en un mapa para hacer un seguimiento continuo con una precisión de ±1 local (teniendo como promedio el frente de un local 11.6mts).

El enfoque del experimento presentado en [Muralidharan et al., 2014] trata de entender la precisión de la ubicación para una navegación exitosa y libre de complicaciones en una ambiente indoor. Para esto variaron la precisión de la posición del usuario mostrada en una herramienta de navegación indoor y midieron el tiempo que le implicó al usuario llevar a cabo una simple tarea de navegación. El argumento usado fue que cuanto más rápido el usuario llegó a destino más útil fue la granularidad de la ubicación provista.

Para validar efectivamente la precisión de la ubicación necesaria para usuarios finales en la navegación usaron 4 variantes de precisión, como se puede apreciar en la Figura 2.8.

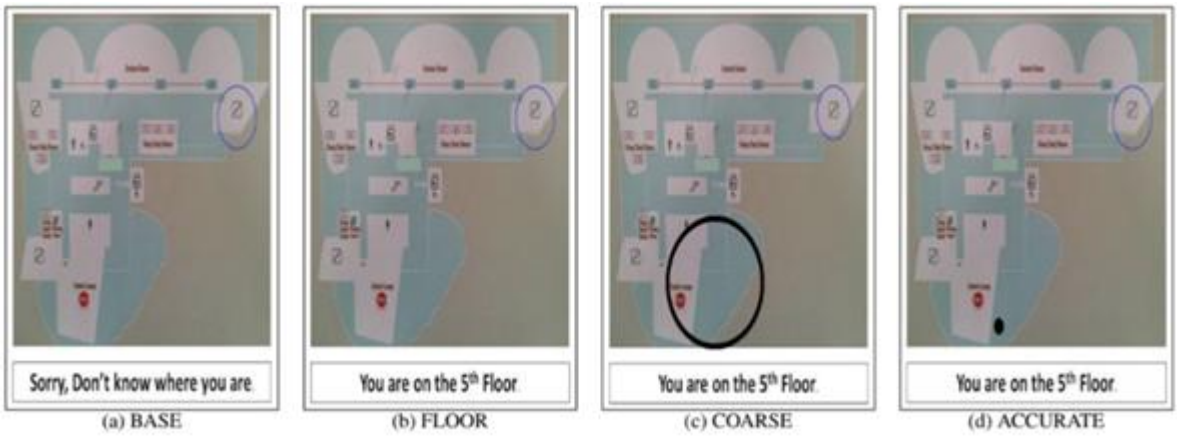


Figura 2.8: Variante de precisión [Muralidharan et al., 2014].

Veamos más nivel de detalle de la Figura 2.8. La primer variante, BASE (Figura 2.8a) muestra el mapa de cada piso del shopping sin informar al usuario en qué piso se encuentra. Esta línea base es representativa de las opciones disponibles con estado actual de las aplicaciones, como *Fast mall* y *Mall Maps* que proveen a los usuarios finales planos de los pisos de un shopping y un directorio de tiendas en sus dispositivos móviles. La segunda variante, FLOOR (Figura 2.8b) muestra en qué piso están los usuarios. La tercera y cuarta variante, COARSE (Figura 2.8c) y ACCURATE (Figura 2.8d) respectivamente, muestra la ubicación del usuario con precisión ± 1 local y un local exacto respectivamente.

Del estudio los autores observaron que los usuarios son bastante ingeniosos y pueden utilizar una precisión de ubicación de grano grueso cuando se considera una tarea común de ubicar una tienda en un shopping. Han observado también que un mayor nivel de precisión de la ubicación indoor provee resultados decrecientes y que las soluciones que proveen niveles prácticos de precisión son suficientes para las necesidades de la mayoría de los usuarios.

- ***inShopnito*: An Advanced yet Privacy-Friendly Mobile Shopping Application [Put et al., 2014]**

  En [Put et al., 2014] se plantea que las *Aplicaciones Móviles de Shopping*, (MSAs, por su sigla en inglés *Mobile Shopping Applications*) están ganando popularidad rápidamente. Estas mejoran la experiencia de compra ofreciendo recomendaciones personalizadas o incorporando programas de lealtad de clientes. Aunque las MSAs son efectivas y atraen nuevos consumidores, tienen varios defectos. La recolección de datos involucrada en MSA y la ausencia de transparencia en esto son conceptos importantes para muchos

clientes. En [Put et al., 2014] los autores presentan *inShopnito*, una aplicación móvil de compra que preserva la privacidad. Todas las transacciones realizadas en *inShopnito* son anónimas. Sin embargo, el sistema sigue ofreciendo las características esperadas de las MSA modernas. Además, la MSA puede sugerir recomendaciones personalizadas aunque el comerciante no construya perfiles de clientes. Estos perfiles son gestionados en el smartphone y pueden ser parcialmente expuestos para recibir mejores recomendaciones personalizadas. Los autores demuestran que es posible tener una MSA que preserve la privacidad sin sacrificar practicidad.

En *inShopnito* se incorpora una serie de tecnologías de seguridad y de mejora de privacidad dentro de una MSA. Los vendedores pueden recompensar a los clientes con puntos de lealtad anónimos y vouchers electrónicos. Además los datos transaccionales son almacenados en forma segura y gestionados en el mismo smartphone, por lo que los perfiles son locales y controlados por los clientes. Los usuarios pueden optar por exponer parte de su perfil anónimo para obtener recomendaciones personalizadas del sistema de recomendación sin tener que sacrificar su personalidad. Además, la integridad de la información revelada es protegida por *inShopnito*, por lo tanto la recolección de datos de los proveedores es no solo menos sensible sino también más precisa. Adicionalmente, *inShopnito* puede ser usado para combinar los programas de lealtad de clientes CLPs de múltiples proveedores para facilitar la cooperación. Finalmente, el sistema ayuda a los vendedores a cumplir con las leyes de privacidad, porque garantiza que los datos sean menos sensibles.

Como se mencionó anteriormente, *inShopnito* es una MSA que permite a los clientes y vendedores obtener las ventajas de las plataformas móviles sin sacrificar la privacidad de los clientes. Para lograr este objetivo *inShopnito* cuenta con los siguientes componentes principales (Figura 2.9): administración de credenciales anónimas, esquemas de lealtad y vouchers que preservan privacidad, recomendaciones que preservan privacidad y mecanismos de almacenamiento seguro. Gracias a estos mecanismos, *inShopnito* brinda soporte a sesiones de shopping que son anónimas por defecto; de esta forma, se reducen las preocupaciones de seguridad asociadas a la recolección de datos de los vendedores. Con *inShopnito*, los clientes tienen un mejor control sobre sus datos personales, ya que la información de sesión es almacenada de forma segura en sus smartphones (en lugar de la base de datos de los vendedores).

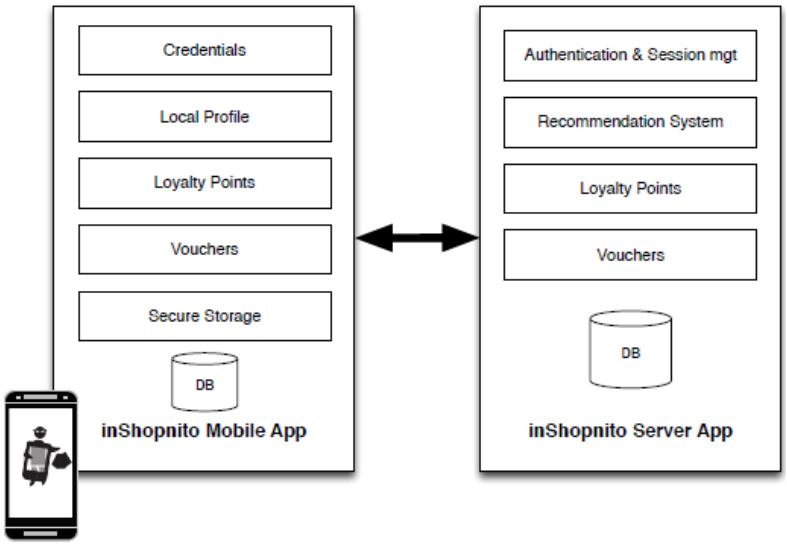


Figura 2.9: Componentes de *inShopnito* [Put et al., 2014].

inShopnito se focaliza en los mecanismos de seguridad usados en los siguientes pasos de la sesión de compra por los clientes durante su visita a una tienda de un vendedor: *Registración*, *Inicio de la aplicación*, *Conexión con el servidor de inShopnito*, *Abrir una sesión anónima*, *Shopping* (y recomendaciones) y *Checkout* (ver Figura 2.10)

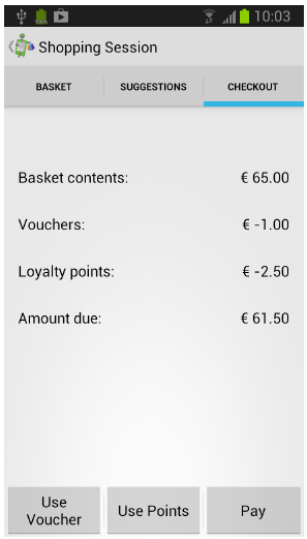


Figura 2.10: Pantalla de ckeckout en *inShopnito* [Put et al., 2014].

- **A location-aware recommender system for mobile shopping environments [Yang et al., 2008]**

  En [Yang et al., 2008] los autores proponen un sistema de recomendación de reconocimiento de ubicación que se adapte a las necesidades de compra de un cliente con ofertas y promociones de proveedores dependientes de la posición. Específicamente, se propone un sistema de recomendación para recomendar páginas web de los vendedores - incluyendo ofertas y promociones- a los clientes interesados.

  Según [Yang et al., 2008], los sistemas de detección de posicionamiento entran en dos categorías: los métodos centralizados, donde la infraestructura consiste en receptores desplegados en algunos lugares con usuarios finales. La ubicación de los clientes se determina en los servidores. En contraste, los métodos descentralizados, la infraestructura consiste en "*beacons*" desplegados en algunos lugares señalizando a los clientes su ubicación. Por lo tanto, la ubicación de los clientes es determinada y almacenada en el dispositivo móvil. Comparado con el enfoque centralizado, el enfoque descentralizado brinda a los usuarios finales mejores opciones sobre si quieren revelar su ubicación a otros usuarios o no. Debido a estos aspectos de seguridad los autores han adoptado el enfoque descentralizado por lo cual los usuarios, solo envían datos de su ubicación cuando necesiten servicios de recomendación.

  La arquitectura general del sistema presentado en [Yang et al., 2008] se muestra en la Figura 2.11.

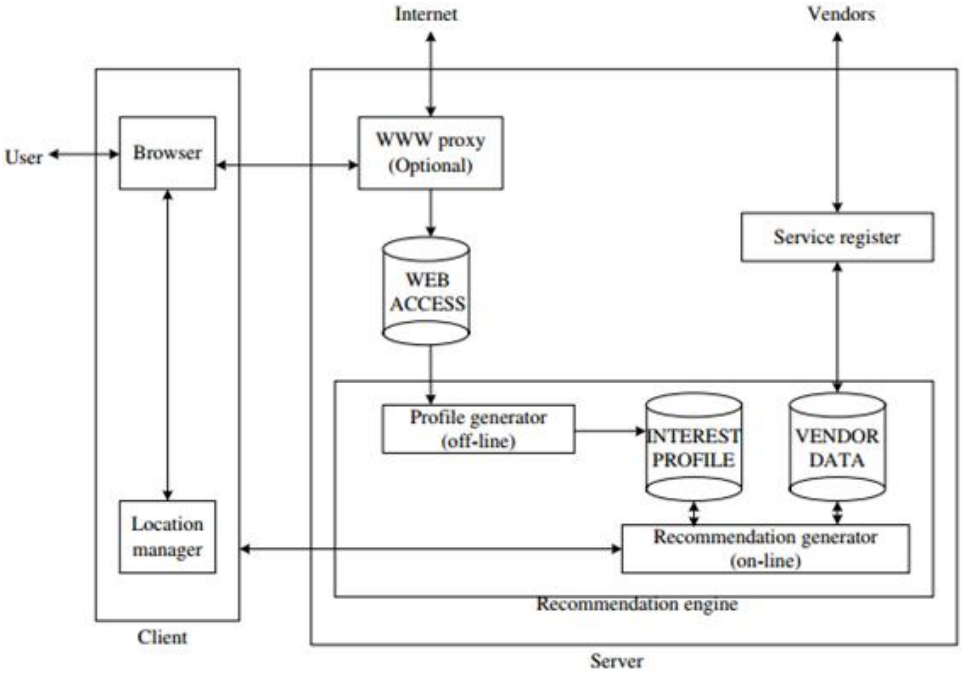


Figura 2.11: Arquitectura general del sistema presentado en [Yang et al., 2008].

Como se puede apreciar en la Figura 2.11, del lado del cliente comprende dos componentes: un browser (navegador) de Internet estándar y un "*location manager*" (administrador de ubicación) que estima la posición del cliente. Cuando un cliente necesita un servicio de recomendación se manda un pedido con la posición del mismo al servidor. Del lado del servidor, el principal componente es el "*recommendation engine*" (motor de recomendaciones) que consiste en subsistemas offline y online. El subsistema offline mantiene una base de datos llamada WEB ACCESS (acceso web) que registra datos acerca de las páginas webs visitadas por cada cliente. Este subsistema también analiza los datos registrados y los deriva al perfil de intereses de cada usuario. El subsistema online mantiene los perfiles de los clientes más una base de datos llamada VENDOR DATA, conteniendo información de los vendedores (nombre, dirección, página web) que es directamente registrada por el vendedor. Cuando se recibe un pedido de servicio, el subsistema online genera una lista de posibles páginas web basadas en el perfil de intereses del cliente, datos del vendedor y la posición actual del cliente provista por el administrador de posiciones (*location manager*).

En la Figura 2.12 se puede ver una captura del subsistema del lado del cliente. Después de mandar un pedido de servicio, el cliente recibe una lista de recomendación conteniendo links a varios vendedores (del lado derecho del cuadro). En un mapa se marcan las posiciones de los clientes y los vendedores recomendados. Cuando se hace click en el link de un vendedor específico se muestra la página web correspondiente en el cuadro de abajo.

Figura 2.12: Pantalla del Cliente [Yang et al., 2008].

- **MANGO - Mobile Augmented Reality with Functional Eating Guidance and Food Awareness [Waltner et al., 2015]**

  El proyecto MANGO presentado en [Waltner et al., 2015], desarrolla un sistema de asistencia móvil brindando información a través de Realidad Aumentada (RA) para dar soporte al usuario durante las compras de alimentos diarios, con una dieta moderna, el usuario es aconsejado de qué fruta o verdura comprar de acuerdo con su perfil. Usando un sistema de reconocimiento basado en imagen la aplicación automáticamente clasifica la comida capturada por cámara usando la metodología de visión computarizada y aprendizaje de máquina. El usuario puede decidir mostrar información nutricional adicional con propuestas alternativas. La aplicación puede reconocer clases de comidas en tiempo real, bajo condiciones de compra en mundo real, y asociar recomendaciones de dieta usando asistencia con realidad aumentada.

  Acorde a lo mencionado, el proyecto presentado en [Waltner et al., 2015] tiene como objetivo el desarrollo de una aplicación móvil con asistencia nutricional basada en recomendaciones a través de *Realidad Aumentada*. La información de comida y la asistencia de recomendaciones proveen una interfaz intuitiva con reconocimiento de comida basado en video. Después de escanear un alimento el resultado del reconocimiento se muestra en la pantalla junto con alternativas nutricionales asociadas a una barra de clasificación que muestra qué tan bien ese alimento encaja en el perfil de la dieta del usuario. Al clickear un resultado se muestra información detallada, incluyendo micronutrientes junto con su correspondiente información nutricional, como además recomendaciones alimenticias de acuerdo al perfil del usuario (como se puede apreciar en la Figura 2.13).

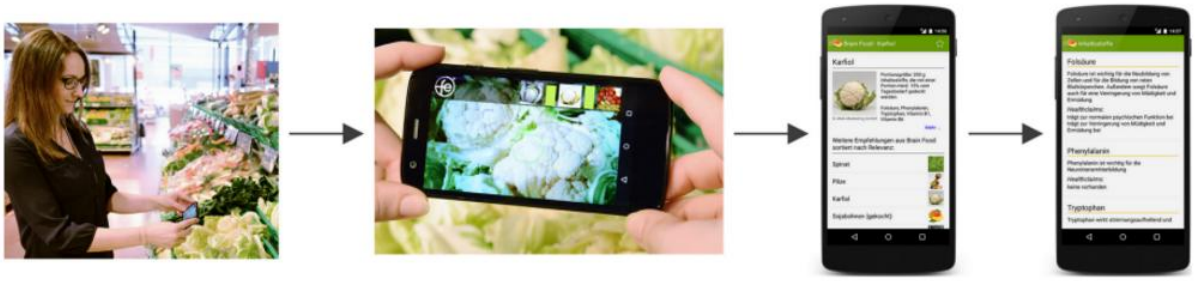


Figura 2.13: Interface de la aplicación móvil [Waltner et al., 2015].

Completando el reconocimiento de alimentos basado en imagen, la aplicación permite búsquedas a través de una base de datos de alimentos creada especialmente para esta aplicación en el contexto global de la alimentación funcional. Los usuarios también pueden leer una introducción acerca de los lineamientos del concepto nutricional de alimentación funcional, esto se puede apreciar en la Figura 2.14.

Figura 2.14: Diferentes pantallas de la aplicación móvil [Waltner et al., 2015].

A continuación se especifican más detalles de la Figura 2.14:

(a) En la pantalla de inicio, el usuario elige entre el modo cámara, la base de datos de alimentos y la información de la alimentación funcional.

(b) La base de datos de alimentos con la información de los ingredientes y la información de nutrición.

(c, d) Son las pantallas de información acerca de la alimentación funcional y la pirámide nutricional.

(e, f) El usuario llena un formulario para determinar su perfil nutricional y asignarle el plan que mejor se adapte.

- **A new approach for indoor customer tracking based on a single Wi-Fi connection [Bai et al., 2014]**

  En [Bai et al., 2014] se menciona que debido a la complejidad de la mayoría de los ambientes interiores, la integración de múltiples tecnologías y/o algoritmos de posicionamiento podría conducir a una mejor solución para el seguimiento de posicionamiento en interiores. Por ejemplo, se han demostrado con éxito los sistemas de seguimiento basado en RFID en los últimos años. Los teléfonos inteligentes como lectores de RFID se han aplicado tanto para los teléfonos iPhone y Android.

A pesar de que la prestación de servicios personalizados a los clientes puede mejorar el rendimiento del negocio, sigue siendo un desafío como se menciona en [Bai et al., 2014]. La información que es directamente relevante para los intereses del cliente se basa en un conocimiento fiable de su ubicación y el contexto o entorno de esa ubicación. Este nivel de información no es accesible actualmente de una manera robusta y fiable, debido principalmente a las dificultades asociadas con la determinación del posicionamiento en interiores donde el GPS no funciona. El uso generalizado de los teléfonos inteligentes y los potenciales beneficios comerciales y sociales alcanzables a partir de la entrega de una capacidad robusta de posicionamiento en interiores ha impulsado la tendencia en la última década hacia el desarrollo de una amplia gama de tecnologías y técnicas de posicionamiento en interiores. En combinación con la disponibilidad de tecnologías no tradicionales y señales que pueden ser utilizados para el posicionamiento, así como reducciones dramáticas en el costo y el tamaño de los sensores non-GNSS, basados en aumento las soluciones disponibles para el posicionamiento robusto en entornos GNSS difíciles están evolucionando rápidamente. El trabajo presentado en [Bai et al., 2014] se centra en el posicionamiento basado en Wi-Fi, ya que no requiere infraestructura adicional o especializada para el posicionamiento.

Según lo descripto en [Bai et al., 2014], hay cuatro tipos de modelos de medición comúnmente utilizados basados en Wi-Fi para posicionamiento en interiores: modelos basados en el tiempo de llegada (ToAs), en diferencia de tiempo de llegada (TDOA), en el ángulo de llegada (AOA) y en la intensidad de señal recibida (RSS). El modelo basado en RSS es la más popular. Los tres métodos de estimación de posiciones más comúnmente utilizados para el posicionamiento en interiores y seguimiento de la ubicación son: la celda de origen (COO), triangulación y toma de huellas dactilares. CoO es menos popular que triangulación y toma de huellas dactilares, y su exactitud depende de la forma de aplicación, donde diferentes métodos de aplicación pueden dar lugar a precisiones de ubicación significativamente diferentes. La triangulación por lo general requiere al menos tres conexiones de puntos de acceso (AP). La toma de huellas dactilares se compone de dos fases: una fase de entrenamiento fuera de línea y una fase de posicionamiento. A menudo se utilizan técnicas deterministas y probabilistas por su estimación de las posiciones. Durante la fase de entrenamiento fuera de línea, se seleccionan una serie de puntos conocidos, llamados puntos de referencia (RP), que cubre el área de interés y se recolectan los valores recibidos a través del indicador de intensidad de señal (RSSI) para cada RP. Estos conjuntos de datos de observación se almacenan en una base de datos de huellas dactilares (DB). En la fase de posicionamiento, se recogen un conjunto de observaciones de RSSI de los puntos de acceso de los alrededores y las observaciones se comparan con los almacenados en la base de datos de huellas digitales para la identificación del mejor conjunto de datos coincidente en la base de datos. La ubicación del RP asociado con el mejor conjunto de datos coincidente se toma como estimación de la posición del usuario.

Al igual que la mayoría de los centros comerciales o despliegues de infraestructura, el sistema Wi-Fi utilizado para el proyecto presentado en [Bai et al., 2014] fue diseñado principalmente para comunicaciones inalámbricas y de entretenimiento en lugar de para la determinación de seguimiento y posicionamiento. Por ejemplo, el sistema Wi-Fi usado registró sólo una conexión a un punto activo (hotspot) Wi-Fi (es decir, un punto de acceso) de un cliente a la vez en un archivo de registro. El interés principal es el seguimiento del comportamiento de compra de los clientes sobre la base de los datos de registro. Sin embargo, la mayoría de los modelos de posicionamiento convencionales o métodos no son aplicables al escenario de conexión individual, por lo tanto se debe

desarrollar un nuevo enfoque, como se presenta en [Bai et al., 2014]. Los autores describen lo siguiente. Debido a la conexión Wi-Fi única registrada, el único método de estimación que puede ser utilizado para este seguimiento es CoO. Una celda de AP (Acces Point) cubierta, usualmente un polígono, está formada usando una regla de que la fuerza de las señales desde ese AP es más fuerte que la señal de otros puntos de acceso de alrededor. La Figura 2.15a muestra cómo se forman los puntos de acceso desde AP5 y AP6 en un espacio libre. De acuerdo al diseño requerido en los sistemas Wi-Fi de Cisco, hay entre un 10 o 20% de superposición entre las áreas de 2 AP adyacentes como muestra la figura 2.15b.

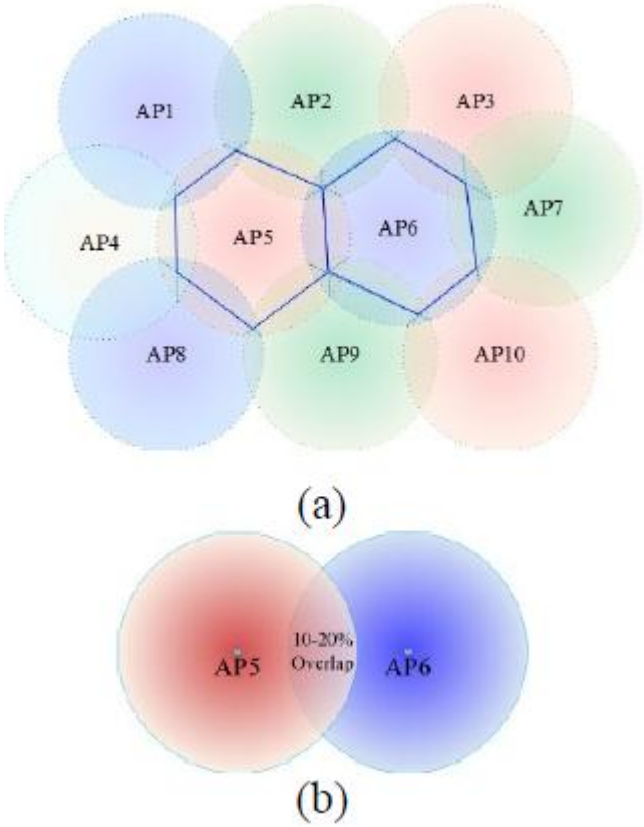


Figura 2.15: Como se forman los puntos de acceso [Bai et al., 2014].

Cabe mencionar que del lado del servidor, el sistema rastrea un dispositivo móvil conectado a un AP y se puede considerar la posición del dispositivo dentro de la celda asociada a ese AP. Casi todos los sistemas basados en celdas WLAN y basados en células RF pueden ser adaptadas fácilmente y con bajo costo para proveer capacidad de estimación de ubicación CoO. Sin embargo, CoO tiene el inconveniente de granularidad gruesa. Por varias razones, los dispositivos móviles pueden estar asociados a celdas que no están próximos físicamente. Esta granularidad gruesa puede ser un desafío cuando se intenta resolver la posición real de un dispositivo móvil en una estructura de varios pisos donde la superposición de celdas es considerable. Para mejorar la granularidad en los métodos de CoO, se puede aplicar una técnica de mayor potencia de señal.

El núcleo del método CoO es la determinación de las celdas, por ejemplo, dividiendo el área de interés en varios polígonos/celdas pequeñas y optimizando las celdas. Se puede utilizar un diagrama Voronoi para dividir un área en un número de regiones que sea óptimo. En la investigación presentada en [Bai et al., 2014], se usó un paquete de software, llamado ArcMap, para generar automáticamente un diagrama Voronoi basado

en la distribución de los AP para cada piso de un centro comercial. Cada celda (o polígono) del diagrama contiene un único AP. Estas celdas pueden llamarse "*celdas iniciales*". En la Figura 2.16 se muestra el diagrama Voronoi para un piso del centro comercial.

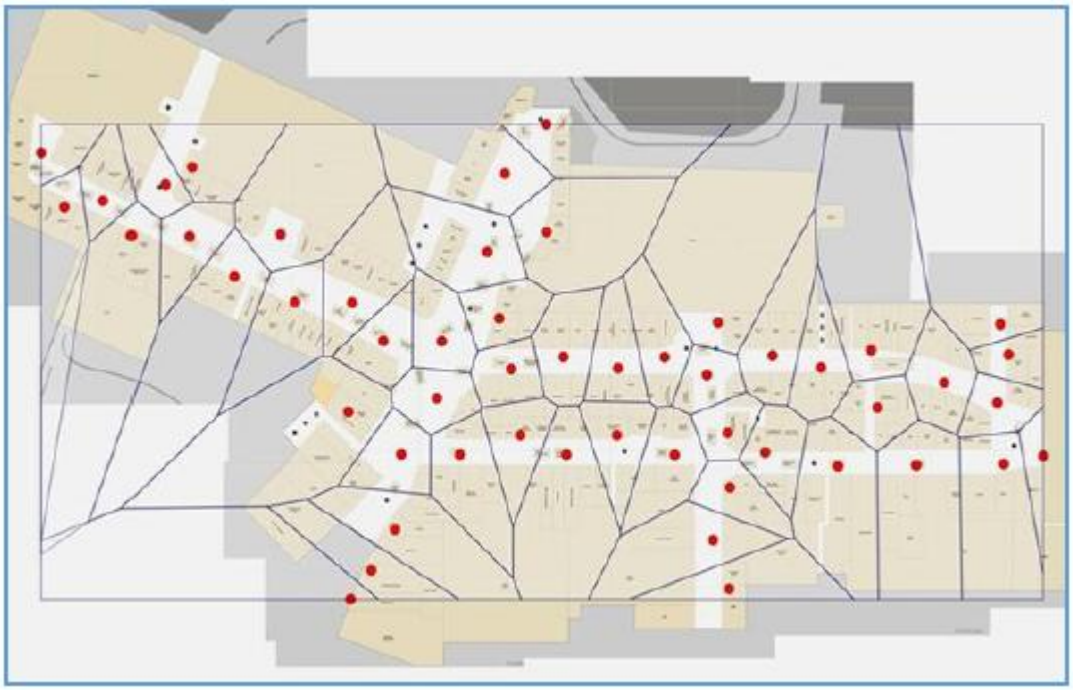


Figura 2.16: Diagrama Voronoi para un piso del centro comercial [Bai et al., 2014]

Las celdas iniciales pueden ser usadas para ambientes de espacios libres pero raramente se pueden usar en un ambiente interior complejo. Por esta razón, se debe llevar a cabo un ajuste manual y un proceso de calibración en el lugar para determinar la proximidad de los límites de las celdas. Luego del ajuste manual basado en el entorno físico, se obtiene un nuevo grupo de celdas, como se muestra en el diagrama de Voronoi de la Figura 2.17.

Figura 2.17: Ajuste manual del Diagrama Voronoi [Bai et al., 2014].

Sin embargo, se sigue necesitando un proceso de calibración en el lugar para lograr que las celdas coincidan perfectamente con el requerimiento de seguimiento. Este proceso es crítico y requiere mucho trabajo. En la Figura 2.18 se puede ver un ejemplo donde se ajusta el límite del área en (a) para que quede como en (b). Una vez que el par AP-celda está confirmado, se puede determinar fácilmente la relación entre el AP y la tienda específica. Se genera una tabla con la información y se almacena en una base de datos para usar en el seguimiento en tiempo real.

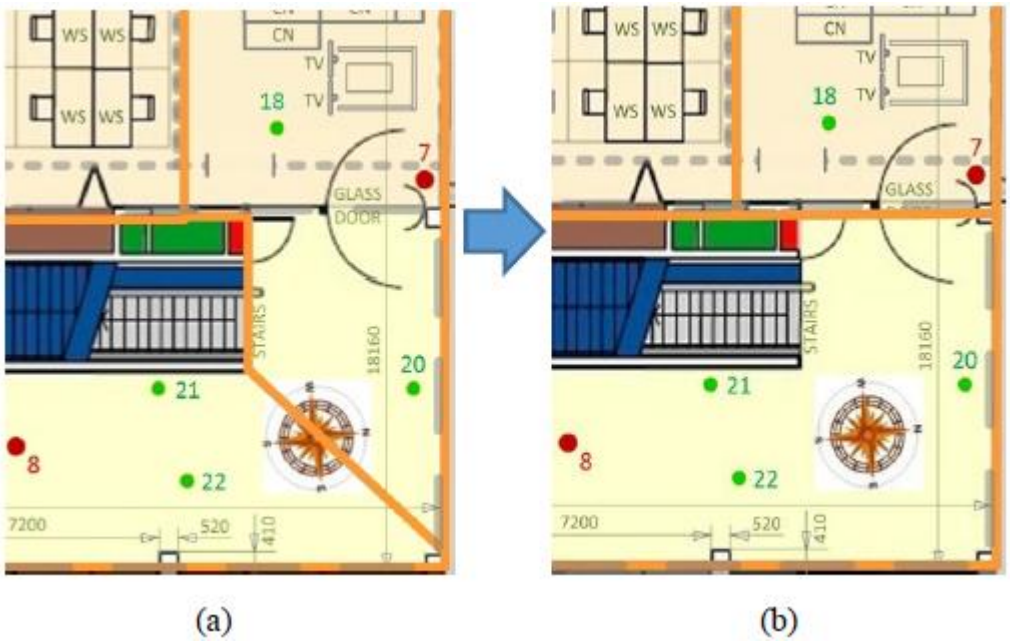


Figura 2.18: Ajuste del límite de un área [Bai et al., 2014].

Para realizar el seguimiento en tiempo real, se toma el AP con el valor RSSI más fuerte recibido por el smartphone. Se considera que la posición del usuario será dentro de la celda asociada a la tienda con dicho AP.

- **An Indoor Wireless System for Personalized Shopping Assistance [Asthana et al., 1994].**

  En [Asthana et al., 1994] se presenta *Personalized Shopping Assistance* (PSA), un asistente de compras que brinda un servicio único que personaliza la atención provista a los consumidores basadas en necesidades individuales y en un perfil de compra acumulativo a través del tiempo. Este servicio se puede ver gráficamente en la Figura 2.19, se basa en dos productos: el dispositivo PSA, que es una unidad de comunicación inalámbrica y el servidor PSA centralizado ubicado en el centro comercial con el cual los consumidores se comunicarán utilizando el dispositivo PSA, este servidor es una computadora responsable de controlar todos los dispositivos PSA en su dominio.

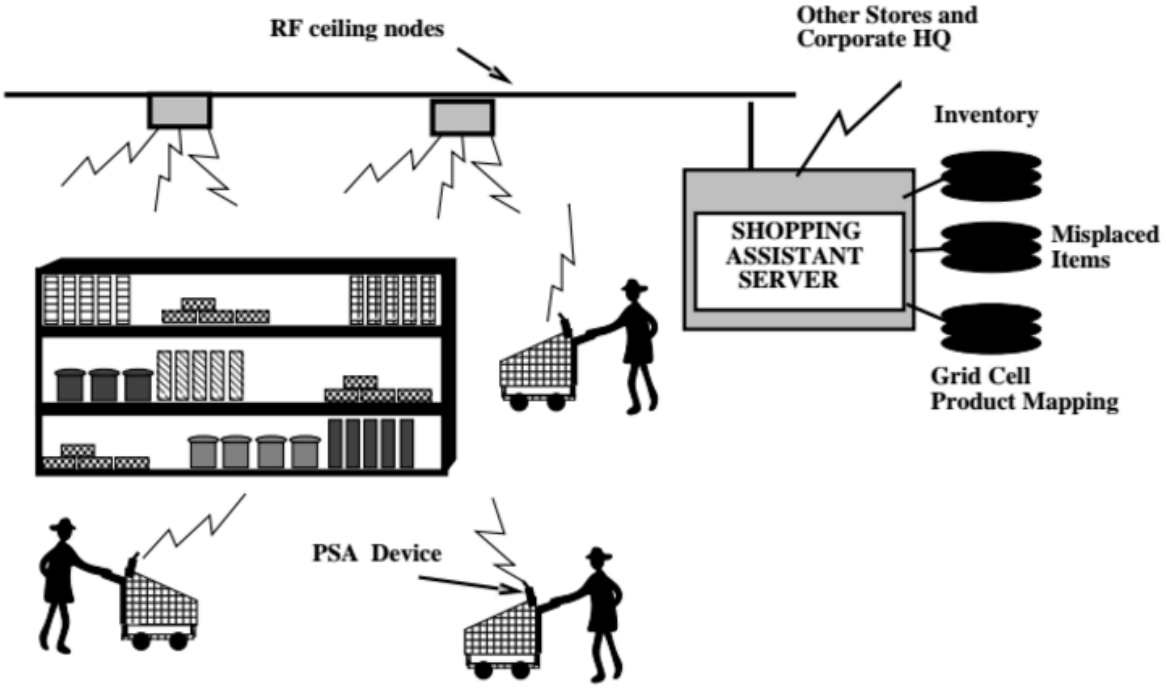


Figura 2.19: Servicio PSA (Personal Shopping Assistant) [Asthana et al., 1994].

El dispositivo PSA es una unidad pequeña de bajo costo que contiene auriculares y un micrófono (como se puede apreciar en la Figura 2.20), que se puede integrar en un carrito de supermercado. Además cuenta con un láser led para escanear códigos UPC. La única salida de este dispositivo es una pantalla LCD para transmitir imágenes y los auriculares para transmitir audio. El PSA recibe señales de otros dispositivos llamados "*beacons*" y triangula estas señales para obtener una medida aproximada de su ubicación geográfica.

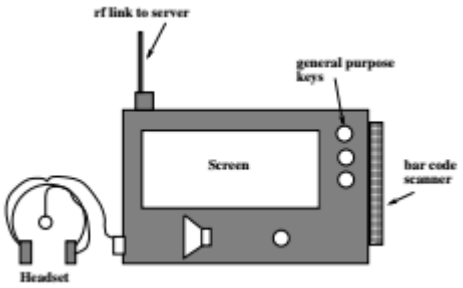


Figura 2.20: Dispositivo PSA

Para mantener el bajo costo y una funcionalidad flexible el dispositivo PSA por sí solo no hace nada, exceptuando enviar y recibir información del servidor. Cuenta con un microprocesador cuya tarea es procesar los protocolos adecuados para la comunicación con el servidor, comprensión y descomprensión de audio e imagen y etiquetar los mensajes desde los dispositivos de entrada correctamente antes de transmitirlo a la interfaz inalámbrica.

Para identificar un usuario existen varias alternativas, según mencionan los autores en [Asthana et al., 1994]: basarse en el identificador de cada dispositivo, se puede escanear una identificación de usuario, se puede tener un sistema de identificación por voz o una autenticación de una tarjeta inteligente. En la Figura 2.21 se pueden apreciar diferentes componentes del servidor PSA.

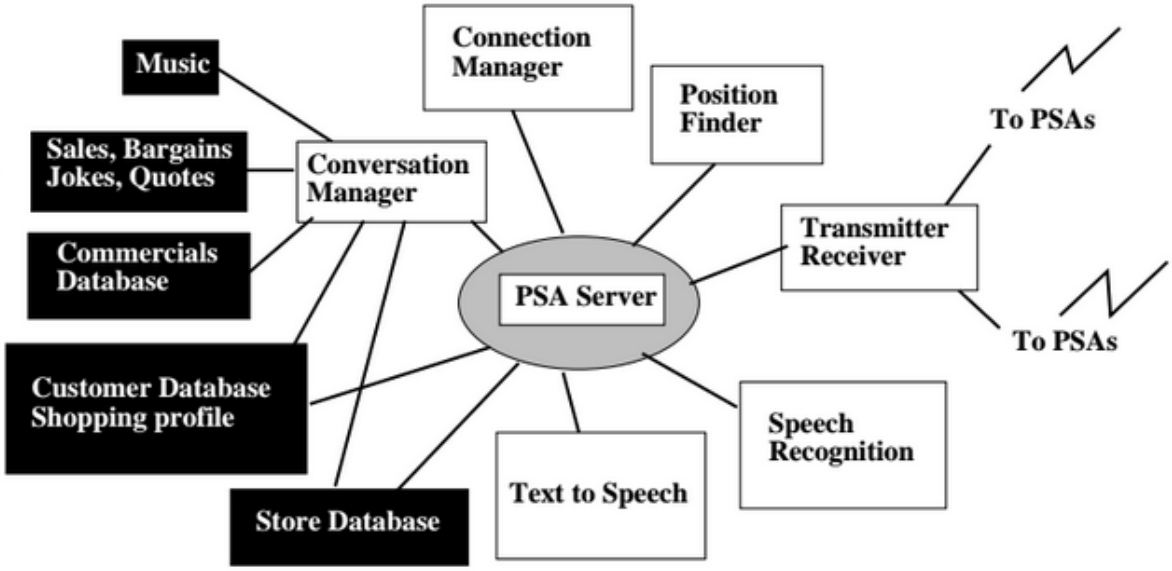


Figura 2.21: Componentes Servidor PSA [Asthana et al., 1994]

3. MODELO PROPUESTO

En este capítulo se describe la problemática que se quiere resolver, para luego presentar el modelo propuesto para dar solución a dicha problemática.

3.1 Problemática a resolver

Se desea contar con una solución de modelado para un sistema que brinde asistencia en la movilidad en un supermercado o cadena de supermercados. Este sistema brindará soporte a los clientes a la hora de realizar su compra, con lo cual deberá contar con información de todos los elementos que forman parte del supermercado. Para una mejor comprensión del problema a resolver, se usarán pantallas esquemáticas para mostrar la funcionalidad a considerar.

La pantalla inicial del sistema se puede apreciar en la Figura 3.1, cuenta con un logo/nombre de la aplicación y un formulario a través del cual un cliente puede iniciar sesión ingresando usuario y contraseña. En caso de no contar con estos datos tiene la opción de *Registrarse*.

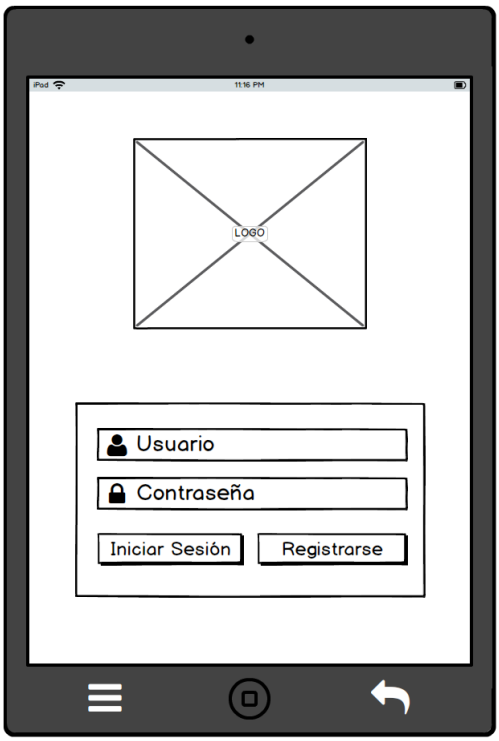


Figura 3.1: Pantalla de Inicio de Sesión

Para registrar un nuevo usuario se accede a otra pantalla, como se ve en la Figura 3.2, donde se visualiza un formulario con los siguientes datos a completar: nombre de usuario, email, contraseña, repetir contraseña. El email sirve para luego realizar una confirmación de usuario.

Figura 3.2: Pantalla para Registrar un Usuario

El manejo de usuarios dentro del sistema permite que se conecten desde distintos dispositivos manteniendo su información y datos ya configurados. Debido a lo dicho anteriormente, podemos identificar a los usuarios como un concepto importante a modelar, ya que tendrán asociada información relevante para la aplicación. El supermercado deberá tener identificados a sus clientes para poder brindarle diferentes servicios.

Una vez que el usuario ingresó al sistema, se visualizará la pantalla de inicio, como se muestra en la Figura 3.3.

Figura 3.3: Pantalla de Inicio

Veamos más detalles del menú de funcionalidades presentado en la Figura 3.3:

- *Listado*: se visualizarán las listas de compras creadas por el usuario con la opción de crear nuevas listas.
- *Buscar*: permite realizar la búsqueda de un producto, góndola o sector.
- *Escanear*: el sistema permite escanear un código QR para ver información de los elementos del supermercado y hacer seguimiento de la posición del usuario dentro del lugar.
- *Ver Mapa*: se muestra el mapa del supermercado a nivel sector, góndola o producto según corresponda.
- *Preferencias*: se muestra la preferencia de compra del usuario y la opción de modificarla.
- *Configuración*: se visualizan los datos referidos al usuario y la opción de modificarlos.

Dicho menú es accesible desde cualquier pantalla desplegándose a través del botón ☰ como se muestra en la Figura 3.4, y cuenta con la funcionalidad *Cerrar sesión* (previa confirmación).

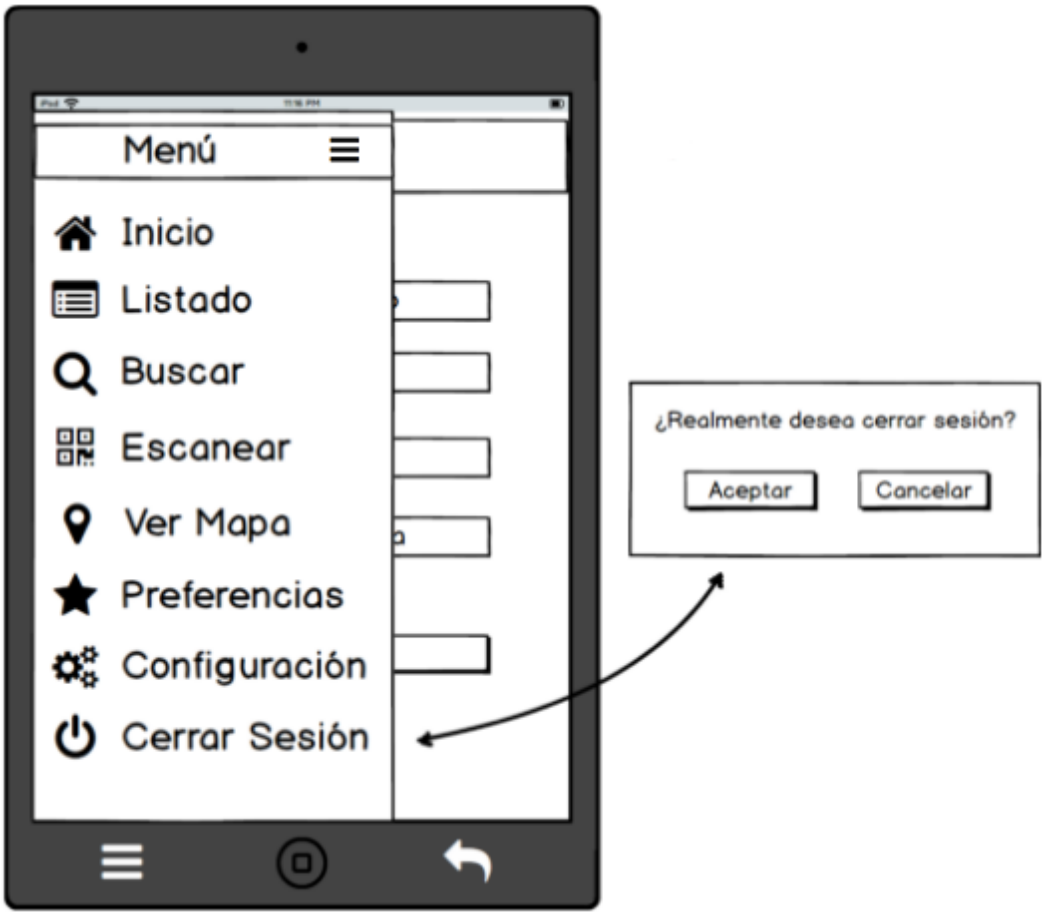


Figura 3.4: Menú desplegable

Veamos a continuación con más nivel de detalle cada una de las opciones previamente mencionadas, analizando las pantallas involucradas dentro de cada una:

- Listado

Al seleccionar la opción *Listado* del menú, se muestran las listas de compras creadas y un botón para *Crear lista* como se puede apreciar en la Figura 3.5.

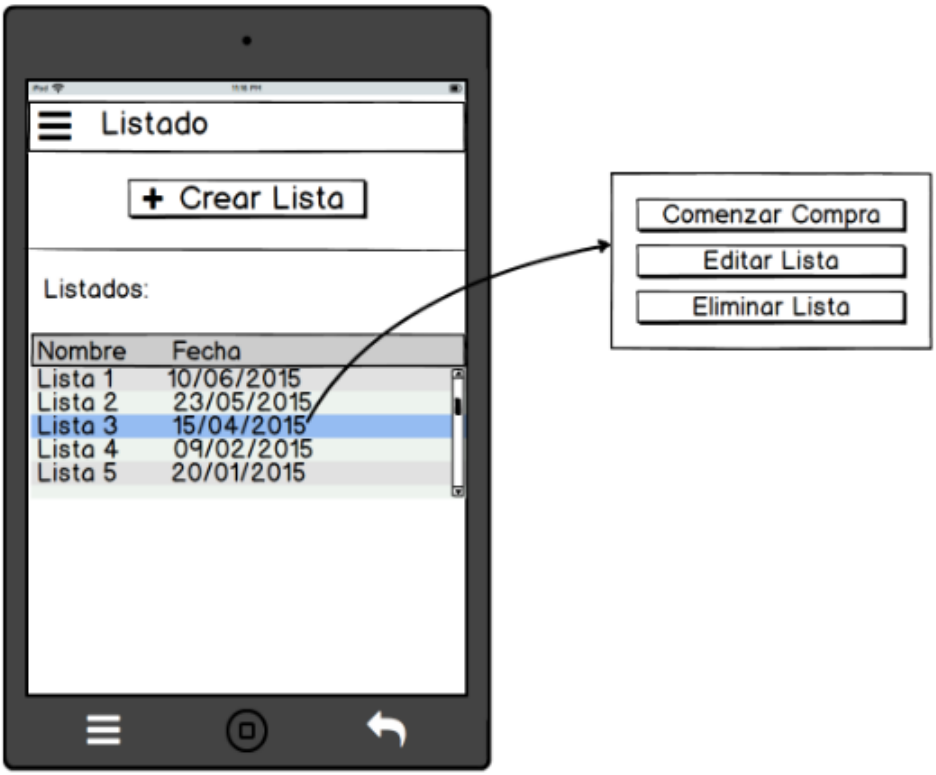


Figura 3.5: Listas de Compras

Se puede apreciar en la Figura 3.5 que por cada lista de compra se tienen las opciones: *Eliminar lista, Editar lista* y *Comenzar compra*. Si se selecciona la opción *Eliminar lista* (de la Figura 3.5), se borra del listado junto con todos sus elementos asociados (previa confirmación).

Si se selecciona la opción *Editar lista* (de la Figura 3.5), se accede a la pantalla de la Figura 3.6, donde se visualiza la información con respecto a la lista seleccionada: nombre y elementos que contiene. De cada elemento se muestra el nombre, precio y cantidad. Al seleccionar alguno se abre una ventana tipo pop up con las opciones para *Editar elemento* o *Eliminar elemento*. Esta pantalla también cuenta con la posibilidad de *Buscar elemento* dentro de la lista actual. Además se pueden agregar elementos nuevos a través del botón *Agregar* que redirecciona a la pantalla de la Figura 3.7 para realizar una búsqueda de todos los elementos y seleccionar el que se desee agregar. Para confirmar los cambios realizados se debe accionar el botón *Guardar*, caso contrario las modificaciones no se verán reflejadas en el sistema.

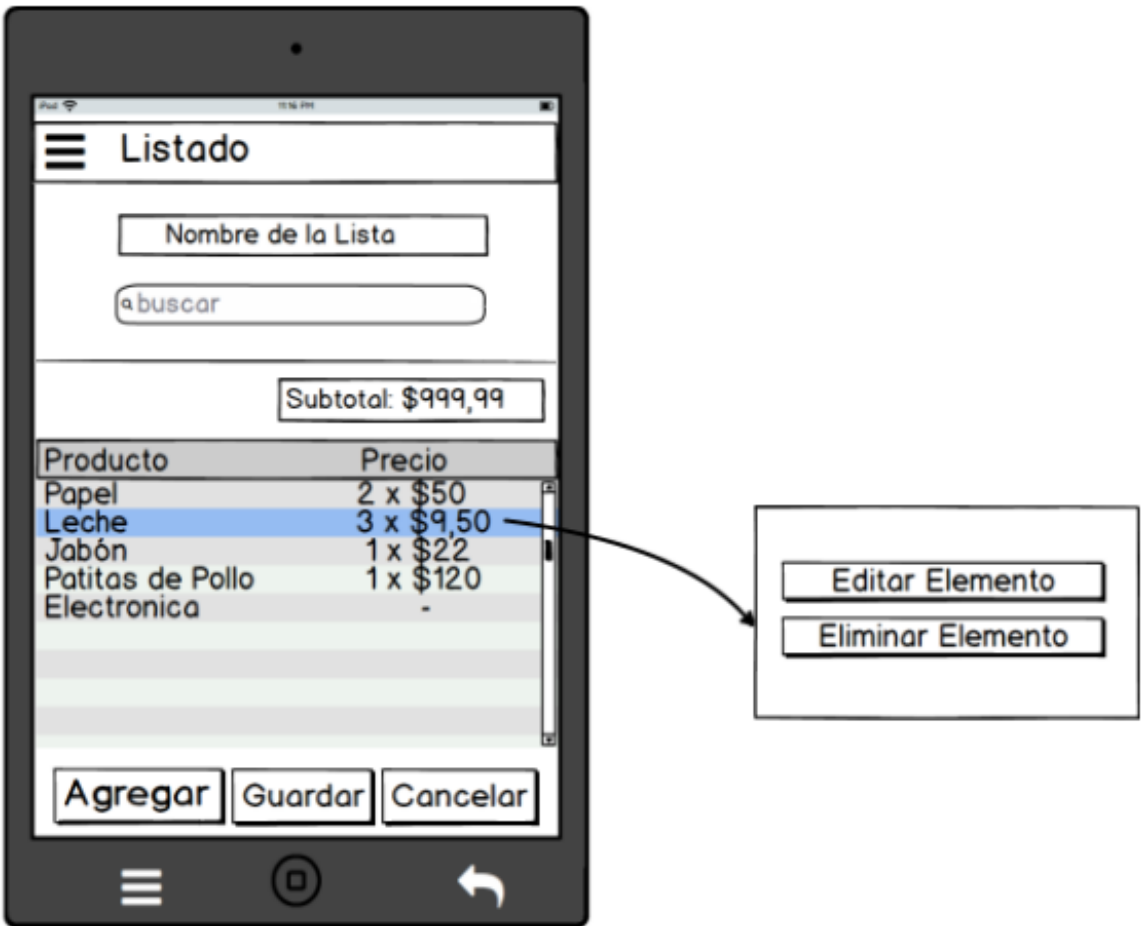


Figura 3.6: Pantalla de Edición de una Lista de Compra

A partir de esta funcionalidad descrita, podemos identificar que una lista de compra es un concepto importante en el sistema que deberá ser modelada. Además de tener nombre y fecha para identificarla, se le asociará un conjunto de elementos. Estos elementos también son parte importante de la problemática a resolver, por lo tanto se modelarán teniendo en cuenta su clasificación: por sector, producto y góndola.

- Buscar

En la pantalla de la Figura 3.7 se puede apreciar cómo se realizan las búsquedas de elementos, este ejemplo, se especificó que empiecen con "*Le*". Existen cuatro opciones de búsqueda que corresponden a los filtros aplicables: *Todo*, *Producto*, *Góndola*, *Sector*. Por defecto, la opción seleccionada es *Todo*. Debajo se muestra un listado con los resultados, detallando nombre y precio (en caso de ser un producto). Al seleccionar alguno se abre una ventana tipo pop up específico para cada tipo de elemento.

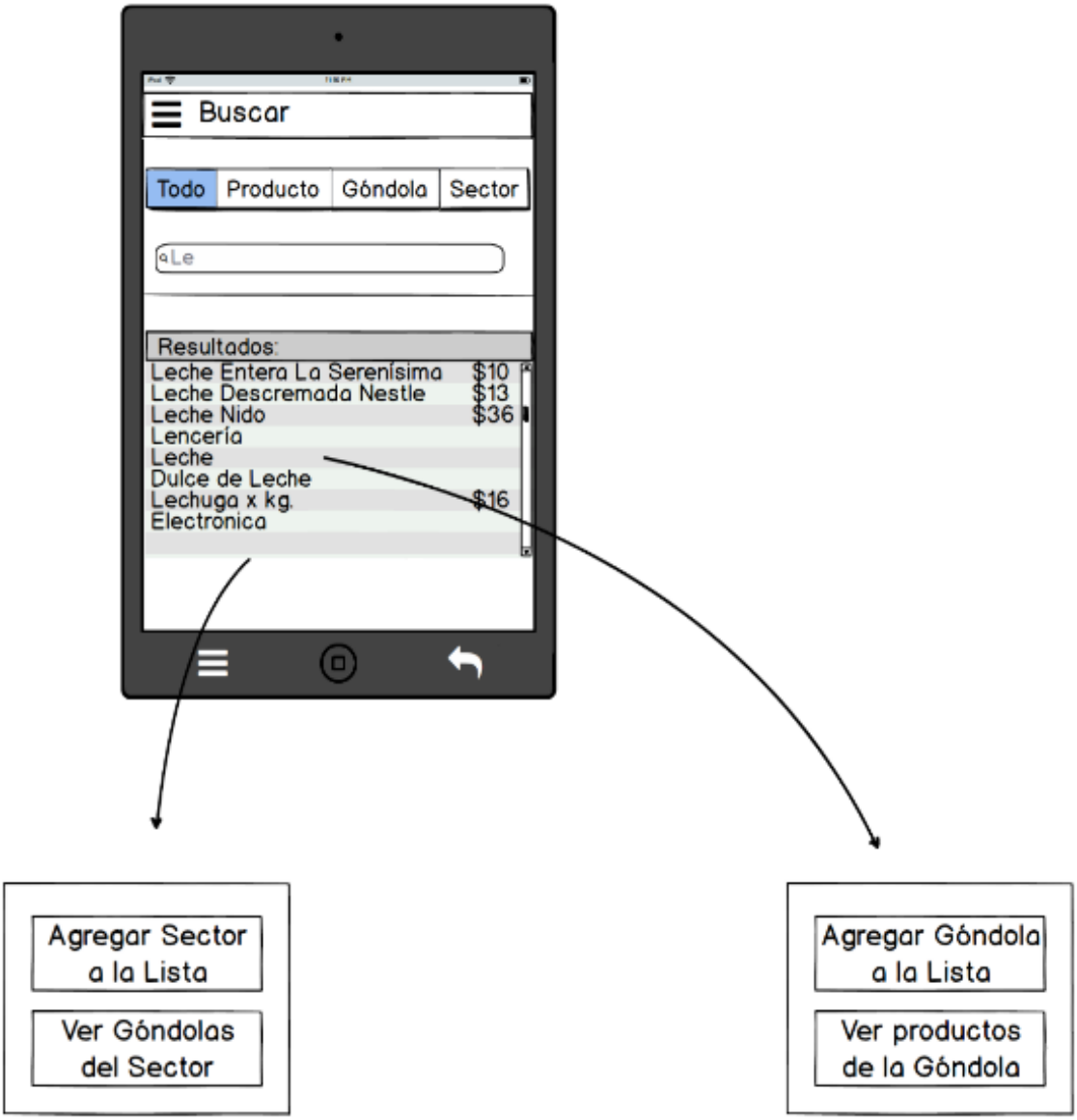


Figura 3.7: Búsqueda de Elementos

Se puede apreciar en la Figura 3.7 que, si es un sector, el usuario podrá *Agregar sector a la Lista* o *Ver góndolas del Sector*, al elegir esta última opción se visualiza una pantalla como la de la Figura 3.8.1.

En el caso de seleccionar una góndola, el usuario podrá *Agregar góndola a la Lista* o *Ver productos de la Góndola*, al elegir esta última opción se visualiza una pantalla como la de la Figura 3.8.2.

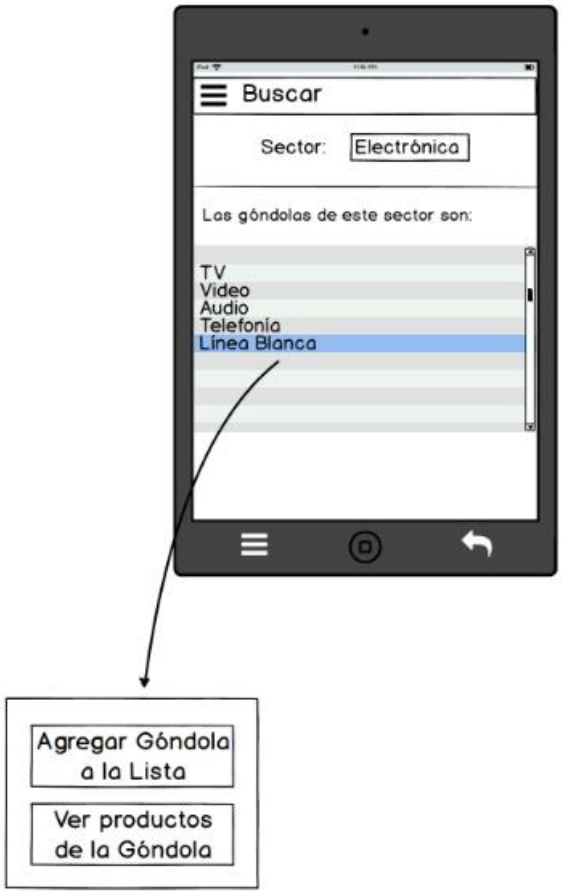


Figura 3.8.1: Resultado seleccionado de tipo Sector

Si se selecciona una góndola desde la Figura 3.8.1, el usuario podrá *Agregar góndola a la lista* o *Ver productos de la góndola*, al elegir esta última opción se visualiza la pantalla de la Figura 3.8.2.

Figura 3.8.2: Resultado seleccionado de tipo Góndola

Si se selecciona un producto, se visualizará la información asociada al mismo y, si el usuario lo desea, agregarlo a la lista de compra detallando la cantidad, como se muestra en la Figura 3.8.3.

Figura 3.8.3: Resultado seleccionado de tipo Producto

Volviendo a las opciones presentadas en la Figura 3.5, seleccionando algún listado se puede acceder a la funcionalidad *Comenzar compra* donde el usuario deberá escanear el código del elemento más cercano para que el sistema pueda determinar su posición actual, tal como se muestra en la Figura 3.9.

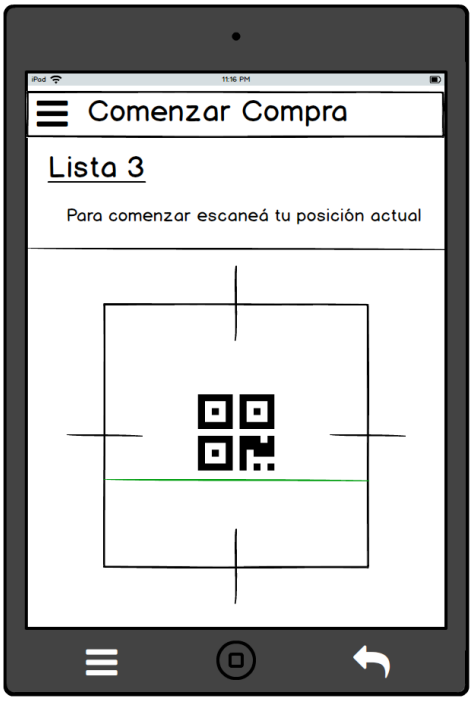


Figura 3.9: Comenzar una Compra

Una vez que escanea su posición actual, el sistema calcula el recorrido óptimo a realizar, teniendo en cuenta la preferencia de compra. En el caso de que no haya una preferencia cargada, se le avisará que debe ingresarla (como se mostrará más adelante). En este caso, identificamos la preferencia como un concepto a modelar, ya que de esta dependerá la forma en que se arma cada recorrido.

Cabe aclarar que el usuario se va a mover físicamente de un lugar a otro del supermercado para realizar el recorrido, y mediante la lectura de códigos QR, se irá modificando la posición del mismo, así el sistema le proveerá diferentes opciones que podrá realizar. Para la ejemplificación de la problemática se optó por posicionar al usuario a través de códigos QR.

Una vez iniciada la compra se visualiza la pantalla de la Figura 3.10.1 que contiene dos pestañas: *Recorrido* y *Lista*.

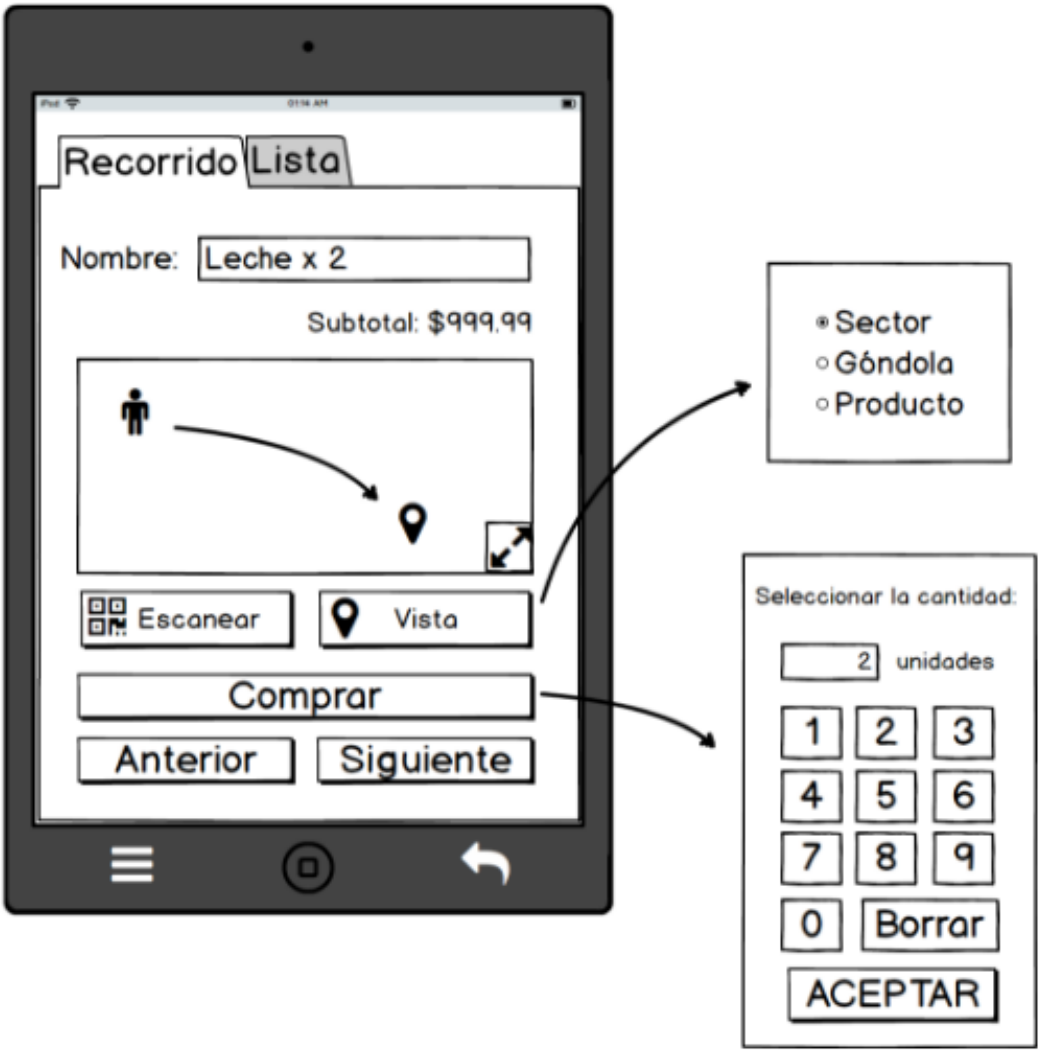


Figura 3.10.1: Realizando una compra, solapa Recorrido activa

En la pestaña *Recorrido* (Figura 3.10.1) se visualiza el nombre del elemento actual a comprar (que será calculado automáticamente a partir del recorrido armado, así como también los próximos a ser comprados), el mapa con el recorrido para que el usuario

pueda llegar a dicho elemento (con la posibilidad de expandir la imagen a pantalla completa), el subtotal de lo comprado hasta el momento, y además los siguientes botones:

- *Escanear*: escanear el código QR de un elemento, ya sea para ver su información como para agregarlo a la lista de compra (como se mostrará más adelante). Si el elemento escaneado coincide con el elemento actual a comprar, la acción de *Escanear* es equivalente a la del botón *Comprar*. El sistema a su vez actualizará la posición del usuario, ya que se asume que el elemento que se escanea se encuentra en su lugar correcto.
- *Vista:* modificar la vista del mapa. Se cuenta con tres opciones para ayudar al usuario a que se ubique mejor en el supermercado dependiendo de su necesidad en cada momento, si requiere una visión general del lugar o más detallada. Las opciones existentes son vista del *Sector*, *Góndola* y *Producto* (las cuales se explicarán más adelante). De esta manera, se lo va a ir asistiendo en la movilidad al usuario.
- *Comprar*: comprar el elemento actual, para lo cual primero abre el escáner para identificar el código del elemento, y luego muestra una ventana tipo pop up para que el usuario seleccione la cantidad que desea comprar, por defecto se muestra la cantidad especificada al momento en que se agregó dicho producto al listado.
- *Anterior*: muestra el elemento anterior al actual, teniendo en cuenta el orden del recorrido armado. Esta opción está disponible excepto cuando el elemento actual sea el primero del recorrido.
- *Siguiente*: muestra el próximo elemento a comprar. Esta opción no estará disponible cuando el elemento actual sea el último del recorrido.

En la solapa denominada *Lista* (la cual se puede apreciar en Figura 3.10.2) permite ver el listado completo de los elementos que conforman la compra, junto con el subtotal de lo comprado hasta el momento. De cada elemento se detalla el nombre, precio, cantidad y si ya fue comprado o no. Además, al seleccionar alguno se abre una ventana tipo pop up con las opciones para *Modificar* el elemento o *Eliminar* dicho elemento de la compra. Por último, se cuenta con un botón para *Agregar un elemento nuevo*, ya sea escaneando su código (como se mostrará en la Figura 3.11) o realizando una búsqueda por su nombre (como se mostró en la Figura 3.7).

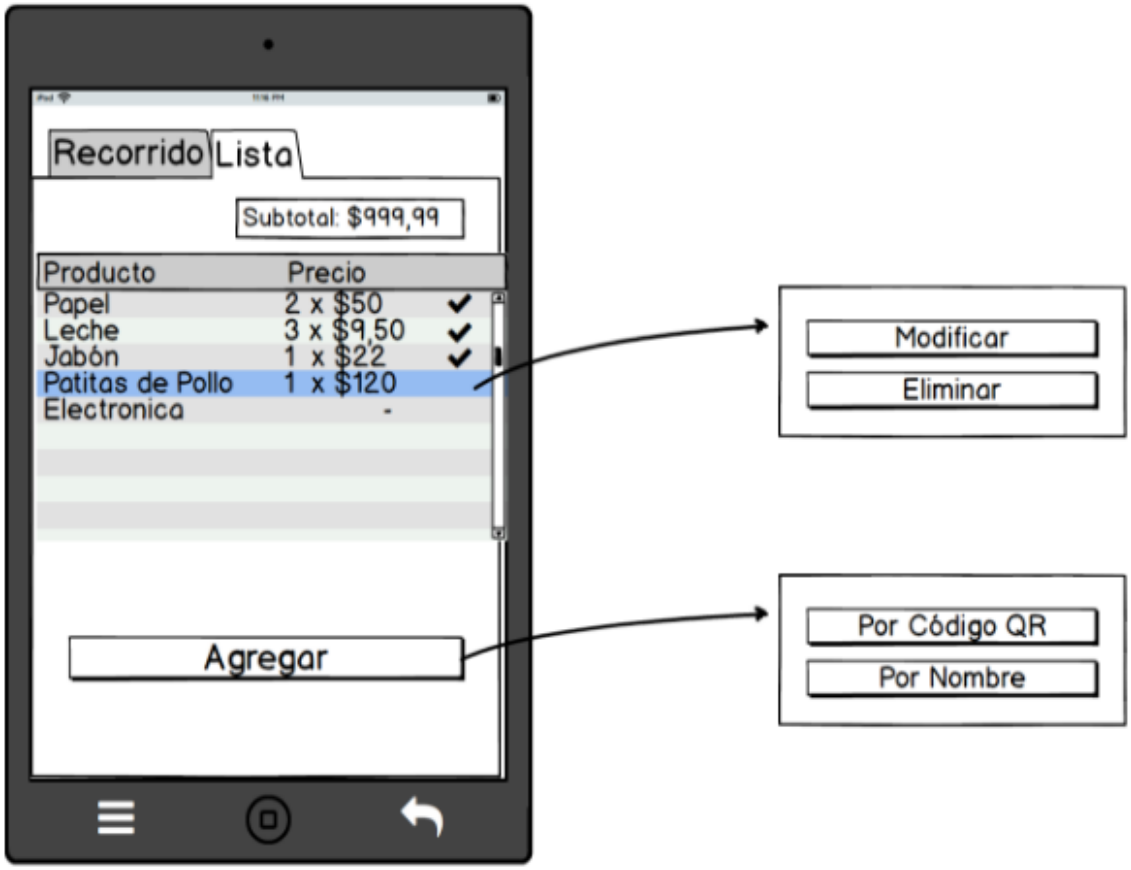


Figura 3.10.2: Realizando una compra, solapa Lista activa

A partir de esto identificamos el recorrido como un concepto a modelar, junto con las posiciones que determinarán la ubicación de cada elemento (*Sector*, *Góndola* y *Productos*) así como también la del usuario. Contar con una forma de representar posiciones en el sistema permitirá hacer un seguimiento de cómo el usuario se mueve en el supermercado.
Como dijimos en párrafos anteriores, los recorridos se formarán teniendo en cuenta una preferencia, una de estas opciones podría ser, por ejemplo, por cercanía de los elementos a comprar. Para poder armar este tipo de recorridos el sistema debe conocer las posiciones de cada uno, con lo cual el posicionamiento es una parte fundamental en nuestro modelo.
Al momento de realizar el modelo se debe tener en cuenta que una posición representa un punto específico, en el caso de la posición del usuario, pero también puede referirse a un área dentro del supermercado, por ejemplo un sector.

- Escanear Código QR

La pantalla de la Figura 3.11 se muestra cómo escanear un código QR. Dicha funcionalidad se puede acceder desde la opción correspondiente en el menú principal (mostrado en la Figura 3.3 y Figura 3.4), donde la información resultante será sólo a modo informativo. Además, se puede acceder desde la pantalla de la Figura 3.10 en cuyo caso el usuario estará realizando una compra y tendrá la posibilidad de agregar el elemento escaneado a la lista actual y actualizar su posición y el mapa del recorrido. Si el

elemento escaneado coincide con el elemento actual a comprar, la acción de Escanear es equivalente a la de Comprar.
Es decir, el escaneo tendrá una semántica distinta si se está realizando una compra o si sólo se quiere obtener información de un elemento. En ambos casos, el escaneo servirá para que el sistema pueda ubicar al usuario en el supermercado.

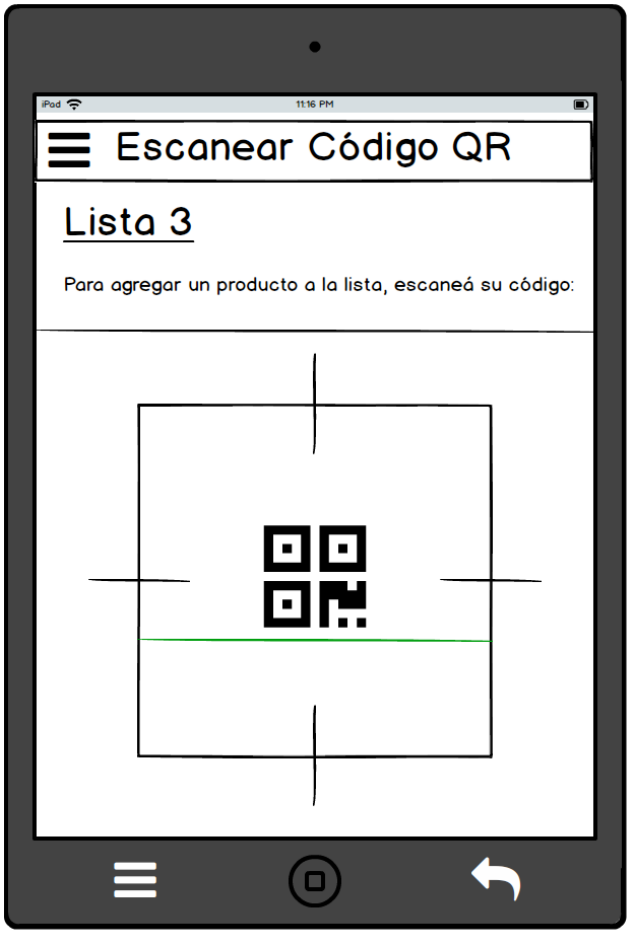


Figura 3.11: Escanear un Código QR

En la siguientes figuras se visualizan los resultados dependiendo del tipo del elemento escaneado (sector, góndola o producto) con la opción de agregarlos al listado, en el caso de que el usuario esté realizando una compra. Si se escaneó un código sólo a modo informativo, la opción para agregar el elemento al listado no se visualizará.

En la Figura 3.12.1 se muestra el resultado de un elemento escaneado de tipo *Sector* (en este ejemplo, "*Electrónica*"), donde se listan las góndolas que contiene dicho sector y el botón que permite *Agregar* el sector a la lista actual.

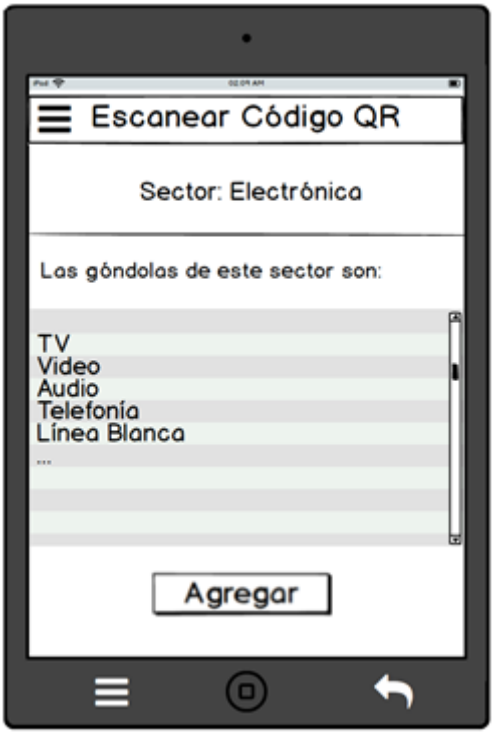


Figura 3.12.1: Elemento escaneado de tipo Sector.

En la Figura 3.12.2 se muestra el resultado de un elemento escaneado de tipo *Góndola* (en este ejemplo, "*Audio*"), donde se listan los productos que contiene dicha góndola y la opción de *Agregar* la góndola al listado actual.

Figura 3.12.2: Elemento escaneado de tipo Góndola.

En la Figura 3.12.3 se muestra el resultado de un elemento escaneado de tipo *Producto* (en este ejemplo, "*Leche Entera la Serenísima*"). Se visualiza la información del producto y si el usuario está realizando una compra, se muestra la opción de agregarlo a la lista de compra detallando la cantidad.

Figura 3.12.2: Elemento escaneado de tipo Producto.

- Ver Mapa

En la pantalla de la Figura 3.13 se puede visualizar el mapa del supermercado. Se accede desde la opción correspondiente en el menú principal (mostrado en la Figura 3.3 y Figura 3.4). Para poder mostrarle al usuario dónde se encuentra, el sistema se basará en la posición del último elemento escaneado. Se puede apreciar en la Figura 3.13 que el nivel de detalle del mapa es a nivel *Sector*.

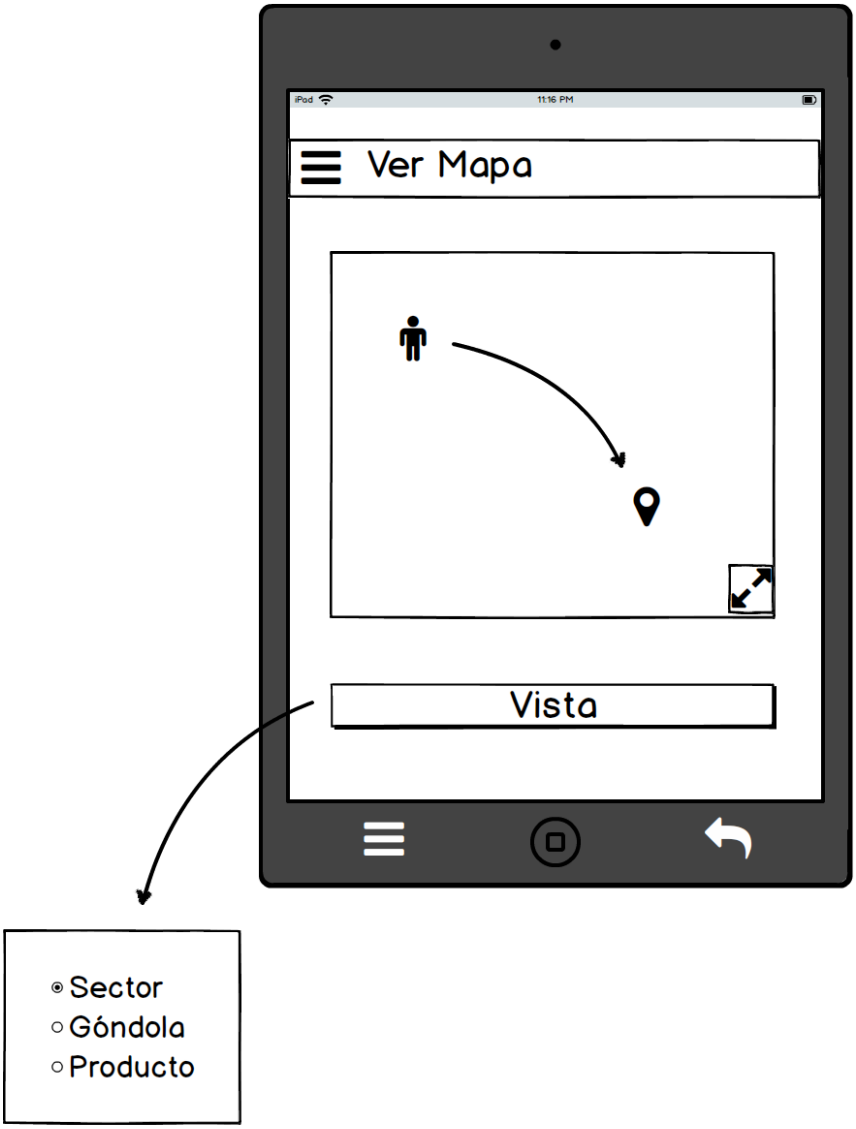


Figura 3.13: Visualizar el Mapa del Supermercado

Si no se ha escaneado un código QR anteriormente, se solicitará al usuario que actualice su posición escaneando el código del elemento más cercano para que el sistema pueda determinar su posición actual (el usuario puede leer cualquier código del supermercado que esté más cerca). Además, se cuenta con un ícono para *expandir el mapa* (como se visualiza en la Figura 3.13), el cual muestra la imagen en pantalla completa, y un botón *Vistas* para seleccionar una de las opciones (*Sector*, *Góndola* o *Producto*) como se muestran en las figuras Figura 3.14.1, Figura 3.14.2 y Figura 3.14.3 para ver a nivel de detalle *Sector*, *Góndola* o *Producto* respectivamente.

La Figura 3.14.1 correspondiente a la vista del *Sector* permite tener una visión amplia del establecimiento. Es útil para el usuario este tipo de vista cuando se encuentra lejos del destino o cuando el destino es un sector particular. En este esquemático no se dan niveles de detalles del espacio físico, en el sistema se mostrará más detalles del supermercado.

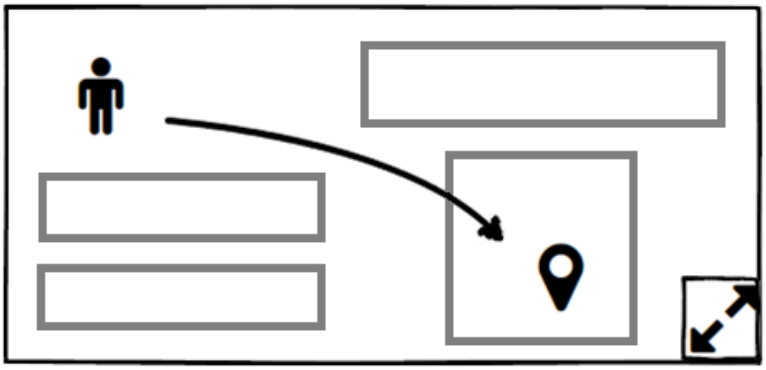


Figura 3.14.1: Opción de vista: Sector.

La Figura 3.14.2 correspondiente a la vista de la *Góndola* permite tener una visión más concreta de dónde se encuentra ubicada con respecto al sector. Es útil al momento de escanear un elemento cuando el usuario está comprando y el destino donde tenía que dirigirse es diferente a su posición. Se puede apreciar en la figura que la vista del mapa es desde arriba.

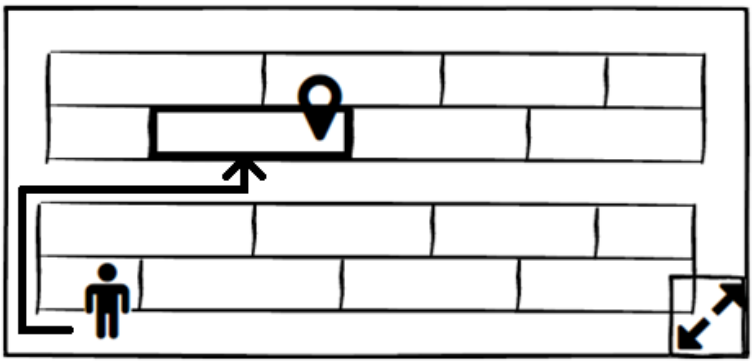


Figura 3.14.2: Opción de vista: Góndola.

La Figura 3.14.3 correspondiente a la vista de un *Producto* permite tener una visión más concreta de dónde se encuentra ubicado con respecto a la góndola. Es útil al momento de escanear un elemento cuando el usuario está comprando y el destino donde tenía que dirigirse es diferente a su posición y dicha góndola lo contiene. Podríamos pensar a esta vista como la que tiene más zoom, ya que hay nivel de precisión en dónde está el producto. Esta vista muestra la góndola de frente, y remarca dónde está ubicado el producto.

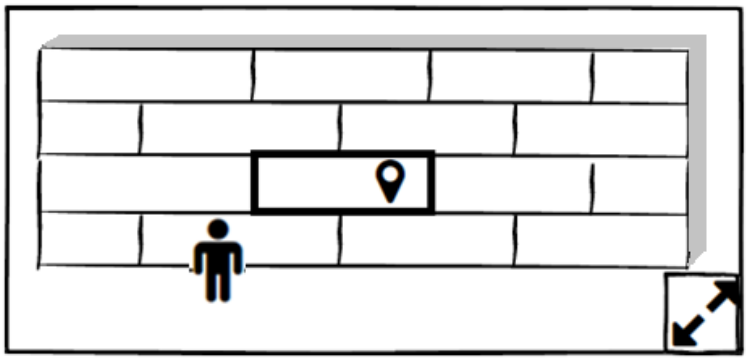


Figura 3.14.3: Opción de vista: Producto.

- Preferencias

En la Figura 3.15 el usuario puede seleccionar su preferencia respecto al armado de los recorridos de compra (por ejemplo, por orden de la lista, cercanía del producto, orden alfabético) y modificarla. El sistema se basará en esta elección al momento de armar cada recorrido de compra. Esta opción (*Preferencias*) será accesible desde el menú principal (mostrado en la Figura 3.3 y la Figura 3.4).

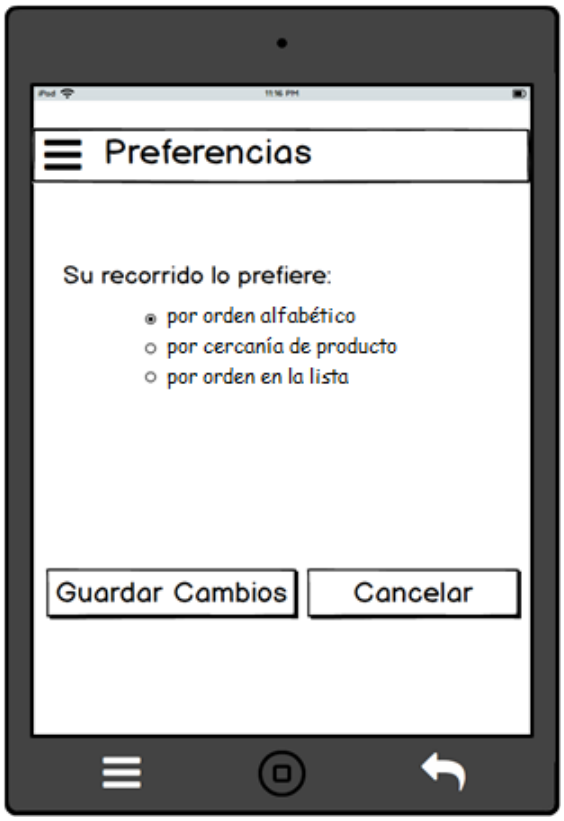


Figura 3.15: Preferencias de Compra.

- Configuración General

Por último, la pantalla de la Figura 3.16 corresponde a la configuración general del sistema. Se puede acceder desde la opción correspondiente en el menú principal (mostrado en la Figura 3.3 y Figura 3.4). Se despliegan los datos del usuario, siendo editables los campos email y contraseña. Para confirmar los cambios realizados se debe accionar el botón *Guardar cambios*, caso contrario las modificaciones no se verán reflejadas en el sistema.

Figura 3.16: Configuración General de la aplicación.

De esta manera, se pudieron apreciar las pantallas relacionadas a la problemática que se busca resolver. En particular, la funcionalidad a modelar estará relacionada a los recorridos de compra, posicionamiento de los diferentes elementos y movilidad del usuario.

Acorde a lo anteriormente explicado y a partir del análisis de las pantallas se identificaron los siguientes conceptos relevantes:

- Supermercado
- Usuarios/Clientes
- Lista de compra
- Tres tipos de elementos: Sector, Góndola y Producto
- Recorrido
- Posicionamiento (punto y área dentro del establecimiento)

Nos basaremos en estos conceptos para desarrollar el modelo propuesto, el cual se presenta en la siguiente sección.

3.2 Descripción del modelo propuesto

Tomando como base los conceptos identificados en la Sección 3.1, en esta sección se presentará de manera incremental el modelo propuesto para dar solución a dicha problemática. Se utilizarán diagramas de clases UML para presentar las distintas clases.

La primera clase que surge es el *Supermercado* la cual se puede apreciar en la Figura 3.17. Esta clase respeta el patrón de diseño *Singleton* [Johnson et al., 1995], esto garantiza que tenga una única instancia. Esto permite disponer de todos los datos del supermercado como así también todos los métodos necesarios para obtener cualquier tipo de información sobre el mismo.
La clase *Supermercado* tiene como atributo *clientes*, que es una colección de la clase *Cliente* con cardinalidad uno a muchos, esto significa que como mínimo existe al menos un cliente. La interacción que se produce en ambas clases queda representada por el patrón de diseño *Observer* [Johnson et al., 1995], la ventaja de utilizarlo es no sobrecargar al *Cliente* con demasiada información. La funcionalidad que los relaciona genera eventos de pedidos. El *Supermercado* observa al *Cliente*, captura sus eventos, los procesa y retorna el resultado.
Además, tiene asociado una clase *ManagerElemento* a la cual podrá delegar la funcionalidad relacionada a la gestión de los elementos que forman parte del supermercado (productos, góndolas y sectores).

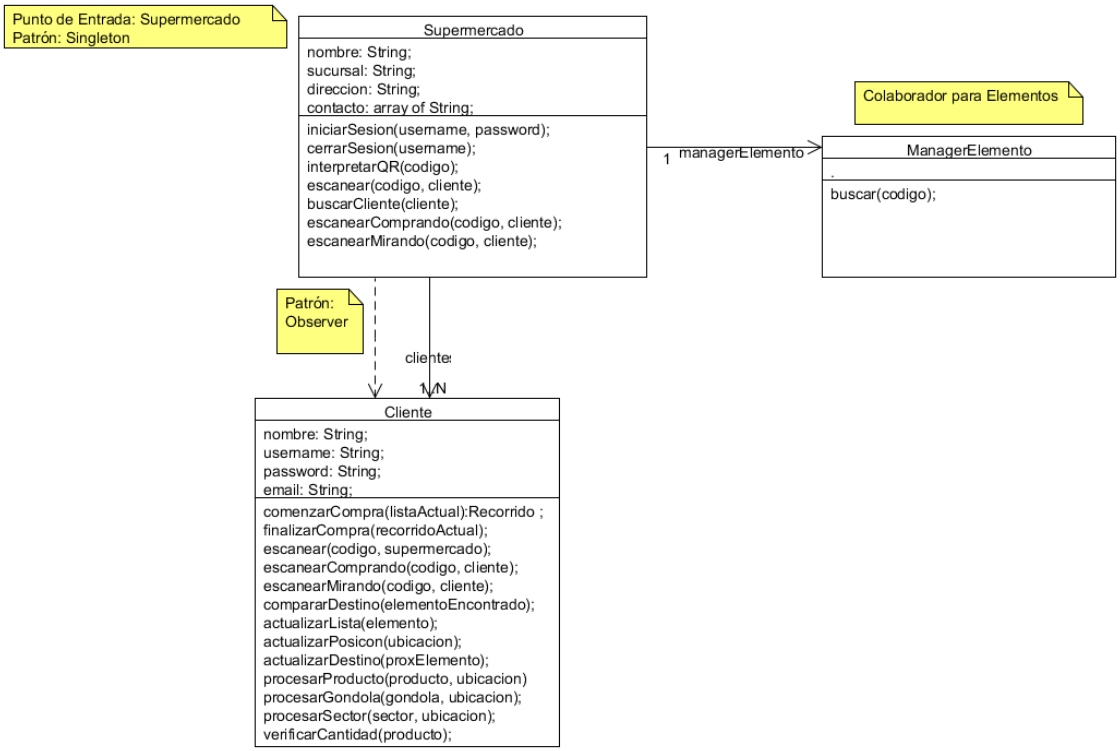


Figura 3.17: La clase *Supermercado* con su *ManagerElemento* y sus *Clientes*.

La clase *ManagerElemento* tiene una colección de la clase *Elemento*, con cardinalidad uno a muchos (como mínimo tiene un elemento) como se puede observar en la Figura 3.18. El beneficio de tener los métodos de alta baja, modificación y búsqueda en la clase *ManagerElemento* es desacoplar esta funcionalidad de la clase *Supermercado* y delegarla a

este colaborador. Si bien en esta figura se presenta la jerarquía de *Elemento* los mismos se verán con más nivel de detalle más adelante.

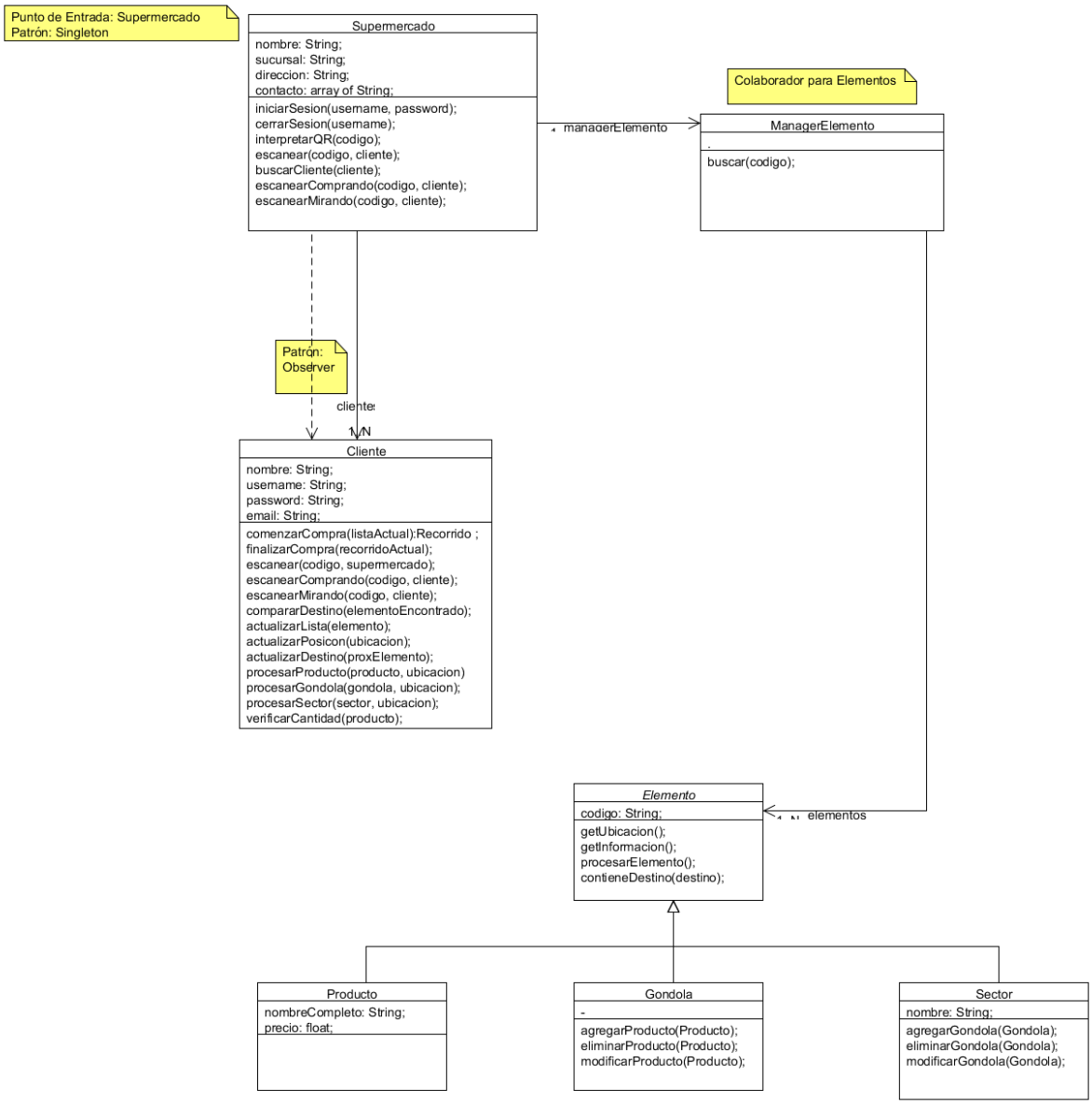


Figura 3.18: Se agrega la relación de la clase *ManagerElemento* con la clase *Elemento*.

En la Figura 3.19 se puede ver la clase *Cliente* y sus dependencias. El *Cliente* puede tener ningún o muchos recorridos realizados, representados por la clase *HistorialRecorrido*, además tiene asociado ningún o un solo *recorridoActual* con la clase *Recorrido.* El atributo *estadoActual* establece una relación con la clase *Estado*, esto define el estado en que se encuentra el cliente en un preciso momento y puede ser *Comprando, Mirando, ArmandoLista* o *Inactivo.* El atributo *posicionActual* está definido como una relación con la interfaz *Position*, y determina en qué lugar se encuentra el cliente durante la compra. La clase *Cliente* tiene relación con la clase *Lista* definida por el atributo *historialListas*, representada por una colección de listas ordenadas por fecha de manera ascendente. Dichas listas son aquellas que el usuario creó y mantuvo guardadas en el historial, sin importar si realizó la compra. Por último, el atributo *preferencia* (de la clase *Cliente*) se define como una relación con la clase

Preferencia, y representa la opción de compra elegida por el cliente, puede ser por *OrdenAlfabético*, *OrdenLista* o *Cercania*. El sistema se basará en esta elección al momento de armar el recorrido de compra. Cabe aclarar que de la interfaz *Position,* las clases *HistorialRecorrido, Recorrido* y *Lista*, la jerarquía de *Preferencias* y de *Estado* se profundizarán más adelante en más detalles.

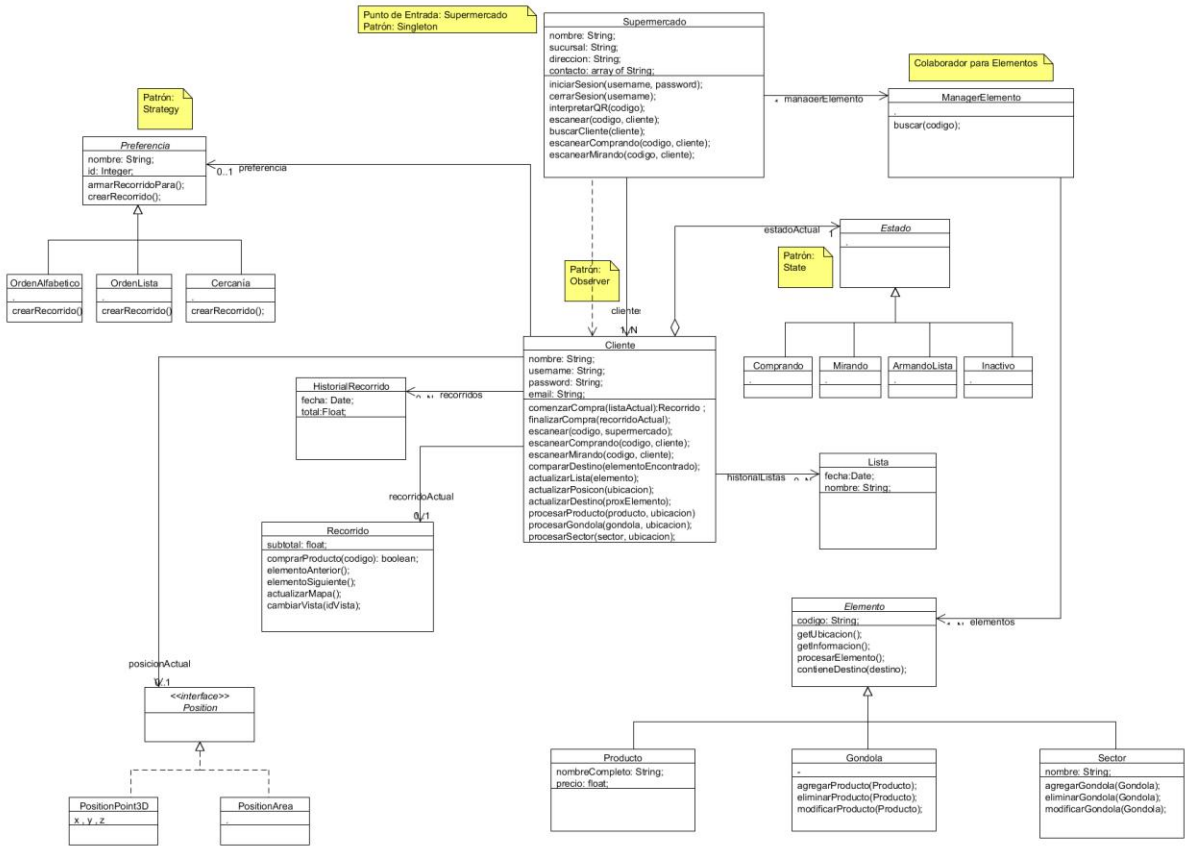


Figura 3.19: Clase *Cliente* y sus relaciones.

Veamos con más detalles los métodos definidos para esta clase Cliente:

- comenzarCompra(listaActual):Recorrido; este método se invoca cuando el cliente indica que quiere comenzar su compra. Recibe como parámetro *listaActual*, que es el último elemento cargado en *historialListas*. Según la preferencia de compra que esté seteada en ese instante, se crea el *recorridoActual* y todas sus dependencias.
- finalizarCompra(recorridoActual); este método se invoca cuando el cliente indica que finalizó una compra. Recibe como parámetro el *recorridoActual*, con el cual crea una instancia de *HistorialRecorrido* y la agrega a la colección recorridos para mantener el registro de dicha compra.
- escanear(código,supermercado):Elemento**;** a partir de un código puede escanearse cualquier elemento ya sea un *Producto*, una *Góndola* o un *Sector*, para conocer su información.
- escanearComprando(código,supermercado); método que se invoca al escanear un elemento y el estado del cliente es *Comprando*.
- escanearMirando(código,supermercado); método que se invoca al escanear un elemento y el estado del cliente es *Mirando*.

- compararDestino(elementoEncontrado); este método verifica si el elemento que fue escaneado por un cliente que está comprando es igual al elemento destino, es decir, que está en el lugar que se le indicó en el recorrido.
- actualizarLista(elemento); se marca el elemento pasado por parámetro como comprado o visitado según corresponda.
- actualizarPosicion(ubicacion); una vez finalizado el escaneo de un elemento el sistema puede determinar la posición actual del cliente para poder actualizarla invocando este método.
- actualizarDestino(proxElemento); se realiza la actualización del destino seteando el mismo con el próximo elemento de la lista de compras en caso que no haya finalizado la compra, ya que si la compra está finalizada no va a existir un elemento para comprar.
- verificarCantidad(producto); lo que realiza este método es verificar en la lista de compra la cantidad de productos que el cliente había ingresado al momento de generar la lista y cuántos está llevando al momento de realizar la compra.
- procesarProducto(producto,ubicacion); una vez finalizado el escaneo del código de un producto, a través de este método se obtiene la ubicación del mismo para determinar la posición actual del cliente.
- procesarGondola(gondola,ubicacion); una vez finalizado el escaneo del código de una góndola, a través de este método se obtiene la ubicación del mismo para determinar la posición actual del cliente.
- procesarSector(sector,ubicacion); una vez finalizado el escaneo del código de un sector, a través de este método se obtiene la ubicación del mismo para determinar la posición actual del cliente.

Ciertas funcionalidades del sistema a las que el cliente puede acceder dependen específicamente de su estado actual, de qué es lo que está haciendo en ese momento. Por ejemplo, si el cliente escanea el código de un producto:

- si su estado es *Mirando* o *Inactivo*, se despliega el detalle del producto,
- si su estado es *ArmandoLista* de compra, además de mostrar el detalle del producto, se brinda la posibilidad de agregarlo a la lista actual,
- si su estado es *Comprando*, además de mostrar el detalle del producto, se brinda la posibilidad de agregarlo como producto comprado.

En este caso, el comportamiento que se desencadena al escanear un producto depende del estado actual del cliente, y debe cambiar en tiempo de ejecución. Por lo tanto, el estado se representa usando el patrón de diseño *State* [Johnson et al., 1995] que permite que un objeto modifique su comportamiento cada vez que cambie su estado interno. Estos estados se pudieron visualizar en la Figura 3.19 como una superclase *Estado* de la que heredan las clases *Comprando*, *Mirando*, *ArmandoLista* e *Inactivo*. Cada subclase implementa un comportamiento asociado con el estado del cliente. El cliente delega las peticiones que dependen de su estado en la variable *estadoActual*.

Veamos con más detalle la clase *Lista* mencionada anteriormente. Esta clase va a tener como atributos *fecha*, *nombre* y asociado un conjunto de elementos de tipo *Elemento* como se puede apreciar en la Figura 3.20.

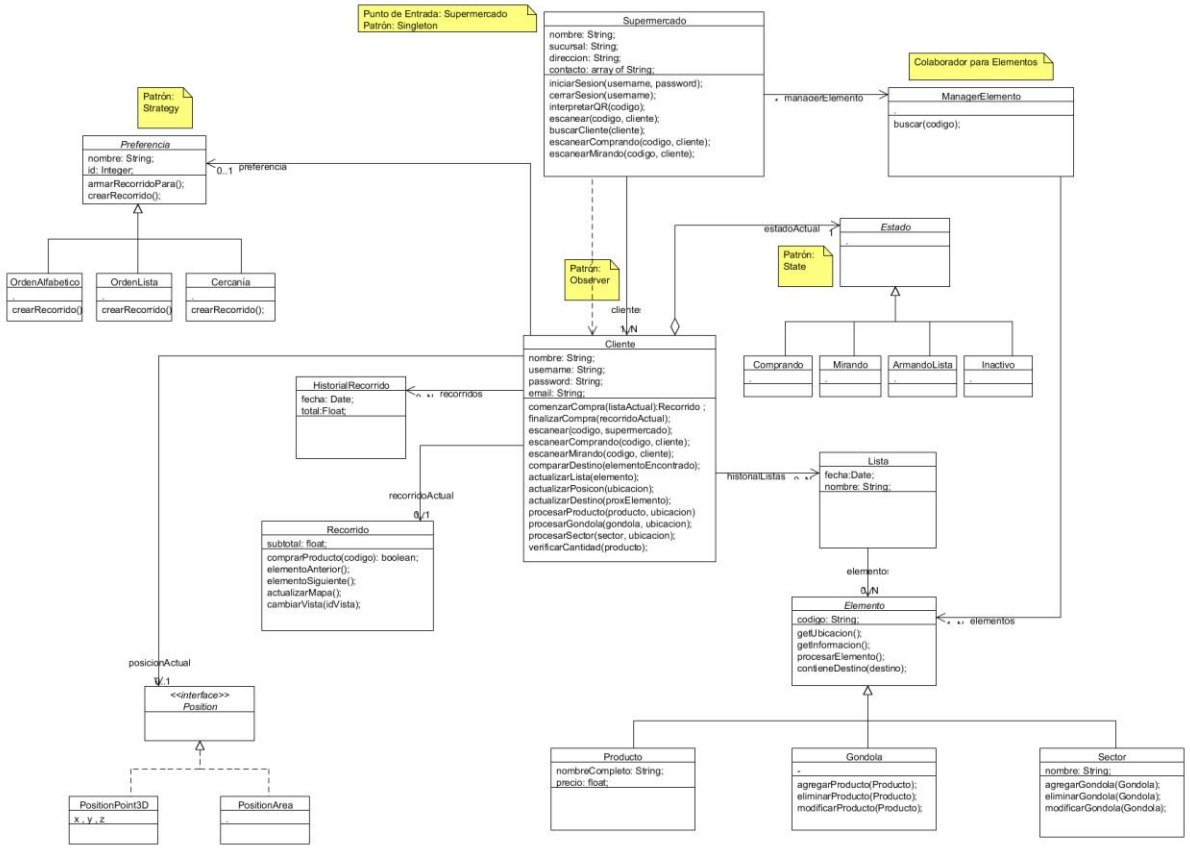


Figura 3.20: Clase *Lista*

Veamos con más detalle la clase *Preferencia* ya mencionada anteriormente (se puede apreciar la misma en la Figura 3.20). El *Supermercado* se basará en esta elección de la preferencia (por orden de la lista, cercanía del producto, orden alfabético) al momento de armar cada recorrido de compra. Para modelar este comportamiento aplicamos el patrón de diseño *Strategy* [Johnson et al., 1995], el cual define un conjunto de algoritmos, encapsula cada uno de ellos en una estrategia y los hace intercambiables. Las estrategias permiten configurar una clase con un determinado comportamiento entre muchos posibles, y que esta configuración puede variar independientemente de los clientes que lo usen. Por lo tanto tenemos una jerarquía donde la clase *Preferencia* es la clase abstracta y cada una de sus subclases representa una estrategia a la hora de armar el recorrido de una lista de compras, teniendo en cuenta un criterio específico de orden. La herencia nos permite sacar factor común de la funcionalidad de cada preferencia, ubicándolo en el método armarRecorridoPara() en la superclase. Mientras que las subclases *OrdenAlfabetico*, *OrdenLista* y *Cercania* implementan el método crearRecorrido() en el cual aplican su estrategia específica.

La clase *Elemento* como se mostró en la Figura 3.20 es una clase abstracta y forma parte de una jerarquía para representar los elementos del *Supermercado* como ya se mencionó anteriormente. Cuenta con tres subclases, cada una de ellas tiene un código como atributo con el cual se puede identificar el elemento específico. Además, cada elemento tiene una posición representada por una relación con la interface *Position* con cardinalidad obligatoria.

- *Producto*: tiene como atributo *nombreCompleto* y *precio*.
- *Gondola*: tiene una colección de productos, con cardinalidad de cero a muchos, y métodos para gestionarla. Además tiene asociada una *Categoría*, con cardinalidad obligatoria.
- *Sector*: tiene un atributo nombre. Además cuenta con una colección de góndolas, con cardinalidad de cero a muchos, y métodos para gestionarla.

Uno de los métodos importantes que se define en la clase *Elemento* es el método contieneDestino(destino) el cual se invoca en aquellos casos donde el cliente con estado *Comprando* escanea un elemento que no coincide con el elemento destino de su recorrido y lo que hace es verificar si el elemento destino está contenido en alguna de sus colecciones devolviendo *Sí* o *No* según corresponda. Por ejemplo, supongamos el caso que el cliente tiene como destino el producto "*Leche*" pero escanea el código del sector "*Lácteos*", el elemento sector verifica en su colección góndolas si contiene el producto destino, en caso de encontrarlo devuelve *Sí*, caso contrario, devuelve *No*.

Otra clase que surge es la *Categoría* que se puede apreciar en la Figura 3.21 tiene como atributo *nombre*, en el caso que no exista una categoría específica toma como valor "*Sin especificar*".

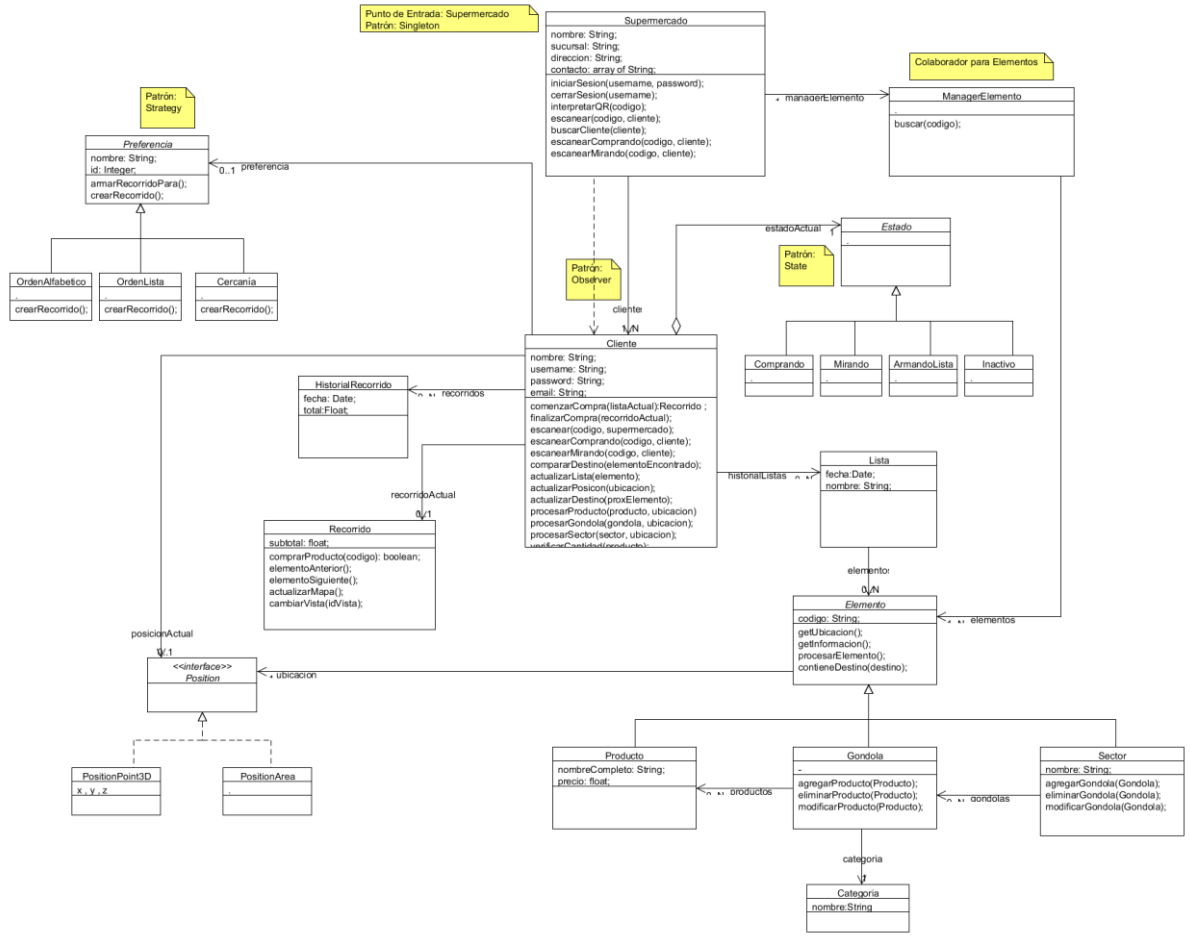


Figura 3.21: Clase *Categoría*

En la Figura 3.22 se muestran más detalles de la clase *Historial Recorrido* mencionada anteriormente. Esta clase tiene como objetivo llevar un registro de todos los recorridos

realizados, que por ende fueron compras que el cliente realizó y finalizó. Tiene la *fecha*, el *total de la compra* y tiene una relación con la clase *Recorrido* (que representan los recorridos realizados).

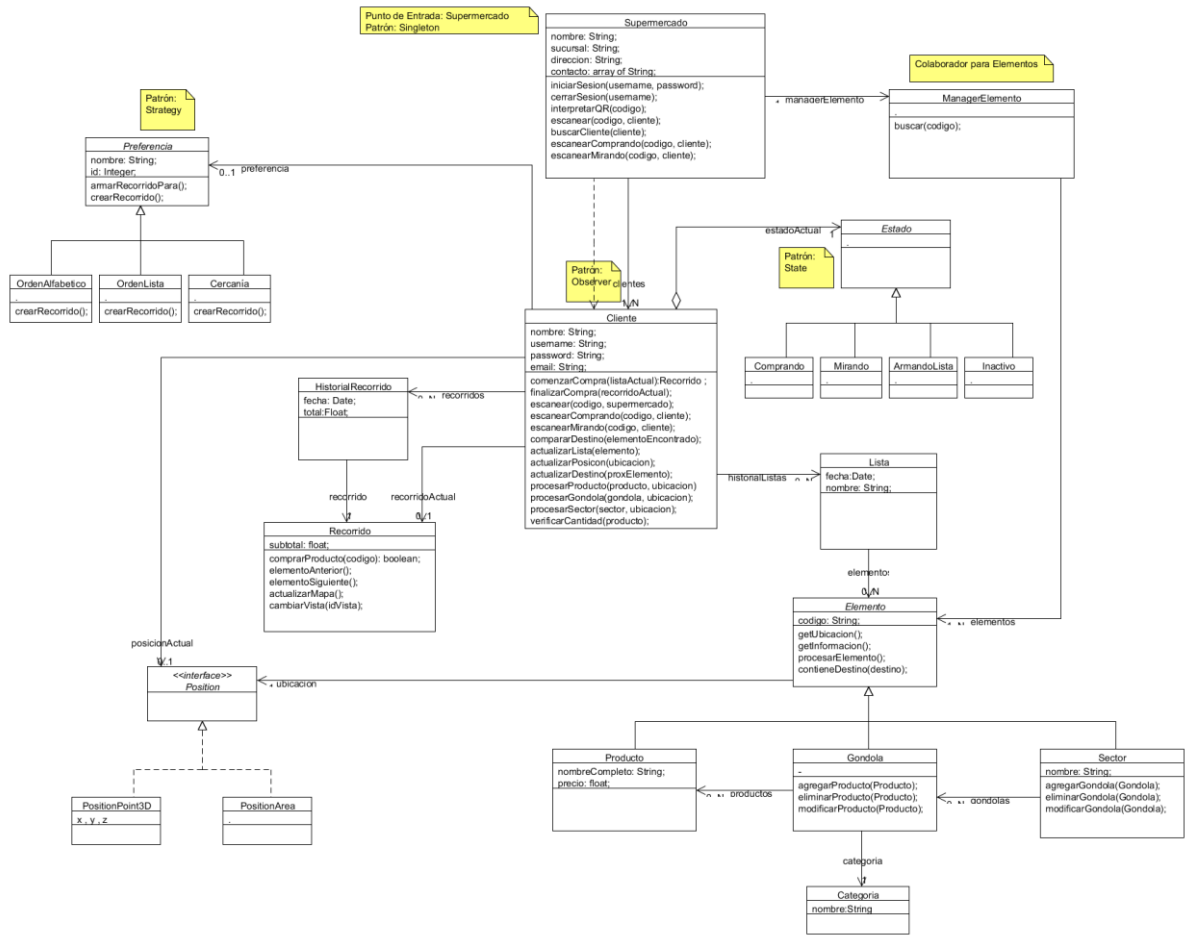


Figura 3.22: Clase *Historial Recorrido*

Veamos más detalles de la clase *Recorrido* mencionada anteriormente, como se puede ver en la Figura 3.23. Esta clase tiene una variable de instancia para llevar el subtotal gastado de la compra en un momento dado. Además, tiene relación con la interface *Position* definida por el atributo *historialPosiciones*, para tener un registro de todos los lugares en los que estuvo el cliente. Por otro lado, también tiene una relación con la clase *Lista* llamada *listaOrdenada*. Esta lista se debe asignar en el momento que se crea el recorrido, y el criterio de orden dependerá de la preferencia del *Cliente*. La relación con la clase *Elemento*, queda definida por el atributo *destino* que determina dónde debe dirigirse el cliente cuando está realizando la compra. Por último, tiene una colección llamada *productosComprados* definida por la relación con la clase *Producto*, donde se irán agregando los productos comprados a lo largo del recorrido.

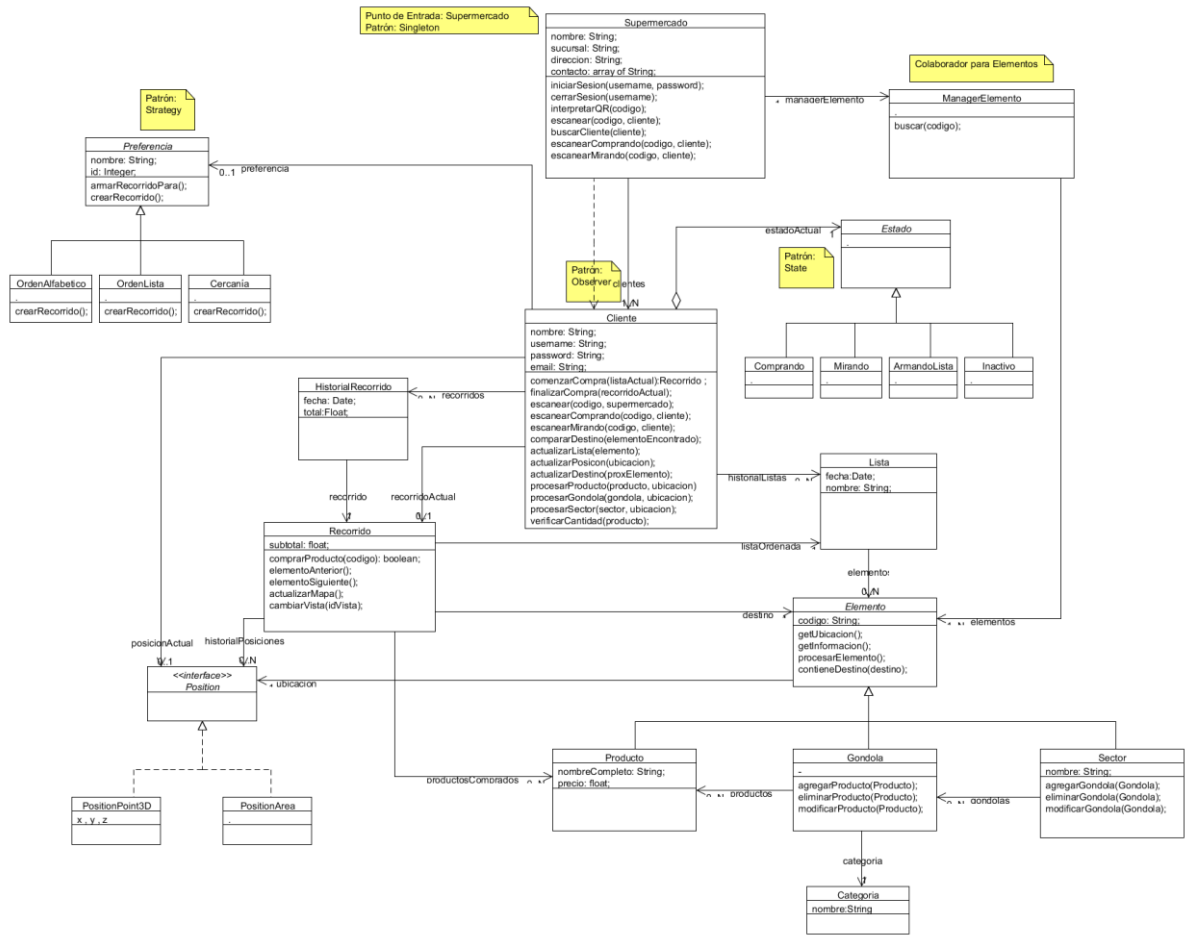


Figura 3.23: Clase *Recorrido*

Veamos los métodos del *Recorrido* listado en la Figura 3.23.

- comprarProducto(codigo): boolean; recibe como parámetro el código de un producto (precondición: el producto siempre va a existir). Actualiza el subtotal sumando el precio del mismo, y lo agrega a la colección *productosComprados*. Este método solo se puede invocar durante la compra.
- elementoAnterior(); retorna el *Elemento* anterior al actual excepto que el actual sea el primero. Este método solo se puede invocar durante la compra.
- elementoSiguiente(); retorna el próximo *Elemento* al actual excepto que el actual sea el último. Este método solo se puede invocar durante la compra.
- actualizarMapa(); genera una nueva imagen donde se indica el camino que deberá recorrer el cliente para llegar al *Elemento* actual, basándose en la opción de vista seteada (*Sector*, *Gondola*, *Producto*)
- cambiarVista(); setea la opción de vista elegida e invoca al método *actualizarMapa*;

Veamos ahora más detalles de la interfaz *Position* mencionada anteriormente, como se puede apreciar en la Figura 3.24. Esta clase a diferencia de las ya explicadas es una *interfaz*, es decir, no contiene ninguna implementación sino que define un protocolo de mensajes que definen las clases que implementan esta interfaz. Se definen dos clases concretas que implementan esta interfaz:

- La clase *PositionPoint3D*: cuenta con tres variables de instancia, que representan los tres puntos en cada eje (x, y, z).
- La clase *PositionArea*: tiene dos relaciones con instancias de tipo *PositionPoint3D*, es decir, dos puntos en el plano. A partir de estos dos puntos es que se obtiene el área que se desea representar.

De esta manera, en la Figura 3.24 queda representado el modelo propuesto para la problemática planteada en la Sección 3.1.

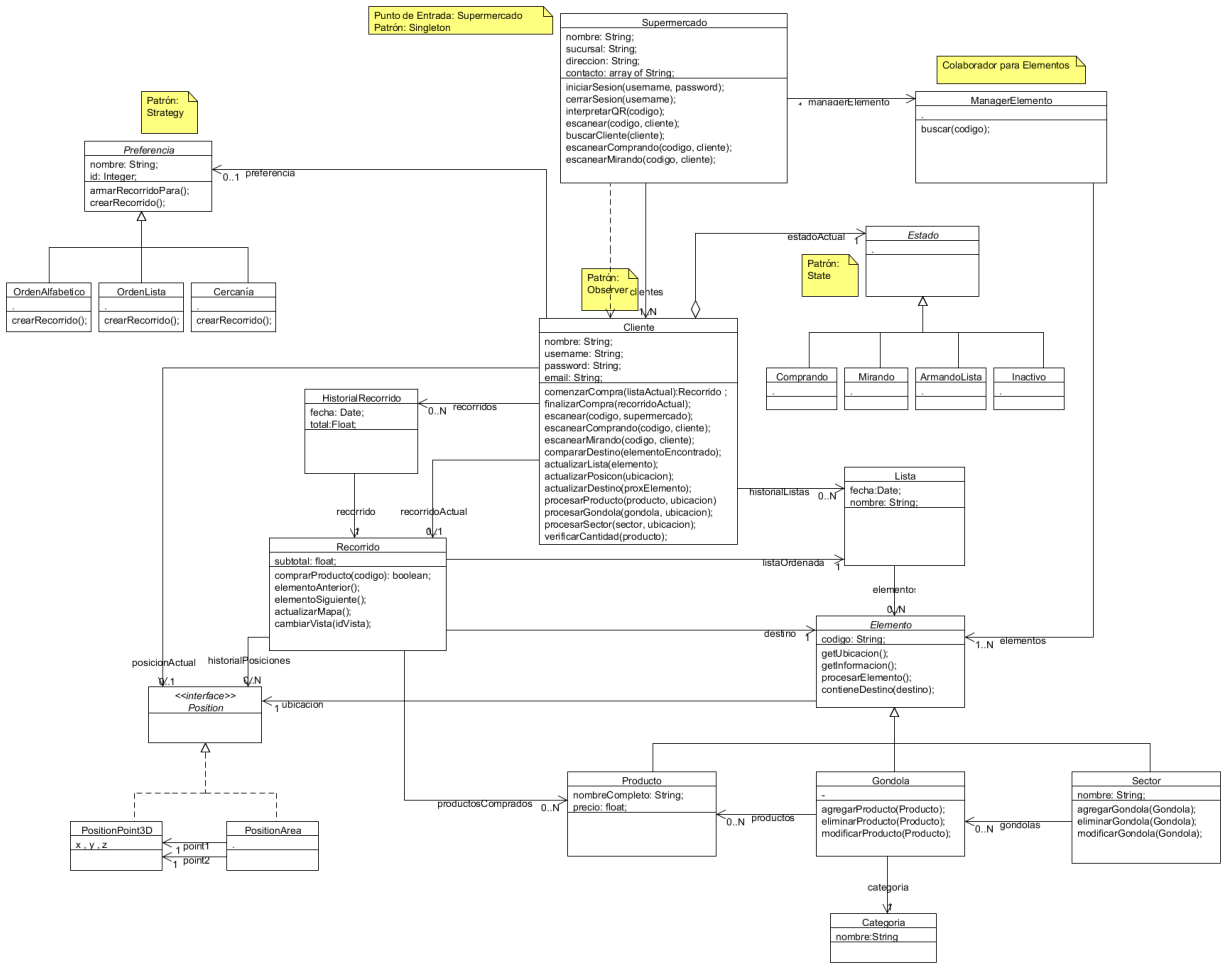


Figura 3.24: Modelo Propuesto

Para comprender mejor el comportamiento del usuario cuando escanea un elemento mostraremos la interacción del mismo a través de diferentes diagramas de secuencia.

- *Escanear código mientras el cliente está mirando.*

  En la Figura 3.25 se puede apreciar el diagrama de la secuencia a seguir cuando se escanea un código mientras el cliente está mirando. En este diagrama el evento inicial se produce cuando el usuario escanea un elemento, que puede ser un producto, una góndola o un sector. Esto desencadena varios eventos: el *Supermercado* busca al *Cliente* que realizó el escaneo, una vez que lo encuentra verifica el estado del cliente, en este caso el estado es *Mirando*. Se realiza la búsqueda del elemento y una vez finalizada el

Supermercado conoce la ubicación del elemento por lo que actualiza la posición del cliente y retorna al usuario la información del elemento escaneado.

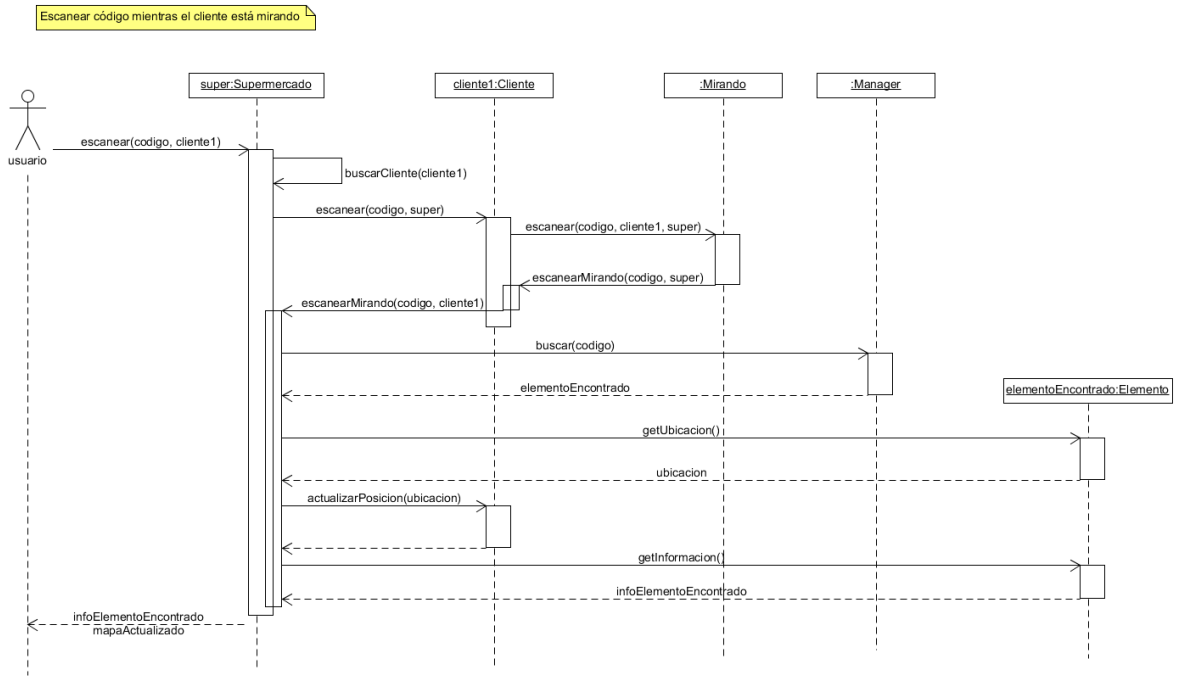


Figura 3.25: Escanear código mientras el cliente está mirando

- *Escanear código de producto mientras el cliente está comprando y su posición es igual al destino.*

  En la Figura 3.26 se puede apreciar que el evento que inicia la secuencia es el escaneo de un producto. El *Supermercado* realiza la búsqueda del cliente para verificar el estado en que se encuentra, siendo para este caso el estado actual del cliente igual a *Comprando*. Finalizada la búsqueda del elemento se comprueba que el producto tenga la misma ubicación que el destino, es decir, es el producto que hay comprar. Siendo el producto que hay que comprar se actualiza la lista de compras y se le informa al usuario el próximo elemento de la lista junto con el mapa actualizado.

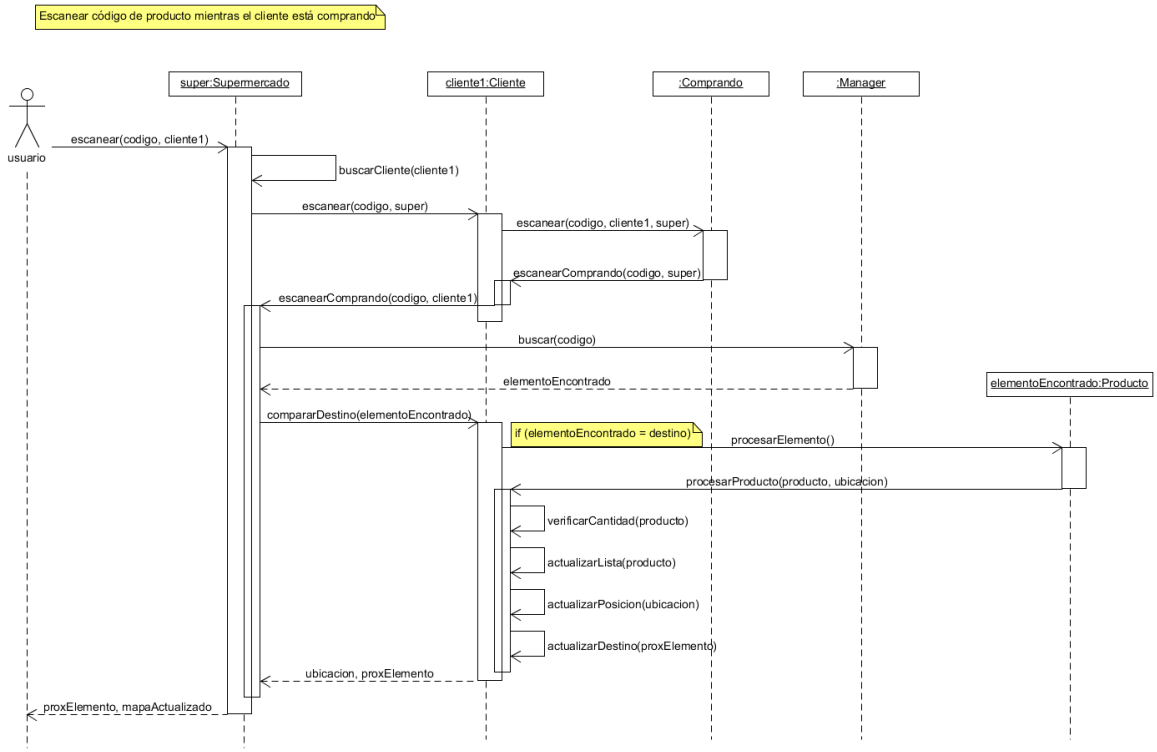


Figura 3.26: Escanear código de producto mientras el cliente está comprando y su posición es igual al destino.

- *Escanear código de producto mientras el cliente está comprando y su posición es diferente al destino.*

  El diagrama presentado en la Figura 3.27 es análogo al descrito anteriormente (Figura 3.26), con la diferencia que el producto que se escanea no es igual al que hay que comprar, por lo que el sistema actualiza la posición del cliente y le sugiere comprar el producto o seguir el orden de la lista de compra devolviendo un mapa de recorrido actualizado.

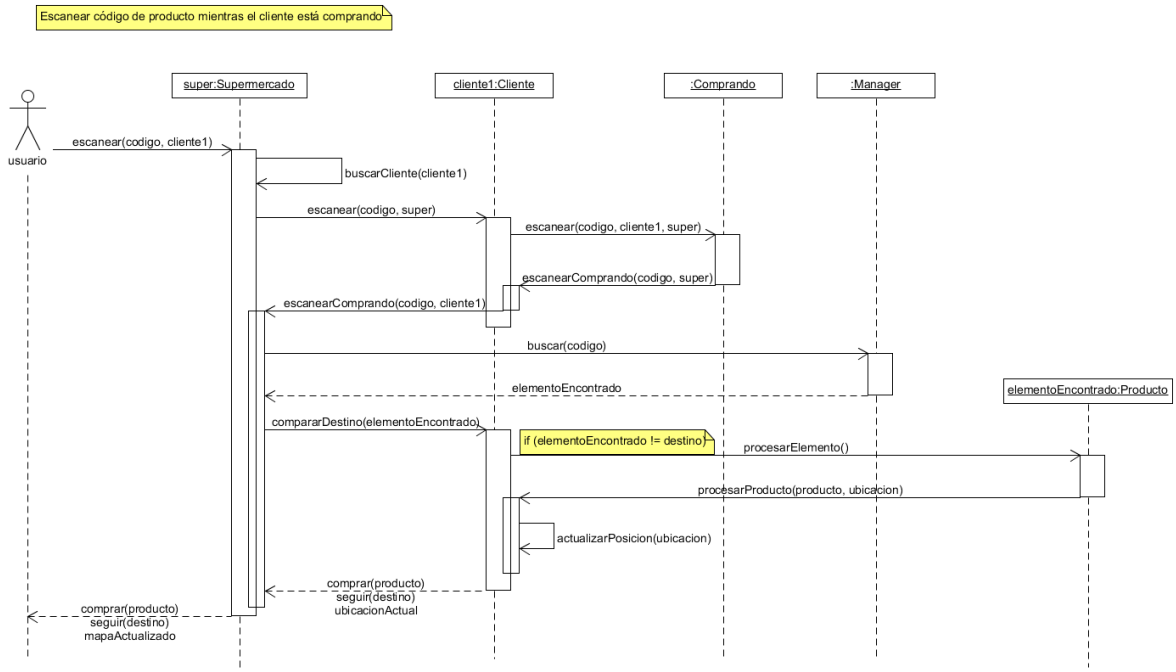


Figura 3.27: Escanear código de producto mientras el cliente está comprando y su posición es diferente al destino.

- *Escanear código de góndola mientras el cliente está comprando y su posición es igual al destino.*

  El diagrama de la Figura 3.28 es similar al presentado en la Figura 3.26 pero sin la opción de compra de producto, porque en este caso es llegar a una góndola.

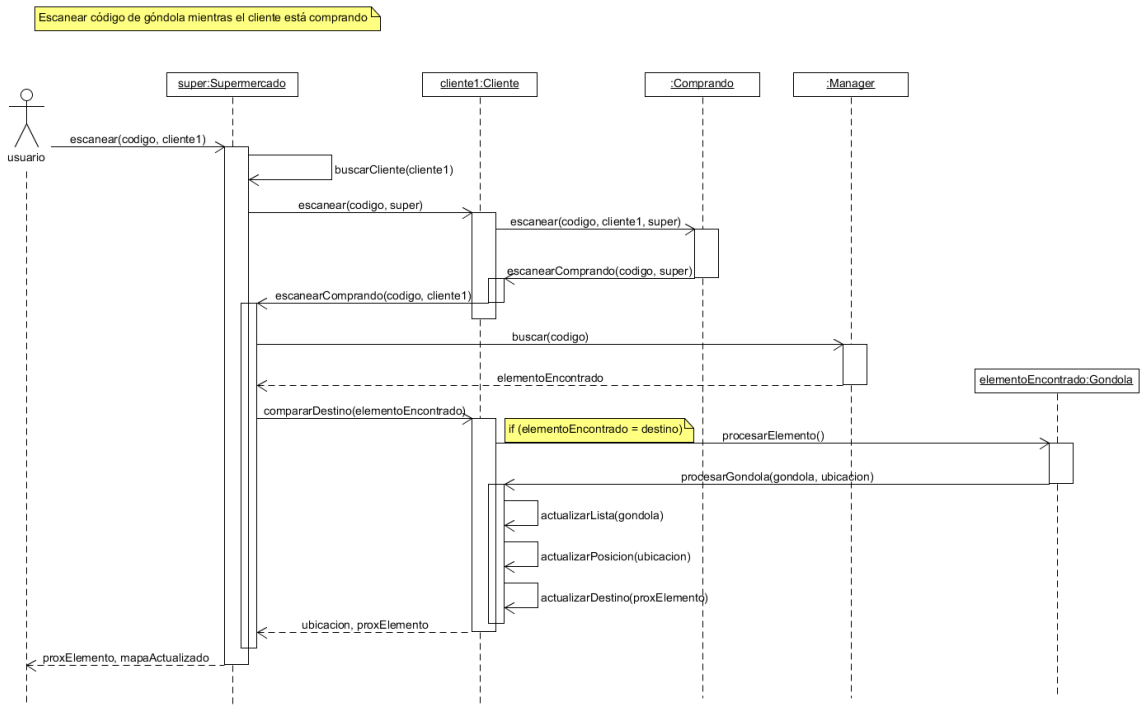


Figura 3.28: Escanear código de góndola mientras el cliente está comprando y su posición es igual al destino.

- *Escanear código de góndola mientras el cliente está comprando y su posición es diferente al destino, con la particularidad que la góndola contiene el producto o sector destino.*

  En la Figura 3.29 se puede apreciar la secuencia de escanear un código de góndola mientras el cliente está comprando y su posición es diferente al destino, con la particularidad que la góndola contiene el producto o sector destino. Es decir, es el próximo a comprar, el sistema devuelve el mapa actualizado con una pantalla similar a la mostrada en la Figura 3.14.3.

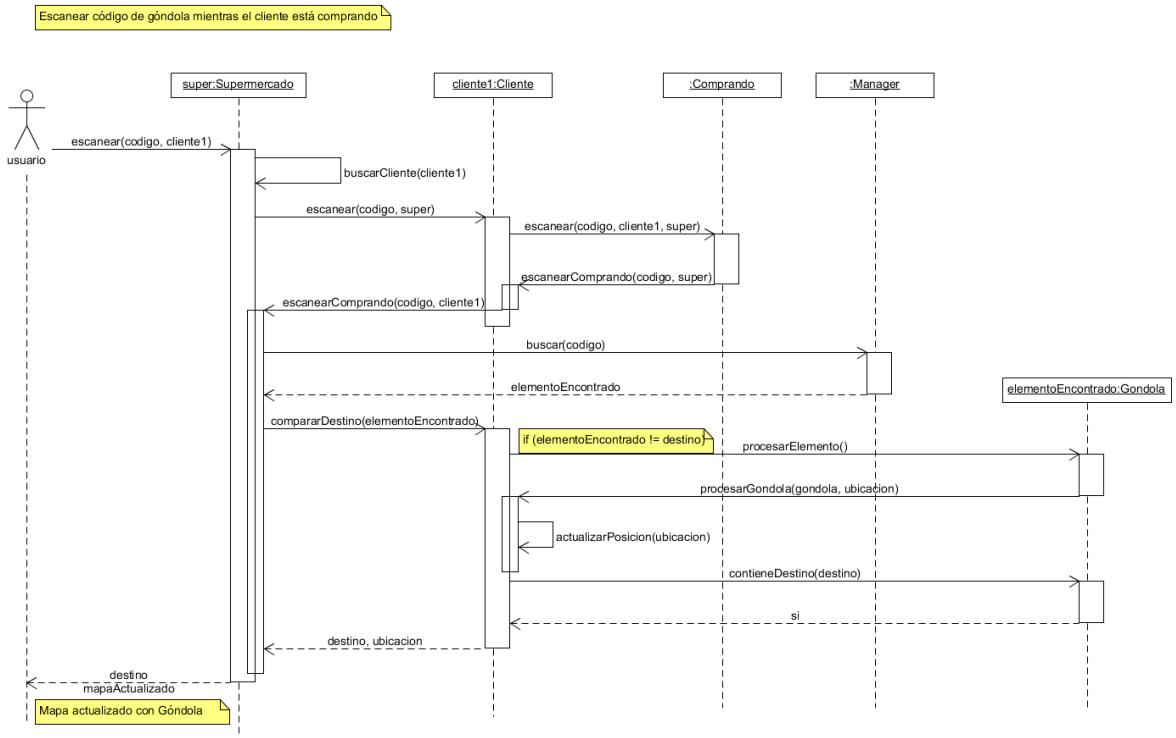


Figura 3.29: Escanear código de góndola mientras el cliente está comprando y su posición es diferente al destino, con la particularidad que la góndola contiene el producto o sector destino.

- *Escanear código de sector mientras el cliente está comprando y su posición es igual al destino.*

  En la Figura 3.30 se puede apreciar la secuencia de escanear un código de sector mientras el cliente está comprando y su posición es igual al destino. Esta secuencia es similar a la presentada en las Figura 3.28 pero en este caso es un sector en vez de una góndola.

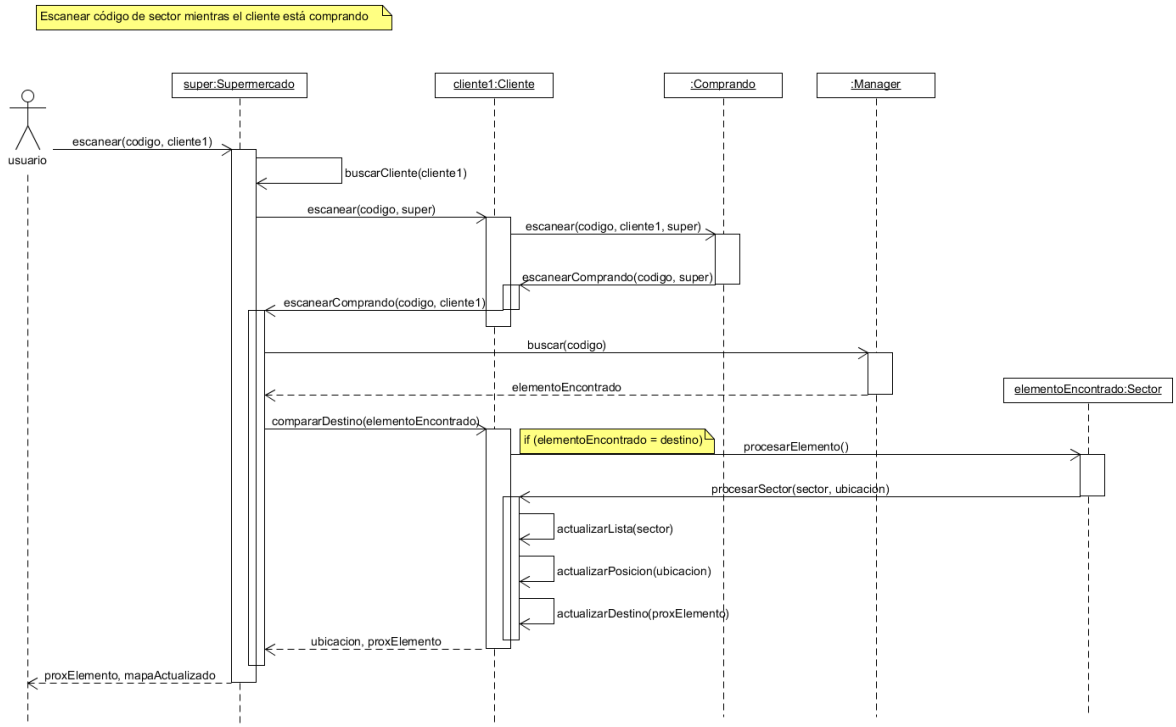


Figura 3.30: Escanear código de sector mientras el cliente está comprando y su posición es igual al destino.

- *Escanear código de sector mientras el cliente está comprando y su posición es diferente al destino, con la particularidad que el sector contiene el producto o góndola destino.*

  En el diagrama de secuencia de la Figura 3.31, se escanea un sector mientras el estado del cliente es comprando y se verifica que dicho sector no es igual al destino proporcionado por el próximo elemento de la lista de compras, con la particularidad que el sector contiene el producto o la góndola donde había que ir. El sistema devuelve el mapa actualizado con la pantalla similar a la mostrada en las Figuras 3.14.2 o 3.14.3 según corresponda.

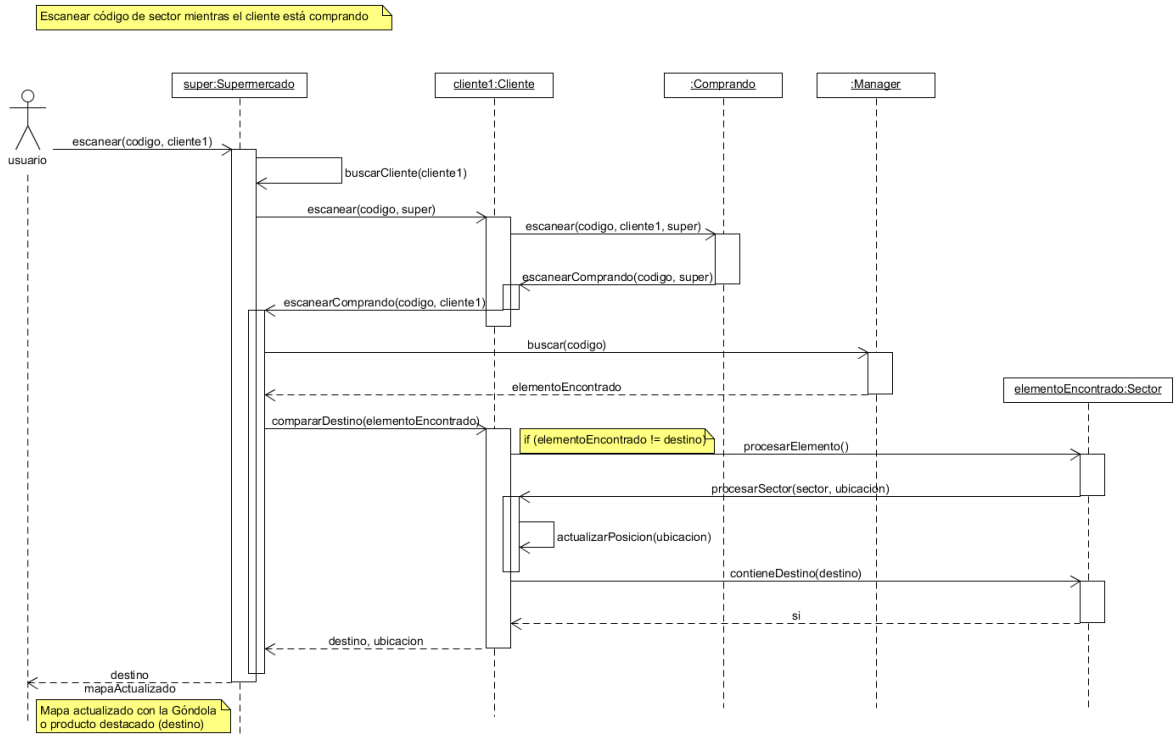


Figura 3.31: Escanear código de sector mientras el cliente está comprando y su posición es diferente al destino, con la particularidad que el sector contiene el producto o góndola destino.

De esta manera, mediante diferentes diagramas de secuencias se pudo apreciar el comportamiento del modelo, en particular cuando se escanean diferentes códigos QR, que representan productos, góndolas y sectores, siendo los mismos destinos a los que se dirigía el usuario o no. Este comportamiento será la base del prototipo implementado para lograr brindar dicho comportamiento.

4. PROTOTIPO DESARROLLADO

El prototipo se realizó en base al modelo propuesto en el Capítulo 3, y el foco del mismo está puesto en el posicionamiento del usuario dentro del supermercado. Acorde a esto, fueron las funcionalidades que se terminaron desarrollando como se describirán más adelante. Cabe aclarar que el prototipo está acotado a un único *Supermercado* y un único *Usuario*.

El prototipo fue desarrollado usando *PhoneGap*, que es un framework para el desarrollo de aplicaciones móviles híbridas que funciona como una solución multi-plataforma y que permite usar las tecnologías web:

- HTML *(HyperText Markup Language* – Lenguaje de Marcado de HiperTexto)
- CSS3 (*Cascading Style Sheets* - Hojas de Estilo en Cascada)
- JavaScript

Varios autores (por ejemplo, [Heitkötter et al., 2012], [Dalmasso et al., 2013] y [Lim, 2015]) coinciden que *PhoneGap* es la plataforma más popular y con más documentación. Esta es la razón de la elección realizada, además de poder trabajar con tecnología web.

Cabe destacar que con *PhoneGap*, a partir de un solo código cuya lógica de programación está sustentada en el lenguaje de programación web *JavaScript*, se puede obtener aplicaciones para múltiples plataformas móviles. Por lo tanto, las aplicaciones resultantes son híbridas, es decir no son realmente aplicaciones nativas al dispositivo (ya que el renderizado es realizado mediante vistas web y no con interfaces gráficas específicas a cada sistema). Sin embargo, no se tratan tampoco de aplicaciones web puras. Son empaquetadas para ser desplegadas en el dispositivo móvil para poder correr sobre el sistema nativo.

Para el desarrollo del prototipo, el cual denominamos *Asistencia Indoor de un Supermercad*o, decidimos realizarlo con *PhoneGap* y en particular empaquetarlo para *Android*.

Debido a problemas surgidos con la aplicación http://build.phonegap.com/ al momento de compilar nuestro código y generar los APKs correspondientes, tuvimos que buscar una manera alternativa para generarlos desde consola de comandos. A continuación, detallamos los pasos que seguimos para lograr tener un entorno funcionando.

Para el entorno de desarrollo en Windows instalamos los siguientes programas:

- Nodejs[4] versión 6.9.1
- *NPM* [5](Repositorio para instalar paquetes) versión 3.10.8.
- *Cordova versión 6.4.0*[6], desde consola con el comando:

  npm install -g cordova
- *Phonegap versión 6.3.5*, desde consola con el comando:

  npm install -g phonegap

[4] Página de *Nodejs*: https://nodejs.org/es (Último Acceso: 19/12/2016)
[5] Página de *NPM*: https://www.npmjs.com (Último Acceso: 19/12/2016)
[6] Página de *Cordova* versión 6.4.0: https://cordova.apache.org/news/2016/10/25/tools-release.html (Último Acceso: 19/12/2016)

- *Java Development Kit* (JDK)[7] versión 1.8.0_65. Además, seteamos la variable de entorno JAVA_HOME y agregamos a la variable:

  PATH: %JAVA_HOME%/bin

- *Android SDK*[8]. *Además, s*eteamos la variable de entorno ANDROID_HOME y agregamos a la variable:

  PATH: %ANDROID_HOME%/tools; %ANDROID_HOME%/platforms-tools

  Desde el programa SDK Manager instalamos los paquetes:

  - *Android Platform SDK API 22* (para *Android* 5.1.1)
  - *Android Platform SDK API 24* (para *Android* 7.0)
  - *Android Platform SDK API 23* (para *Android* 6.0, que fue la que se terminó usando para el prototipo),
  - *Android SDK Tools*,
  - *Android SDK Platform-tools* y
  - *Android SDK Build-Tools,*

  y en la pestaña Extra, *Android Support Repository*.

- Una vez instalado todo correctamente, pudimos instalar la plataforma en la cual probamos nuestro prototipo. Para esto desde consola nos posicionamos dentro de la carpeta del proyecto y corrimos el comando:

  cordova platform add android@6.0.0

  Se instaló la versión de *Android* 6.0.0 correspondiente a las versiones de los dispositivos con los cuales contamos para realizar las pruebas. Así se crea una carpeta en 'platforms\android' con la versión de *Android* que se definió.

- Después, se tiene que construir el proyecto con el comando:

  cordova build android

- Por último, conectamos el dispositivo con la opción de *Depuración USB activada*, y ejecutamos:

  cordova run android

  Este comando actualizará el build, si aplica, generará el apk correspondiente y lo instalará en el dispositivo conectado. El apk resultante se puede encontrar en la carpeta 'platforms\android\build\outputs\apk'.

Cabe mencionar, que el dispositivo debe tener la versión del SO que se definió, en este caso *Android 6.0.0*, en caso de tener un SO con una versión anterior, se puede emular un dispositivo *Android* con el programa *Android Studio* o instalar otra versión de la plataforma *Android*.[9]

[7] Página de JDK: http://www.oracle.com/technetwork/java/javase/downloads/index.html (Último Acceso: 19/12/2016)
[8] Página de *Android* SDK: https://developer.android.com/studio/index.html?hl=es-419 (Último Acceso: 19/12/2016)
[9] Para más información sobre esta temática se puede consultar en: https://cordova.apache.org/docs/es/latest/guide/platforms/android (Último Acceso: 19/12/2016)

PhoneGap maneja APIs (plugins) que permiten tener acceso a elementos internos del dispositivo móvil, como el acelerómetro, cámara, contactos en el dispositivo, red, almacenamiento, notificaciones, etc. Para el prototipo desarrollado se utilizó el plugin *BarcodeScanner*[10] el cual se detalla en el Anexo A. Este plugin permite la lectura de los códigos QR desde el prototipo. Luego, se entrará en detalle de cómo se usa el mismo dentro del prototipo.

A continuación se describen los pasos que fueron realizados para llevar a cabo el prototipo:

1. Configuración del archivo config.xml, por ejemplo, definiendo el nombre de la aplicación, plataforma (*Android*), y además que se va usar el *BarcodeScanner*. En el Anexo B se puede apreciar la especificación realizada.
2. Tomando como base el modelo propuesto en el Capítulo 3, las mismas fueron pasadas a *JavaScript* para poder usarlas dentro de *Phonegap*. Se instanciaron los siguientes elementos para este prototipo (por simplicidad no se usó una base de datos).

   Los *Sectores* que se definieron son (cabe destacar que como son sectores cuadrados con dos puntos se puede determinar el área de los mismos):

   - Elemento *sector1* con
     código: 201
     nombre: *Almacén*
     posición: *areaSector1* con puntos:
     point1: (x: 20, y: 2)
     point2: (x: 120, y: 80)
   - Elemento *sector2* con
     código: 202
     nombre: *Bazar*
     posición: *areaSector2* con puntos:
     point1: (x: 20, y: 100)
     point2: (x: 120, y: 130)
   - Elemento *sector3* con
     código: 203
     nombre: *Electrónica*
     posición: *areaSector3* con puntos:
     point1: (x: 150, y: 2)
     point2: (x: 270, y: 30)
   - Elemento *sector4* con
     código: 204
     nombre: *Perfumería*
     posición: *areaSector4* con puntos:
     point1: (x: 150, y: 50)
     point2: (x: 210, y: 80)

[10] *Barcode Scanner*: https://github.com/phonegap/phonegap-plugin-barcodescanner (Último Acceso: 12/12/2016)

- Elemento *sector5* con
  código: 205
  nombre: *Limpieza*
  posición: *areaSector5* con puntos:
  point1: (x: 150, y: 100)
  point2: (x: 210, y: 130)
- Elemento *sector6* con
  código: 206
  nombre: *Bebidas*
  posición: *areaSector6* con puntos:
  point1: (x: 240, y: 50)
  point2: (x: 270, y: 130)

En la Figura 4.1 se pueden apreciar los distintos sectores mencionados anteriormente y sus posiciones dentro del supermercado usado para el prototipo. Se puede observar que el *sector1* tiene línea punteada en su contorno, esto se debe a que por ahí hay un pasillo.

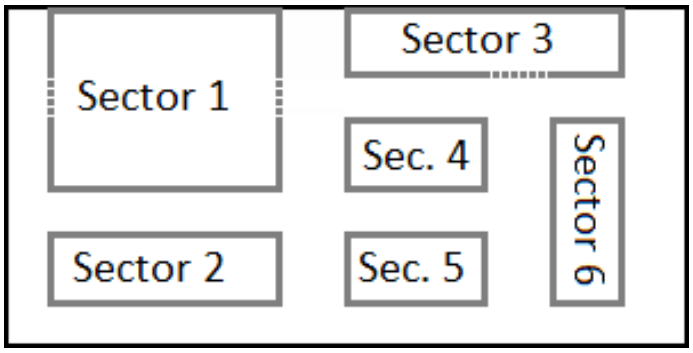


Figura 4.1: Sectores del *Supermercado*

Se definieron tres góndolas, las cuales están dos dentro del *sector1* y otra en el *sector6.* Los detalles de las mismas se describen a continuación, donde para cada una de las góndolas se han creado diferentes categorías. Cabe mencionar que como son góndolas cuadradas con dos puntos se puede determinar el área de las mismas.

- Elemento *gondola1* con
  código:101
  categoría: *categoria1*
  con nombre: *Golosinas y Chocolates*
  posición: *areaGondola1* con puntos:
  point1: (x: 20, y: 2)
  point2: (x: 120, y: 30)
- Elemento *gondola2* con
  código:102
  categoría: *categoria2*
  con nombre: *Pastas secas, Arroz y Legumbres*
  posición: *areaGondola2* con puntos:
  point1: (x: 20, y: 50)
  point2: (x: 120, y: 80)

- Elemento *gondola3* con
  código:103
  categoría: *categoria3*
  con nombre: *Bebidas Isotónicas*
  posición: *areaGondola3* con puntos:
  point1: (x: 240, y: 50)
  point2: (x: 270, y: 130)

En la Figura 4.2 se pueden apreciar las distintas góndolas mencionadas anteriormente, y sus posiciones. Dentro del *sector1* están las góndolas *gondola1* y *gondola2* mientras que en el *sector6* está la *gondola3*.

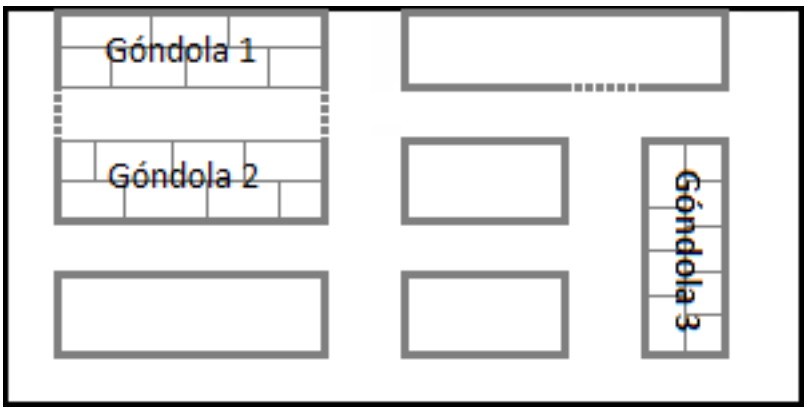


Figura 4.2: Góndolas definidas para el *Supermercado*

Los elementos definidos para este supermercado se listan a continuación, estos son representativos a fines de poder probar el prototipo y la movilidad del usuario dentro del supermercado. Cada uno de los elementos se define con un punto que toma valores (x,y,z), donde el z sirve para poder determinar la altura del mismo cuando hay que brindar una vista frontal.

- El elemento *producto1* con
  código: 1
  nombreCompleto: *Chocolate Cofler Block con Maní Tableta por 38gr.*
  precio*:* 16.80
  posición: (x: 30,y: 25, z: 15)
- El elemento *producto2* con
  código: 2
  nombreCompleto: *Alfajores Havanna Mixtos Con Cobertura De Chocolate y Merengue Caja 6u.*
  precio: 130.00
  posición: (x: 80, y: 25 , z: 45)
- El elemento *producto3* con
  código: 3
  nombreCompleto: *Fideos Fettuccini Don Vicente x 500gr.*
  precio: 21.29
  posición: (x: 60, y:75 , z: 3)

- El elemento *producto4* con
  código: 4
  nombreCompleto: *Arroz Gallo Oro Autointegrado x 1kg.*
  precio: 39.99
  posición: (x: 105, y: 60, z: 3)
- El elemento *producto5* con
  código: 5
  nombreCompleto: *Gatorade Naranja x 500ml.*
  precio: 16.57
  posición: (x: 265, y: 115, z: 3)

En la Figura 4.3 se pueden apreciar los distintos elementos mencionados y sus posiciones dentro de las góndolas definidas. Dentro de *gondola1* están los productos *producto1* y *producto2*, en la *gondola2* están los productos *producto3* y *producto4* y en la *gondola3* está el *producto5*.

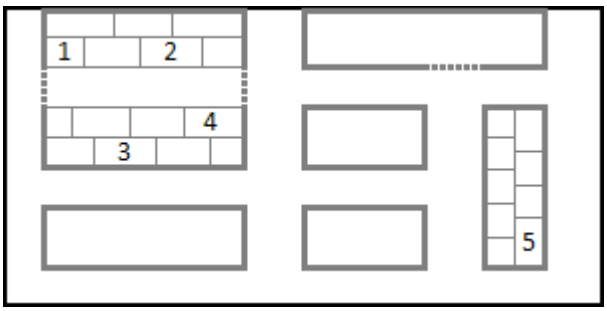


Figura 4.3: Productos definidos para el *Supermercado*

3. Se definió una única lista de compras. Denominada *lista1* que tiene los siguiente datos:

   nombre: *Lista 1 - default*

   elementos: *producto1, producto2, gondola2, sector6*

   Se puede observar que la lista tiene dos productos, una góndola y un sector, dependerá de la estrategia de recorrido que elija hacer el usuario el orden en que estos se compren, esto se explicará más adelante.

4. Se crearon los HTMLs reflejando las pantallas especificadas en la Sección 3.1, luego se brindarán más detalles de las mismas. Solo se implementó el funcionamiento de aquellas relevantes para el posicionamiento del usuario, el resto se pueden visualizar pero los botones correspondientes no tienen funcionalidad en sí.

5. Para la generación dinámica de los recorridos utilizados al dar comienzo una lista de compras, el sistema determina el criterio de ordenación según la preferencia seleccionada y calcula el camino mínimo según dicha estrategia. El cálculo del camino mínimo se realizó con el algoritmo de *Dijkstra*[11] a partir de un código existente en *JavaScript*[12]. Para calcular el camino mínimo se definieron aquellos puntos relevantes del mapa del supermercado. En la Figura 4.4 se pueden apreciar los puntos considerados, para los

[11] *Dijkstra* es un algoritmo para la determinación del camino más corto dado un vértice origen al resto de los vértices en un grafo con pesos en cada arista.
[12] Código extraído del sitio: https://github.com/nojacko/dijkstras-js

mismos, se crearon conexiones y estos son los que se usan para que el *Dijkstra* busque el camino.

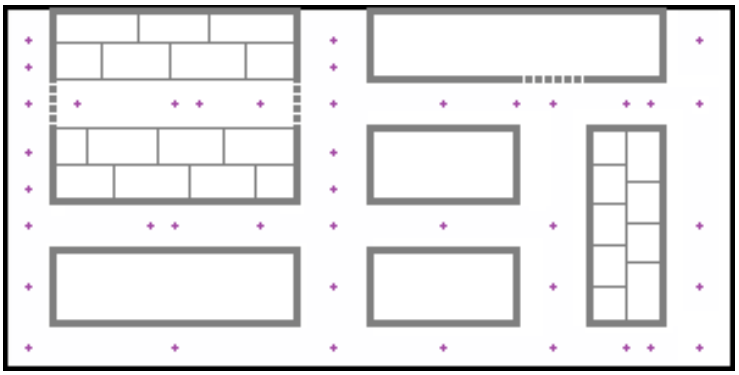


Figura 4.4: Puntos relevantes usados para calcular caminos con *Dijkstra*

Dado que los puntos destacados en la Figura 4.4 no coinciden con las posiciones de los productos, góndolas y sectores se hace un cálculo para determinar cuál es el punto destacable más cercano para poder a partir de allí realizar el cálculo del camino. Lo mismo pasa con la posición del usuario (la cual se determina a partir que el mismo lee un código QR de un producto, góndola o sector). Más adelante se mostrarán cómo el prototipo muestra los caminos calculados.

6. Se define la invocación al *BarcodeScanner* dentro del prototipo acorde a los especificado en el Anexo A. En el Código 4.1 se puede apreciar un pseudocódigo para explicar cómo se hace la invocación al *BarcodeScanner*. Se puede apreciar que se deben enviar dos funciones, una que está relacionada a la lectura correcta del código (en este caso se debe analizar si el usuario está mirando o comprando, y en base a esto actualizarle la vista y su estado) y otra función corresponde a la lectura errónea. Y además, se pasan algunos parámetros de configuración. El código completo de esta invocación realizada en el prototipo se puede observar en el Anexo C.

```
cordova.plugins.barcodeScanner.scan(
    function (result){
                        // Se analiza si el usuario está mirando o comprando
                        // y acorde a esto se actualiza su vista y estado

    },
    function (error){
                         navigator.notification.alert("Scanning failed: "
                         + error, null, "Error", "Aceptar");
    },
    {
      "orientation": "portrait",
      "prompt" : "Coloque un código QR en el interior del rectángulo del visor",
      "formats" : "QR_CODE"
    }
   );
```

Código 4.1: Pseudocódigo de la invocación del *BarcodeScanner*

7. Decidimos crear un código QR representando la entrada del supermercado, y utilizarlo para dar inicio a la compra. Este código (000) se puede apreciar en la Figura 4.5.

Figura 4.5: Código 000

Los Códigos QR definidos para los *Sectores* se pueden apreciar en la Figura 4.6. Los códigos coinciden con la definición de cada una de los sectores mencionados anteriormente.

Figura 4.6: Códigos de Sectores. **(a)** Código 201 **(b)** Código 202 **(c)** Código 203 **(d)** Código 204 **(e)** Código 205 **(f)** Código 206

Los Códigos QR definidos para los *Góndolas* se especifican en la Figura 4.7, estos coinciden con la información definida para cada una de las góndolas instanciadas en el modelo.

Figura 4.7: Códigos de Góndolas. **(a)** Código 101 **(b)** Código 102 **(c)** Código 103

Los Códigos QR creados para los elementos se pueden apreciar en la Figura 4.8.

Figura 4.8: Códigos de Productos. **(a)** Código 001 **(b)** Código 002 **(c)** Código 003 **(d)** Código 004 **(e)** Código 005

A continuación se brindan algunas pantallas principales del prototipo, las mismas fueron tomadas de un *Smartphone Samsung Galaxy J7 Número de Modelo SM-J700M con Versión de Android 6.0.1*.

- *Login*

  Una vez instalada la aplicación en nuestro dispositivo, haremos uso de la misma haciendo click en el acceso directo denominado "*Supermercado Inteligente*" creado en el momento de la instalación (ver Figura 4.9).

Figura 4.9: Acceso Directo Supermercado Inteligente

Al abrir la aplicación tenemos la ventana de *Login* (como se muestra en la Figura 4.10), donde aparecen como campos obligatorios *Usuario* y *Contraseña*.

Figura 4.10: Pantalla de Login

Para el caso de intentar iniciar sesión obviando alguno de los campos requeridos en la Figura 4.10, el sistema mostrará el mensaje de error "*Completar usuario*" (como se muestra en la Figura 4.11.a) y/o "*Completar contraseña*." (como se observa en Figura 4.11.b), según corresponda.

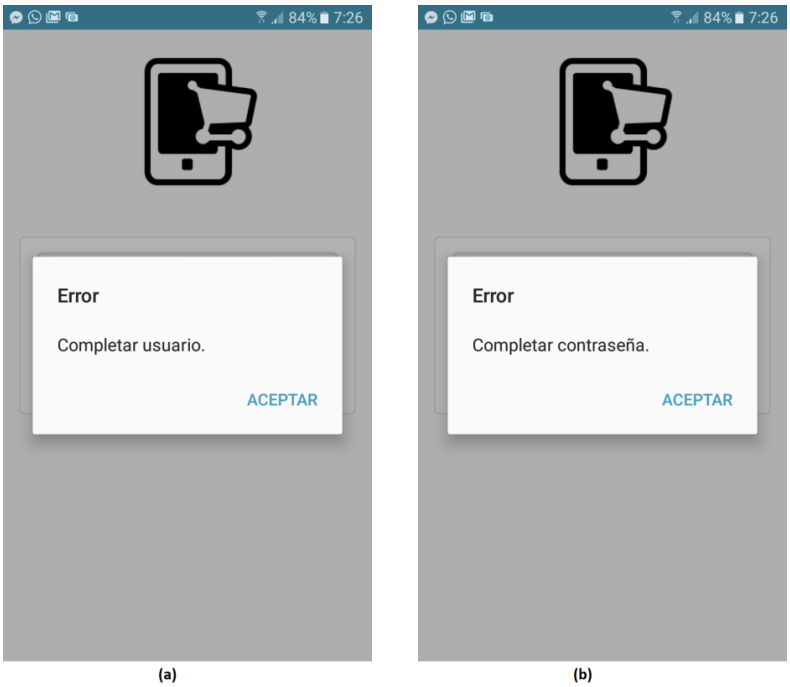


Figura 4.11: (a) Error indicando que falta *Completar Usuario* **(b)** Error indicando que falta *Completar Contraseña*

Caso contrario, si el usuario y la contraseña son incorrectos, el sistema visualizará el siguiente mensaje de error: "*Iniciar sesión con cliente1 (pass 1234)*" que son los datos con los que fue instanciado el prototipo como se muestra en la Figura 4.12. Cabe aclarar que el prototipo está acotado a un único *Supermercado* y un único *Usuario*.

Figura 4.12: Error en los datos de login, se muestra al mensaje "*Iniciar sesión con cliente1 (pass 1234)*"

El botón *Registrarse* (mostrado en la Figura 4.10), nos envía a un formulario de registración como se muestra en la Figura 4.13, para poder completar *Nombre de Usuario, Email, Contraseña, Repetir Contraseña*. Sin embargo, esta pantalla es solo visual, no tiene comportamiento alguno, dado que se decidió tener un usuario precargado para el prototipo y el mismo es único. Esto podría ser un aspecto a extender del prototipo desarrollado.

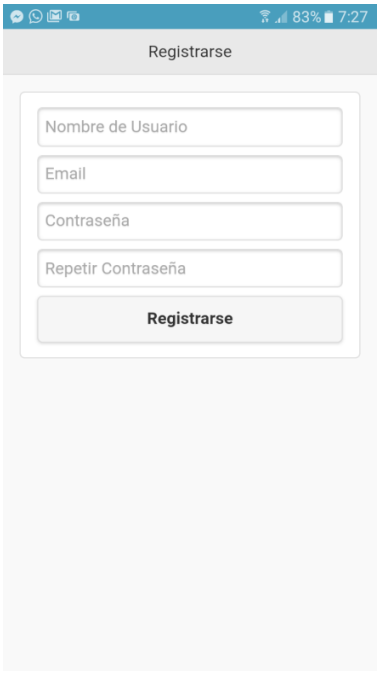


Figura 4.13: Pantalla de Registrarse

- *Pantalla de Inicio*

  Una vez logueado, el usuario accede a la pantalla de Inicio (como se muestra en la Figura 4.14), donde se tendrá acceso a todas las funcionalidades del prototipo.

Figura 4.14: Pantalla Inicio del Prototipo

- *Menú Desplegable*

  En la esquina superior izquierda de todas las pantallas aparece el botón del menú desplegable (como se puede apreciar en la Figura 4.14), que es accesible en cualquier momento. Al seleccionarlo se muestra el listado de acciones que se puede observar en la Figura 4.15.

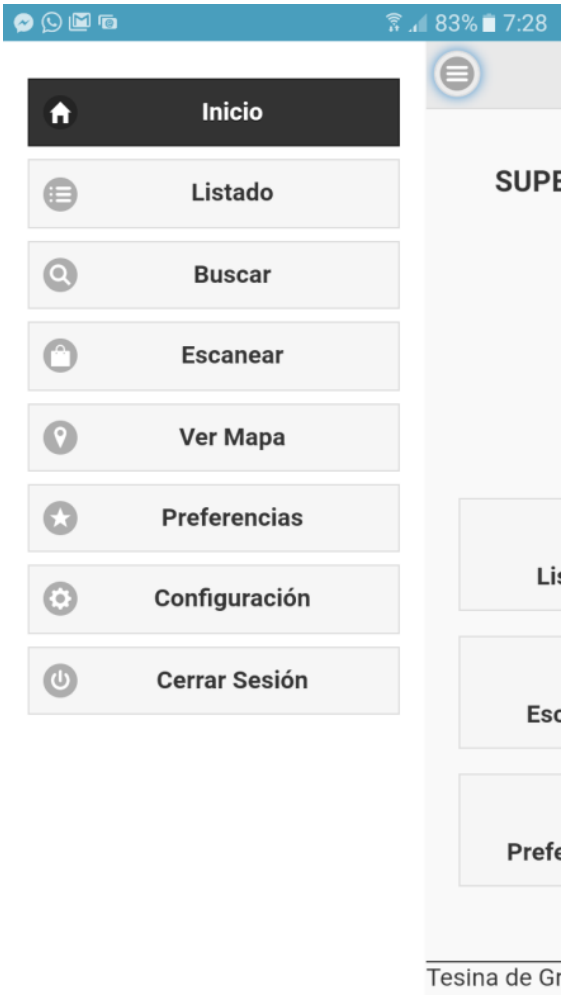


Figura 4.15: Menú Desplegable

- *Listado*

  El botón *Listado* (de la Figura 4.14 o 4.15) nos redirecciona a la vista *Listado* que se puede observar en la Figura 4.16. En esta opción se visualizarán las listas de compras creadas por el usuario y un botón *Crear Lista* (especificado solo a modo visual sin haber implementado su funcionalidad, ya que las listas de compras fueron precargadas acorde a los elementos creados del *Supermercado* y a fines de poder probar la movilidad del usuario conteniendo en la lista los distintos elementos definidos).

Figura 4.16: Ejemplo de Listado

Al hacer click sobre una de las listas precargadas (de la Figura 4.16), se abrirá una ventana tipo popup con diferentes acciones posibles: *Comenzar Compra*, *Editar Lista*, *Eliminar Lista* (esta última no tiene implementada funcionalidad, dado que luego no se pueden crear desde el prototipo nuevas listas). Estas funciones se pueden apreciar en la Figura 4.17.

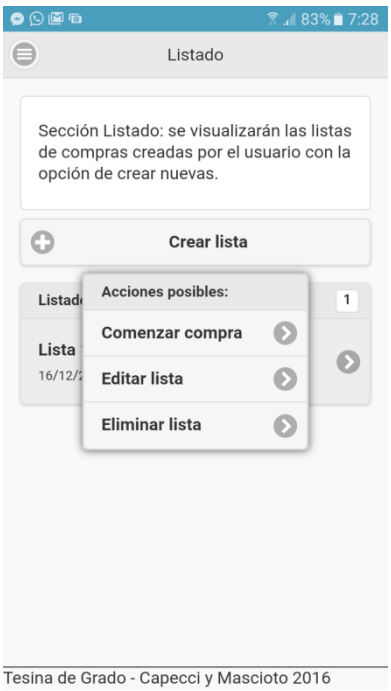


Figura 4.17: Listado Acciones Posibles a una lista de compras

Supongamos que se elige la opción "*Comenzar Compra*" (de la Figura 4.17) como resultado se puede ver la Figura 4.18. Se puede apreciar que como aún la aplicación no cuenta con una posición del usuario dentro del *Supermercado* se le pide que escanee un código para poder determinar a partir del mismo, dicha posición. Si es la primera vez que

se intenta escanear un código, entonces el sistema preguntará al usuario si permite que la aplicación utilice la cámara del dispositivo para realizar el escaneo.

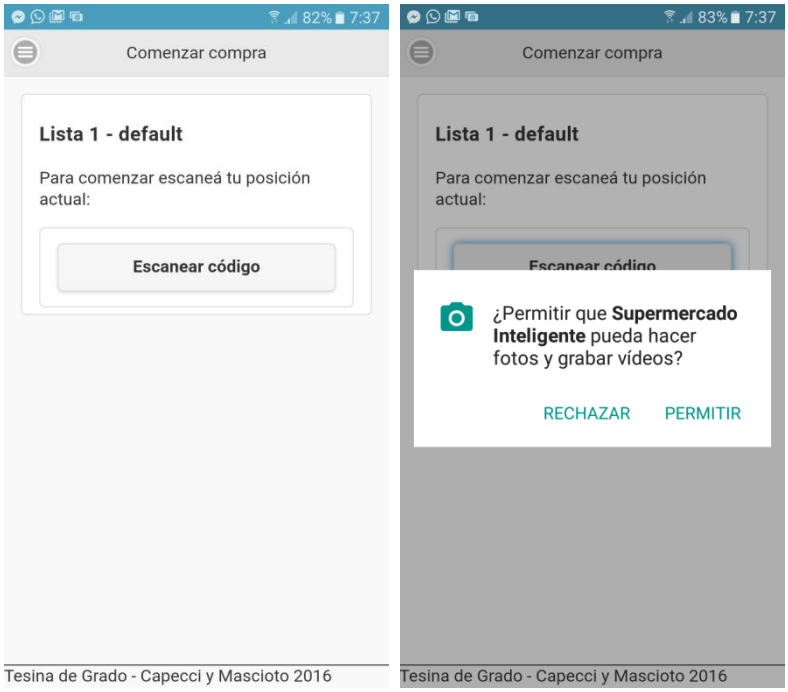


Figura 4.18: Pantalla de *Comenzar Compra*

El usuario puede, por ejemplo, escanear el código 000, para posicionarse en la puerta del supermercado, como se puede apreciar en la Figura 4.19.

Figura 4.19: Usuario posicionado en la *Entrada del Supermercado*

Una vez que el usuario escanea su posición actual el sistema calcula el recorrido de la compra según la preferencia elegida. Muestra el mapa con la posición del usuario y el camino al próximo destino de la lista como se puede apreciar en la Figura 4.20.

Figura 4.20: Pantalla de *Compra Actual* mostrando el primer producto a comprar.

En la pantalla de *Compra Actual* decidimos unificar los botones *Escanear/Comprar* respecto del mockup mostrado en la Figura 3.10 (de la Sección 3.1). Esta decisión fue realizada ya que el botón *Escanear* podría permitir comprar un producto, entonces unificamos a nivel de prototipo esta funcionalidad, ya que la interpretación de cómo reacciona depende de si el usuario está realizando una compra o no. Si escanea y está realizando una compra, se puede interpretar que quiere comprar ese producto leído.

Los botones *Anterior* y *Siguiente* (de la Figura 4.20) permiten navegar en la lista de compras al elemento previo al actual siempre y cuando no sea el primero, para este caso el botón estará deshabilitado, o al próximo elemento de la lista según la acción correspondiente seleccionada. *Siguiente* estará deshabilitado cuando el elemento destino sea el último elemento de la lista ordenada. Esto es una navegación que no implica que el usuario se mueve, sino que puede ver cómo debería ser su recorrido físico.

Cuando se accede a la funcionalidad *Vista* (de la Figura 4.20) se comporta de la siguiente manera:

- Si el destino es un *Sector*, por defecto se muestra la vista *Sector* (ver Figura 4.21), y se deshabilitan las otras opciones de vista.

Figura 4.21: Pantalla de *Compra Actual – Vista Sector*

- Si el destino es una *Góndola*, por defecto se muestra la vista *Góndola* (ver Figura 4.22) con opción de cambiar a la vista *Sector*.

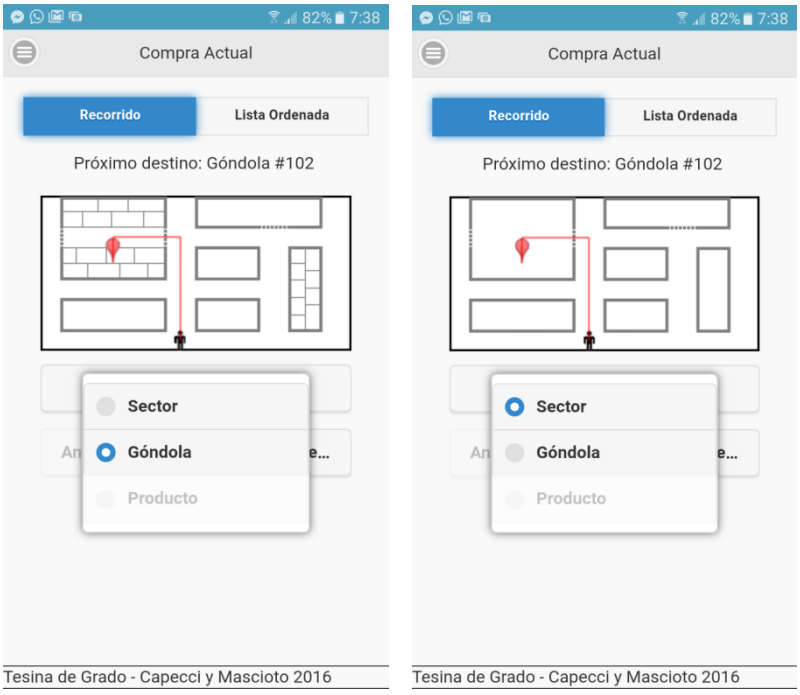


Figura 4.22: Pantalla de *Compra Actual – Vista Góndola y Sector*

- Si el destino es un *Producto*, por defecto se muestra la vista *Góndola*, con opción de cambiar a las vistas *Sector* y *Producto*. No pusimos por defecto a la vista *Producto* porque en dicha vista no mostramos la posición del usuario, es una vista cuyo objetivo es mostrar con el máximo nivel de detalle la ubicación del producto

dentro de su góndola, y se asume que el usuario se encuentra justo en frente de dicha góndola. La vista Producto se puede apreciar en la Figura 4.23.

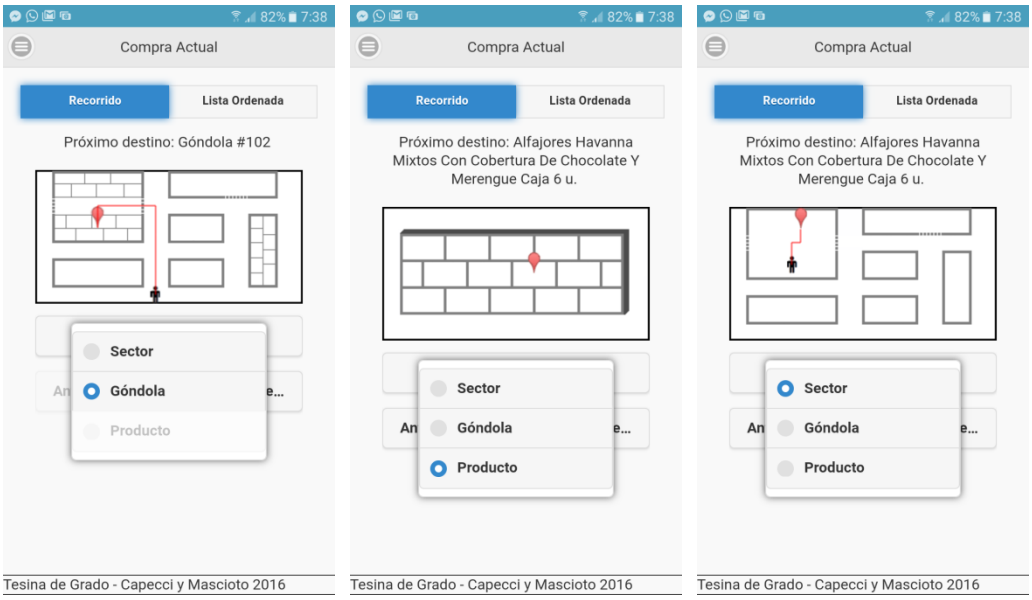


Figura 4.23: Pantalla de *Compra Actual – Vista Góndola, Producto y Sector*

En el caso de elegir la opción "*Editar Lista*" (de la Figura 4.17) el usuario recibe una pantalla como se muestra en la Figura 4.24.

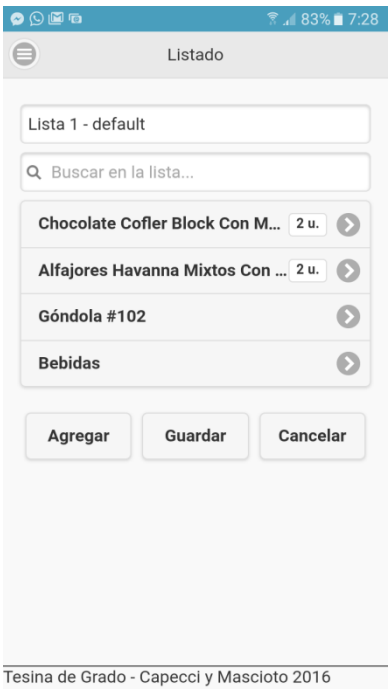


Figura 4.24: Pantalla de *Editar Lista*

- *Buscar*

  El botón *Buscar* (de la Figura 4.14 o 4.15) nos redirecciona a la vista *Buscar*, en esta opción se permite realizar la búsqueda con los siguientes filtros: todo, producto, góndola o sector como se puede apreciar en las Figura 4.25.

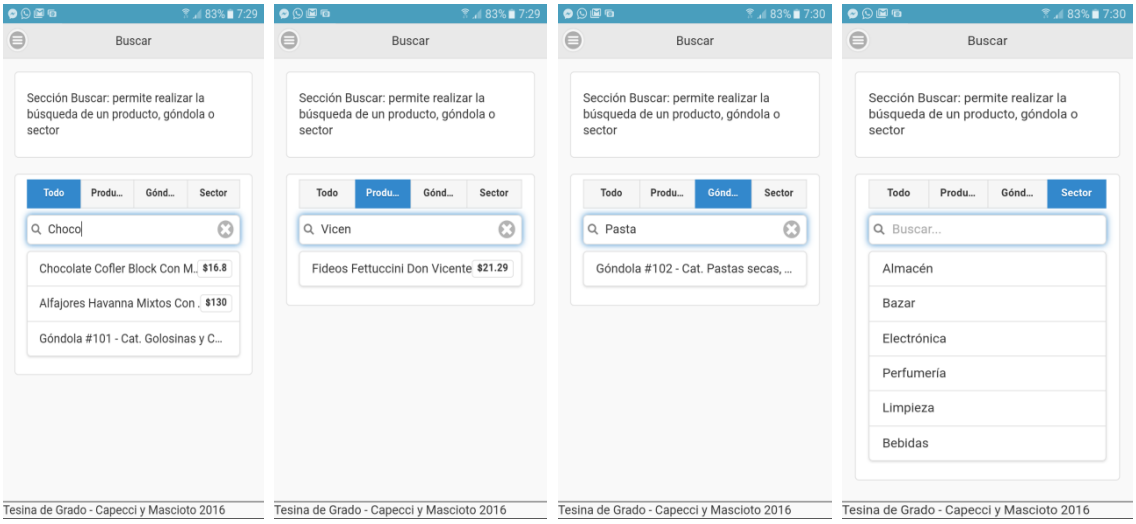


Figura 4.25: Pantalla de Búsquedas usando diferentes filtros

- *Escanear*

El botón *Escanear* (de la Figura 4.14 o 4.15) nos redirecciona a la vista Escanear (Figura 4.26). El sistema permite escanear códigos QR para ver información de los elementos del supermercado y acorde a esto determinar la posición del usuario dentro del establecimiento.

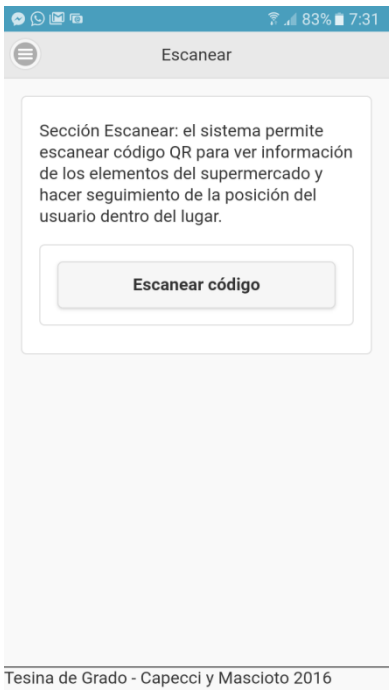


Figura 4.26: Pantalla de *Escanear Códigos*

Al hacer click en el botón *"Escanear código"* (de la Figura 4.26), se habilita la cámara del dispositivo y queda a la espera de identificar un código. Se puede apreciar en la Figura 4.27 un rectángulo que es el sector donde se debe colocar el código QR.

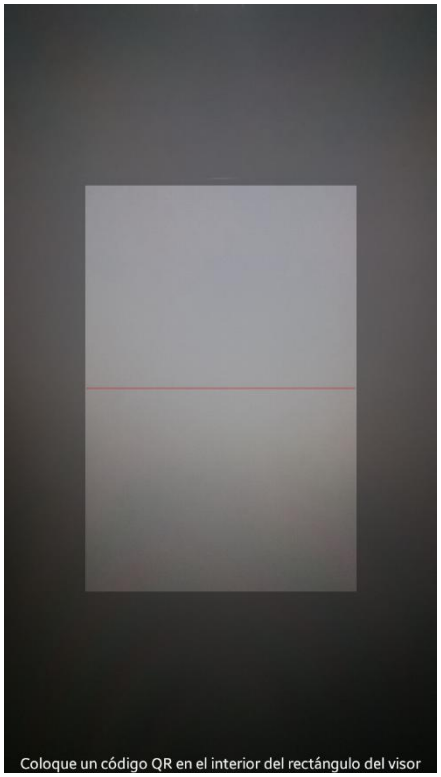


Figura 4.27: Visor Escaner

Si el usuario sale de esta funcionalidad sin haber escaneado nada, el sistema mostrará mensaje de error "*Scanner cancelado*" como se muestra en la Figura 4.28.

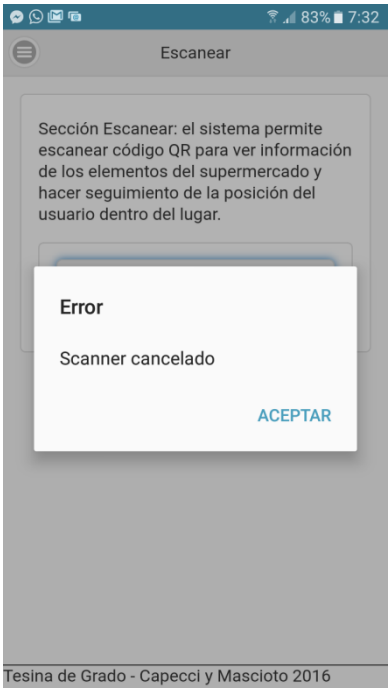


Figura 4.28: Mensaje de *Scanner Cancelado*

Si el usuario intenta escanear un código que no es de tipo QR, la aplicación no lo reconocerá como un código posible a escanear, debido a que dentro de la configuración del plugin se definió que solo se aceptan códigos de tipo QR. Si realiza el escaneo a un código QR inválido el sistema mostrará mensaje de error como se muestra en la Figura 4.29.

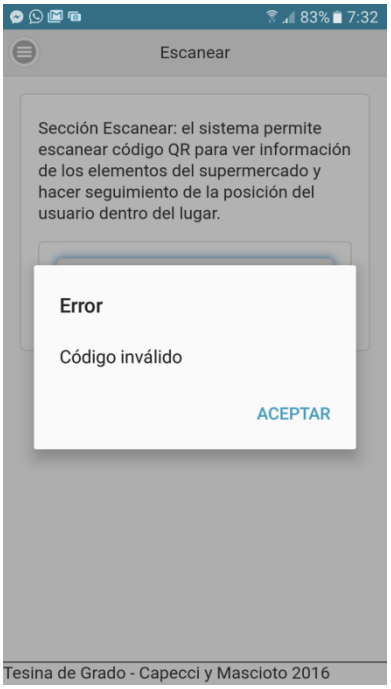


Figura 4.29: Mensaje de *Código inválido*

En el caso de haber escaneado un código de los definidos en el prototipo (código válido), es decir, es un caso exitoso, el comportamiento de esta lectura dependerá del estado actual del usuario. Acorde a esto, se dan dos situaciones:

- Si el estado del usuario es *Mirando* el comportamiento del botón escanear actualiza la posición del cliente y retorna al usuario la información del elemento escaneado (ver Figura 4.30). Se puede ver que esta información varía si el código se corresponde con un producto, una góndola o un sector. Se puede ver en las tres pantallas que se le muestra la posición actual del usuario.

Figura 4.30: Pantalla de *Resultado Escaner* - Estado Mirando

- Si el estado del usuario es *Comprando* el comportamiento del botón escanear desencadena varias situaciones que se describen a continuación:
    - Si se escanea el código de un producto:
        - *y el mismo es el destino del usuario*, es decir, es el producto que hay que comprar, se actualiza la lista de compras y se le informa al usuario el próximo elemento de la lista junto con el mapa actualizado.
        - *y el mismo NO es el destino del usuario*, el sistema actualiza la posición del usuario y le sugiere comprar el producto o seguir el orden de la lista de compra devolviendo un mapa de recorrido actualizado.
    - Si se escanea el código de una góndola:
        - *y esta es el destino del usuario*, se actualiza la lista de compras y se le informa al usuario el próximo elemento de la lista con el mapa actualizado.
        - *y esta NO es el destino del usuario*, con la particularidad que la góndola contiene el producto o sector destino, el sistema devuelve el mapa actualizado.
    - Si se escanea el código de un sector:
        - *y este es el destino del usuario*, se actualiza la lista de compras y se le informa al usuario el próximo elemento de la lista de compras junto con el mapa actualizado.
        - *y este NO es el destino del usuario*, con la particularidad que el sector contiene el producto o góndola destino, el sistema devuelve el mapa actualizado.

  Este funcionamiento se podrá observar más en detalle en el Capítulo 5 con un ejemplo concreto de uso.

- *Mapa*

  El botón *Ver Mapa* (de la Figura 4.14 o 4.15) nos redirecciona a la vista *Ver Mapa*, cuando no se cuenta con una posición actual se muestra una pantalla como se puede observar en la Figura 4.31.

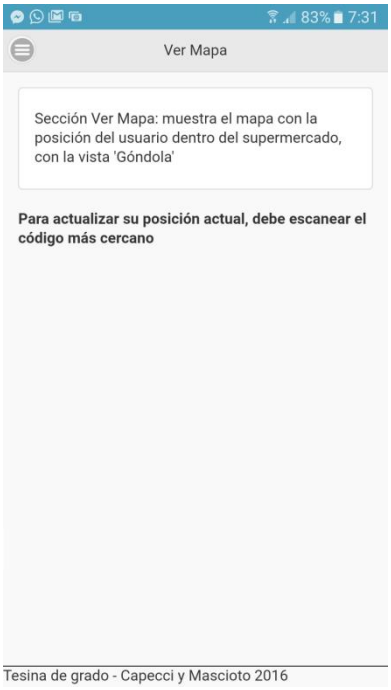


Figura 4.31: Pantalla de *Ver Mapa*

En el caso que la posición del usuario sea un dato conocido por el prototipo, la funcionalidad del *Ver Mapa* superpone el ícono del usuario de acuerdo a la posición escaneada en el mapa de góndolas como se muestra en la Figura 4.32.

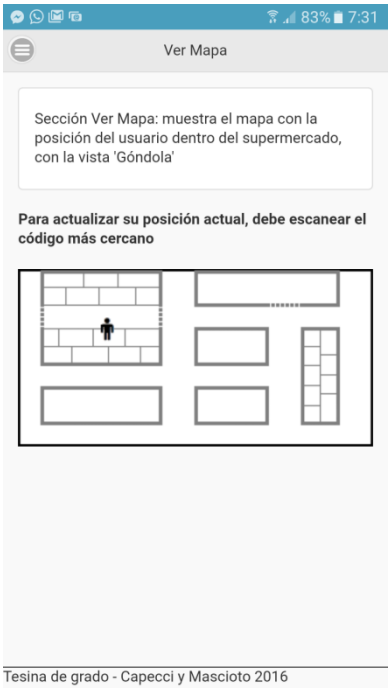


Figura 4.32: Pantalla de *Ver Mapa - Posición del usuario*

- *Preferencias*

  El botón *Preferencias* (de la Figura 4.14 o 4.15) nos redirecciona a la vista *Preferencias* como se muestra en la Figura 4.33. Las opciones disponibles son: *por orden alfabético*, *por cercanía de producto* y *por orden en la lista*. Según la opción elegida es como se van a recorrer los elementos de una lista para comprarlos.

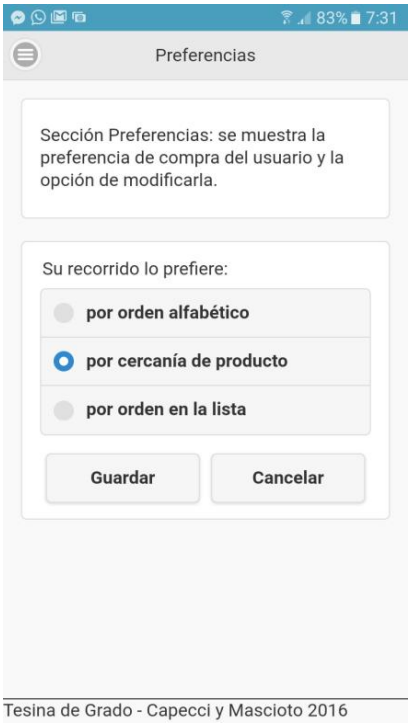


Figura 4.33: Pantalla de *Preferencias*

- *Configuración*

  El botón *Configuración* (de la Figura 4.14 o 4.15) nos redirecciona a la vista que se puede observar en la Figura 4.34, la misma decidimos dejarla como *template* visual pero no nos enfocamos en implementar esta funcionalidad dado que nuestro foco estuvo puesto en la movilidad del usuario.

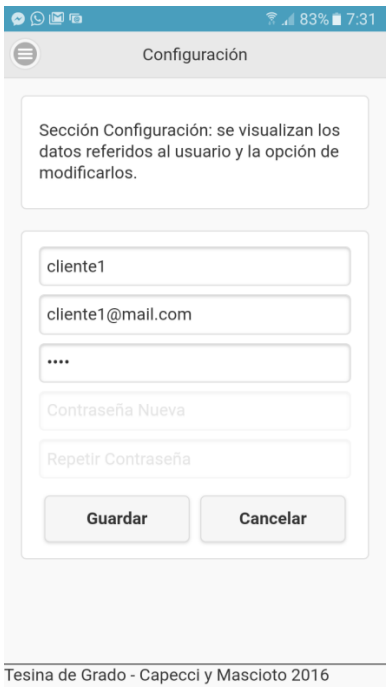


Figura 4.34: Pantalla de *Configuración*

- *Cerrar Sesión*

  Desde el Menú Desplegable podemos acceder a la acción *Cerrar Sesión* (como se mostró en la Figura 4.15) donde se pedirá confirmación a través de un popup como se muestra en la Figura 4.35.

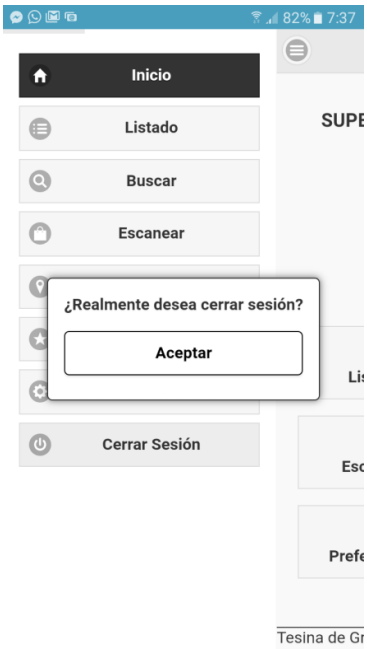


Figura 4.35: Pantalla de *Cerrar Sesión*

5. EJEMPLO DE USO DEL PROTOTIPO

En este capítulo se describe un posible caso de uso del prototipo propuesto, donde se muestra cómo el usuario se puede ir moviendo por el supermercado para así ir realizando sus compras.

Cabe mencionar que el prototipo fue probado con los siguientes smartphones con versión de *Android* 6.0.1:

- Samsung Galaxy J5 Número de Modelo SM-J500M, usado para pruebas durante el desarrollo del prototipo
- Samsung Galaxy J7 Número de Modelo SM-J700M, usado para capturar las pantallas que se muestran en este documento,
- Samsung Galaxy S6 EDGE Número de Modelo SM-G925l, usado para pruebas finales.

A continuación se muestra el funcionamiento del prototipo según cómo estén seteadas las preferencias de compra.

➢ *Recorrido por cercanía de producto*

La primera simulación que se va a mostrar toma como criterio para la generación del recorrido la preferencia de compra *"por cercanía de producto"*. Abrimos la aplicación, iniciamos sesión ingresando usuario y contraseña (como se mostró en el Capítulo 4). Una vez logueados ingresamos a la pantalla de *Preferencias* (como fue mostrado en la Figura 4.33), para verificar que la preferencia seleccionada sea *"por cercanía de producto"*. En caso que la preferencia tenga un valor diferente procedemos a seleccionar la opción indicada y presionamos en el botón *Guardar*. Esta selección debería quedar como se muestra en la Figura 5.1.

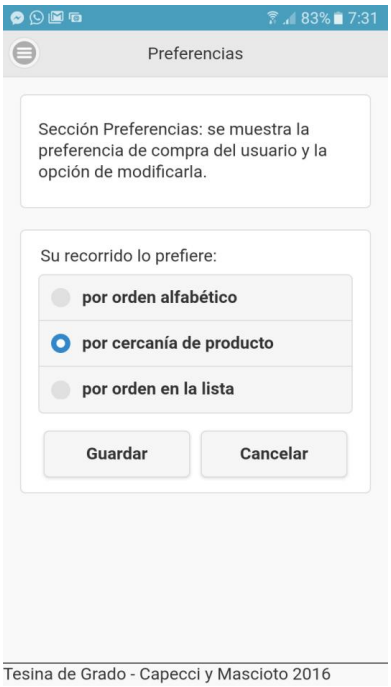


Figura 5.1: Preferencia por cercanía de producto

Una vez configurado el criterio para la generación del recorrido, se ingresa a la pantalla *Listado*, se debe hacer click sobre la lista "*Lista 1 – default"* y seleccionar la acción posible *"Comenzar Compra"*. Esto se puede observar en la Figura 5.2.

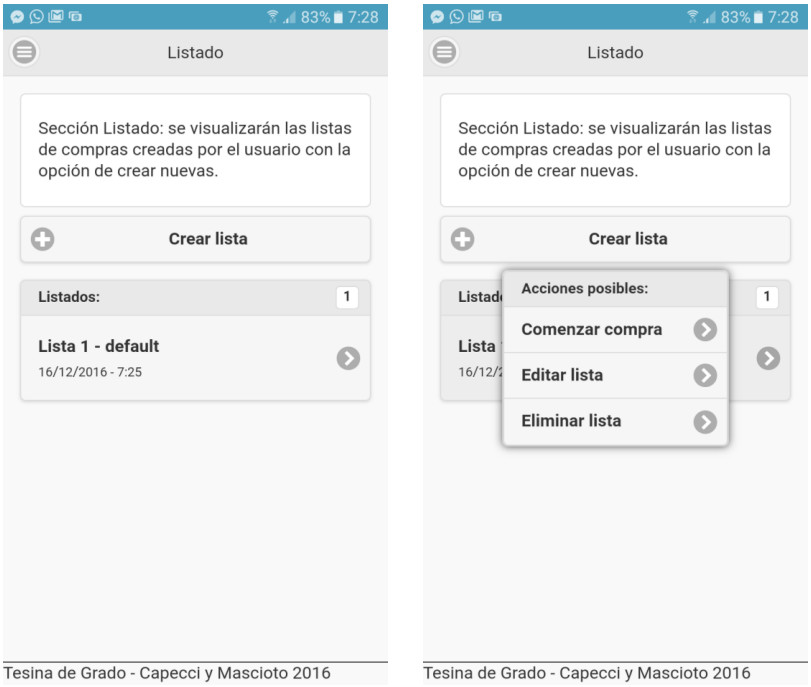


Figura 5.2: Lista 1 – Default. Acción: *Comenzar Compra*

Al seleccionar la opción *"Comenzar compra"* (de la Figura 5.2), el prototipo muestra la pantalla de la Figura 5.3. Se puede apreciar que el sistema nos indica que necesita la posición actual para ubicarnos dentro del supermercado.

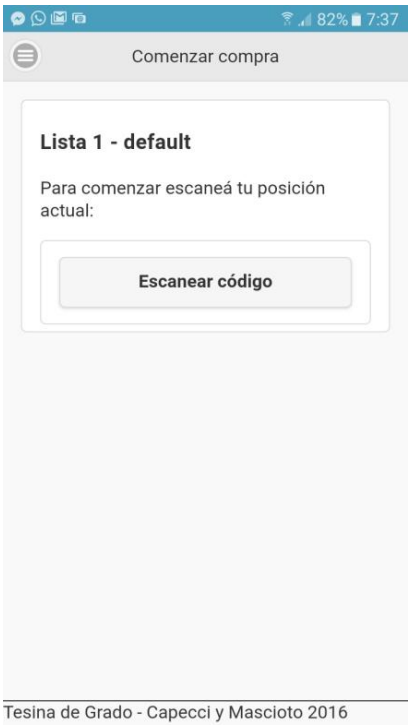


Figura 5.3: Comenzar Compra

Apretamos el botón *"Escanear código"*, y se abrirá la cámara integrada de nuestro dispositivo móvil (como se mostró en la Figura 4.27), y ya se puede escanear el código QR más cercano. Asumimos que el usuario está en la entrada del supermercado por lo que el código QR que se va a escanear va a ser el código QR "*000*", como se puede apreciar en la Figura 5.4.

Figura 5.4: Lectura del Código QR "000"

El sistema decodifica el código (leído en la Figura 5.4) y detecta la posición del usuario. Como resultado, muestra la pantalla de *Compra Actual* con la pestaña *Recorrido* activa, el mapa actualizado indicando la posición del usuario, la posición del elemento destino y el posible camino entre la posición actual y la del elemento destino (acorde a la lista de compras, en este caso la *Góndola #102*). Esto se puede apreciar en la Figura 5.5.

Figura 5.5: Primer elemento de la Lista de compras

En el caso de cambiar a la pestaña *Lista Ordenada* (de la Figura 5.5), se muestra una pantalla como se puede observar en la Figura 5.6. El usuario puede ver el nombre de la lista de compras, el subtotal que se va a ir actualizando a medida que se avance en la lista (se puede apreciar que por ahora es $0), y el listado con los elementos ordenados según la preferencia de compra seteada, en este caso "*por cercanía de producto"* (como se mostró en la Figura 5.1).

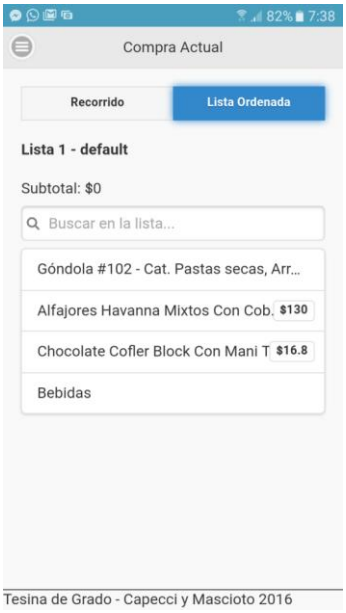


Figura 5.6: Lista Ordenada por Cercanía de Producto

El próximo destino que nos indica el sistema es: *Góndola #102*, cuando el usuario llega a destino aprieta el botón *Escanear* y apunta con la cámara del celular al código QR del elemento, en este caso escanea el código "*102*" como se muestra en la Figura 5.7.

Figura 5.7: Lectura del Código QR "102"

El sistema procesa el código, actualiza el mapa y el próximo destino, siendo ahora el elemento de tipo producto con nombre *Alfajores Havanna Mixtos Con Cobertura De Chocolate Y Merengue Caja 6u.* Como se muestra en la Figura 5.8. Se puede observar

que el usuario está posicionado ahora en la *Góndola #102*, dado que al leer el código el prototipo actualiza su posición. En este caso, la góndola no es algo que se pueda comprar sino que el usuario lo usa para dirigirse a esta parte específica del supermercado.

Figura 5.8: Segundo elemento de la Lista

Se puede ver en la Figura 5.8 que la vista del mapa por defecto es *Góndola*, pero se puede cambiar ingresando a la funcionalidad del botón *Vista*, que particularmente para este tipo de elemento tiene todas las opciones habilitadas (*Sector*, *Góndola* y *Producto*) como se puede apreciar en la Figura 5.9.

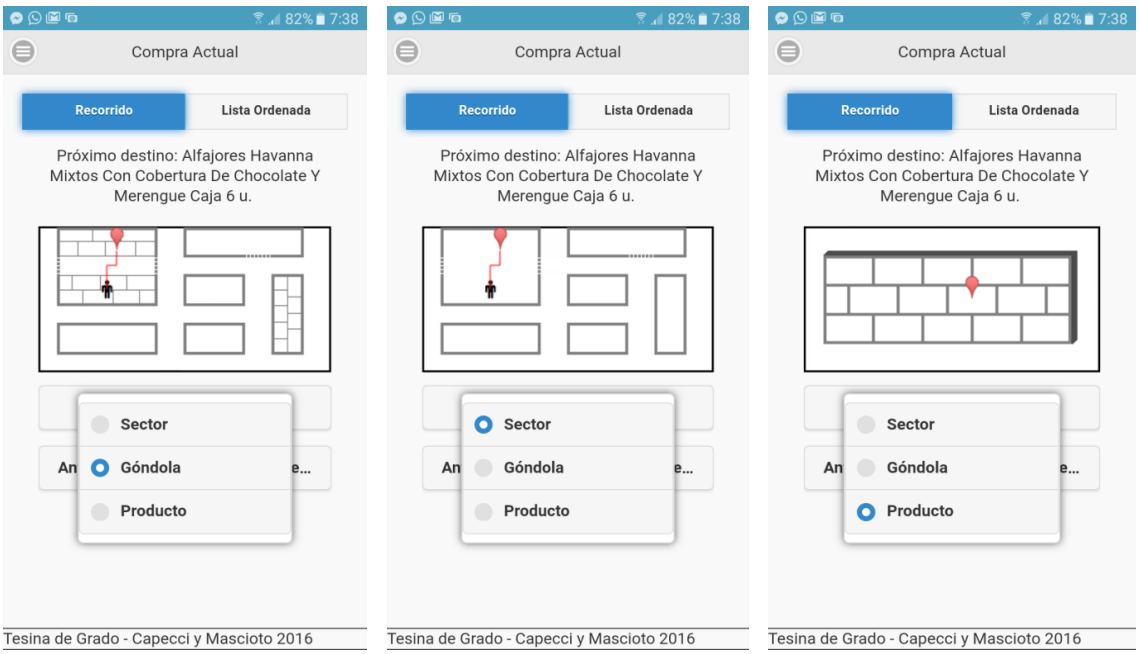


Figura 5.9: Diferentes vistas del recorrido

Cuando el usuario está posicionado en el próximo destino y escanea el código QR “002”, el prototipo calcula el subtotal en la lista ordenada, que pasó de $0 a $130 siendo este

valor el precio del producto recién comprado, como se puede observar en la Figura 5.10. En este caso, la lectura del Código QR al ser el destino del usuario, se interpreta como que se realizó la compra del mismo.

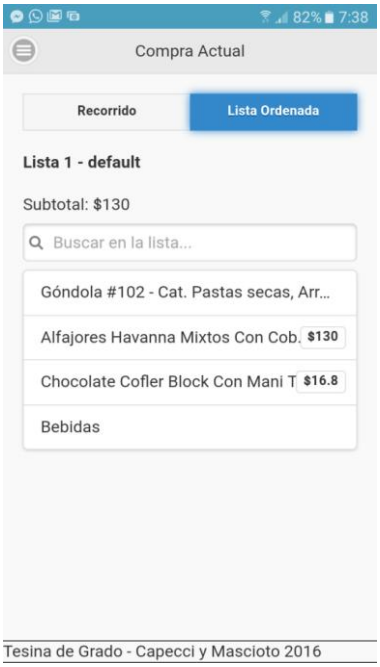


Figura 5.10: Actualización del subtotal de compra a $130

Al instante que el usuario leyó el código QR "002", además de la lista de compras, el prototipo actualiza el mapa y brinda información del siguiente elemento de la lista de compras, que también es de tipo producto con nombre *Chocolate Cofler Block Con Maní Tableta x 38gr.*, como se puede apreciar en la Figura 5.11.

Figura 5.11: Actualización del siguiente destino

De manera análoga al caso anterior, se procesa el elemento destino una vez escaneado su código QR "*001*". En el caso de ver la pestaña *Lista Ordenada*, el subtotal que era de

$130 se le adicionó el precio del último producto escaneado (en este caso $16.80) quedando ahora con un valor de $146.80 como se puede apreciar en la Figura 5.12.

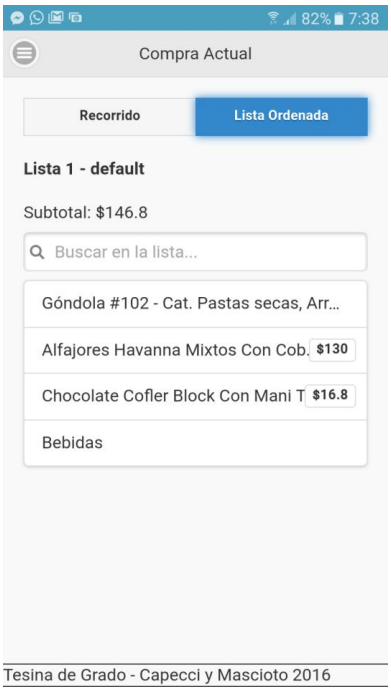


Figura 5.12: Actualización del subtotal de compra a $146.8

Como se mencionó anteriormente, al instante que el usuario leyó el código QR "001", además de la lista de compras, el prototipo actualiza el mapa y brinda información del próximo destino, que es un elemento de tipo *Sector* con nombre *Bebidas* (ver Figura 5.13).

Figura 5.13: Actualización del siguiente destino

Siguiendo las instrucciones de la Figura 5.13, el usuario se dirige al sector, escanea el código QR "206" y finaliza su compra.

En la Figura 5.14 se puede ver que el prototipo notifica al usuario que la compra ha finalizado. Al aceptar el mensaje, se puede apreciar que el mapa muestra la ubicación final del usuario, y se encuentran deshabilitados los botones *"Escanear", "Anterior" y "Siguiente"*. Sólo se podría cambiar la vista, pero en este caso al ser el último elemento de tipo Sector, solo es posible seleccionar dicha vista.

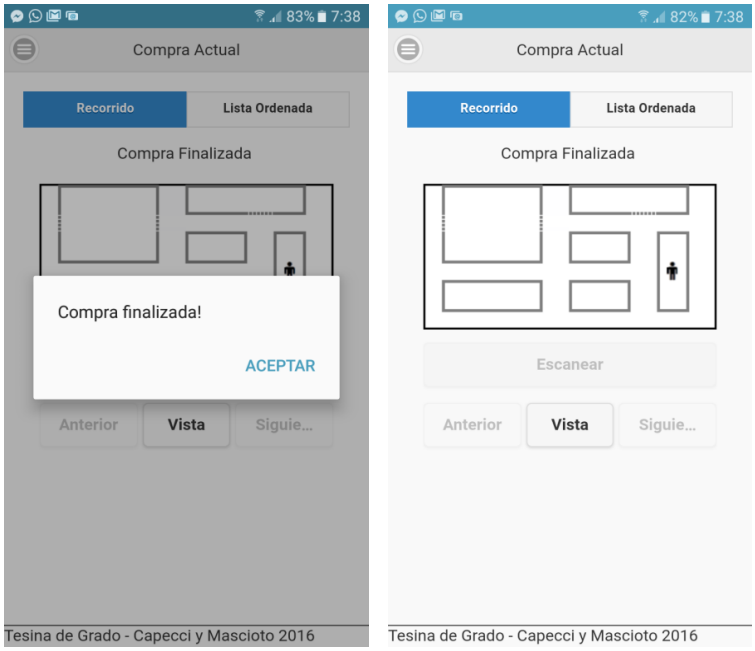


Figura 5.14: Compra finalizada

- *Recorrido por orden alfabético*

En el caso, que en la Figura 5.1 se hubiera seteado la preferencia de compra con la opción *"por orden alfabético"*, la principal diferencia está dada por el orden en que se muestra la lista de compras, como se puede visualizar en la Figura 5.15 (se puede apreciar que la misma varía respecto de la Figura 5.6)

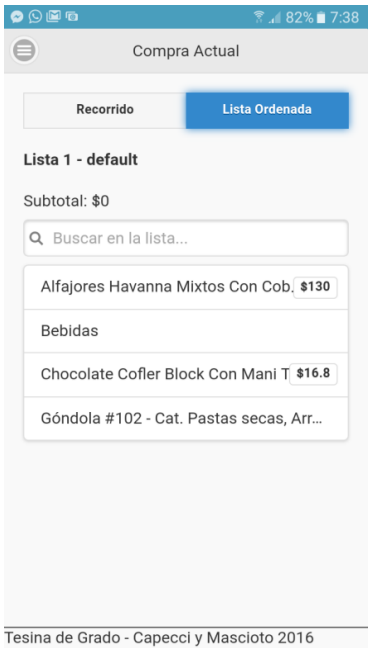


Figura 5.15: Lista Ordenada por *Orden Alfabético*

El orden mostrado en la Figura 5.15 es el que va a usar el prototipo, en este caso, para brindar los diferentes destinos. Supongamos que el usuario va siguiendo las instrucciones del prototipo, y lee los códigos QR de los destinos correctamente. Los mapas que va a ir recibiendo se pueden apreciar en la Figura 5.16. Se puede apreciar cómo desde la Figura 5.16.a a la 5.16.d se muestran los distintos caminos acorde al orden alfabético, luego se le muestra el cartel de finalizó la compra (ver Figura 5.16.e) y por último dónde queda posicionado el usuario, como se puede apreciar en la Figura 5.16.f.

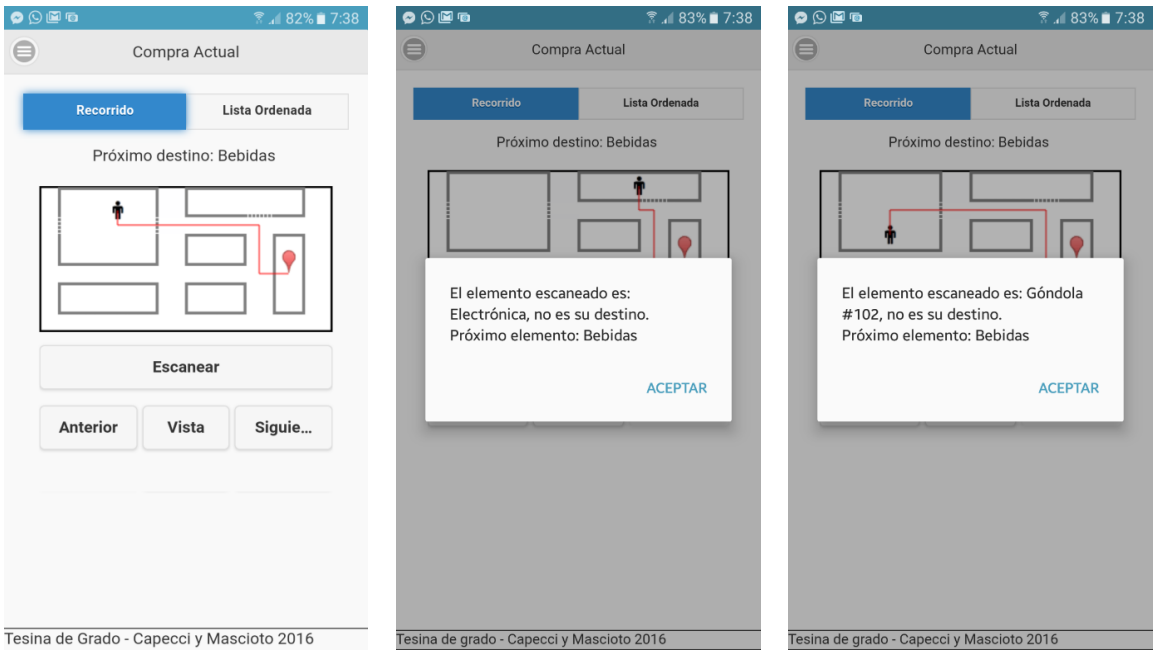
Figura 5.16: Recorridos para el caso de compra por *Orden Alfabético. (a-d) recorrido que se le muestra al usuario. (e) cartel de finalización de compra. (f) posición actual del usuario.*

De esta manera, se pudo apreciar cómo se comporta el prototipo dependiendo de cómo esté seteada la preferencia de compra, brindando acorde a esta el recorrido al usuario.

A continuación se mostrarán distintas situaciones que pueden pasar al usar el prototipo. Estas están relacionadas a la lectura de un código QR que no era el destino del usaurio. Cabe destacar que en las simulaciones de compras realizadas anteriormente, el usuario siempre leía correctamente el código. Los casos que se muestran a continuación son de lecturas de códigos QR que no son el destino, pero son códigos válidos dentro del supermercado, es decir, se le actualiza la posición del usuario.

- *Lectura de un código QR que no guarda relación con el destino*

  Como se puede apreciar en la Figura 5.17.a el próximo elemento destino es el sector *Bebidas*. Si el usuario escanea el código QR de otro sector sin tener relación con su destino, por ejemplo, el sector *Electrónica* como muestra la Figura 5.17.b, el prototipo informa al usuario con una ventana tipo *popup* y actualiza el mapa con la posición del usuario. Se puede apreciar que se le recalculó el camino al destino, acorde a su nueva posición. Lo mismo sucede para el caso que escanee el código QR de una góndola que no está asociada a su elemento destino, como en el caso que se puede apreciar en la Figura 5.17.c, donde se escanea la *Góndola* con código "*102*". Para este caso, se puede apreciar también que el prototipo lo único que realiza es la actualización de la posición del usuario, y como consecuencia le recalcula el camino al destino.



(a) **(b)** **(c)**

Figura 5.17: *(a) El destino es el sector Bebidas. (b) Escanea otro sector. (c) Escanea una góndola.*

Si el usuario escanea un producto que no es el destino, entonces se le brinda la posibilidad de comprarlo o seguir con el recorrido previsto. Este caso se muestra en la Figura 5.18.a, donde el usuario tiene como destino la góndola con código "*102*" pero escanea el producto "*Gatorade Naranja x 500ml.*" como se puede apreciar en la Figura 5.18.b. Si el usuario decide comprarlo, el prototipo procede a agregar dicho producto a la lista de compras y a actualizar el subtotal de la misma. Si la operación se realiza correctamente, se le informa al usuario a través de un mensaje tipo *popup* como se puede observar en la Figura 5.18.c. En la Figura 5.18.d se muestra cómo quedaría la pantalla de *Lista Ordenada* con el producto agregado al final de la lista y el subtotal actualizado. Luego de esto, el usuario puede seguir con su lista de compras, con el orden que tenía pautado.

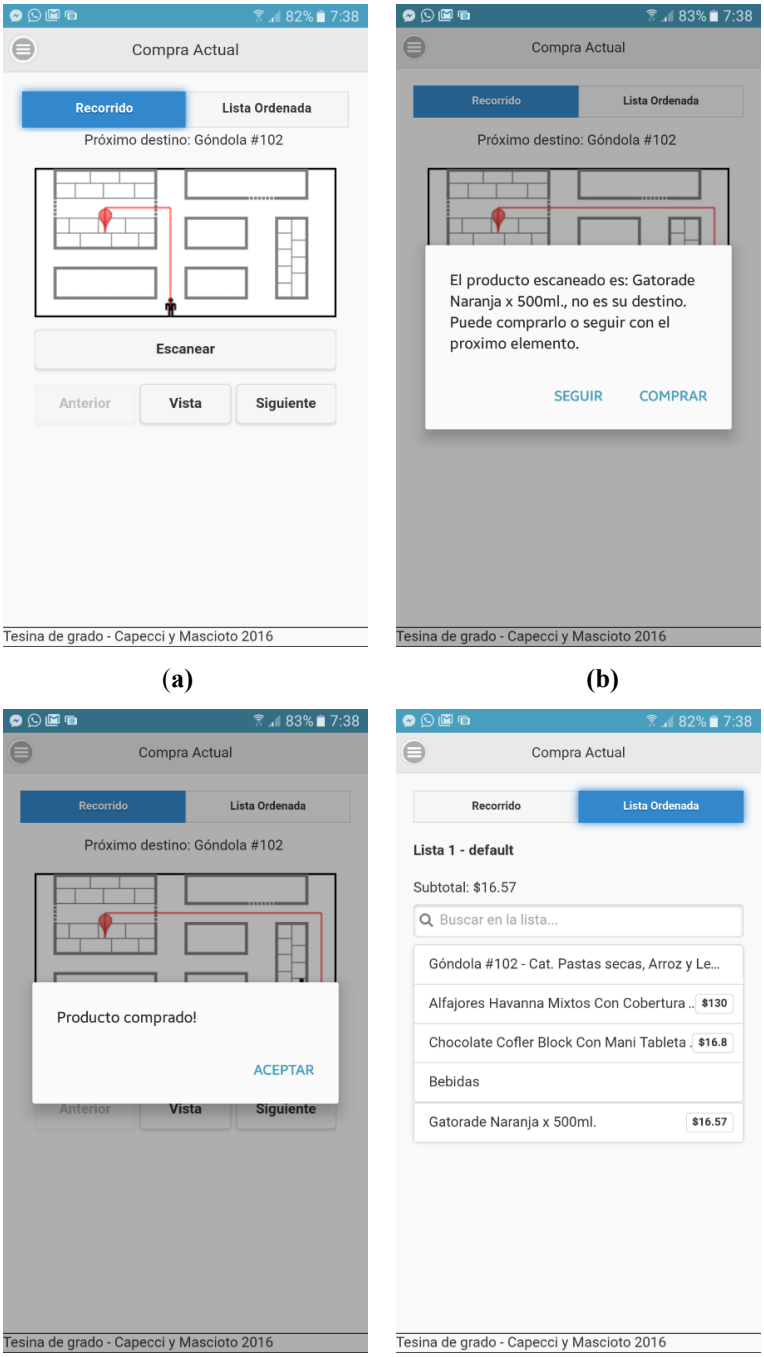


Figura 5.18: *(a) El destino es la góndola con código "102". (b) Escanea otro producto. (c)El usuario decide comprar el producto. (d) Se muestra la Lista ordenada actualizada.*

- *El destino era un producto y leyó el código QR de la Góndola que lo contiene*

  En el caso que el usuario tenga como próximo destino un producto (ver Figura 5.19.a), por ejemplo, *Alfajores Havanna Mixtos Con Cobertura de Chocolate y Merengue Caja 6u.*, pero escanea la *Góndola* con código *"101"*, la cual contiene al producto indicado como destino, el prototipo muestra un mensaje tipo *popup* como se puede observar en la Figura 5.19.b, informando al usuario que se encuentra cerca de su destino. En este caso el usuario podría ingresar a la opción "*Vista*" (ver Figura 5.19.c) y seleccionar la vista

Producto, para obtener la posición más exacta del producto dentro de la góndola, como se muestra en la Figura 5.19.d. Cabe recordar que la vista *Producto* es una vista frontal de la góndola, en este caso, el producto está en el estante del medio de la góndola.

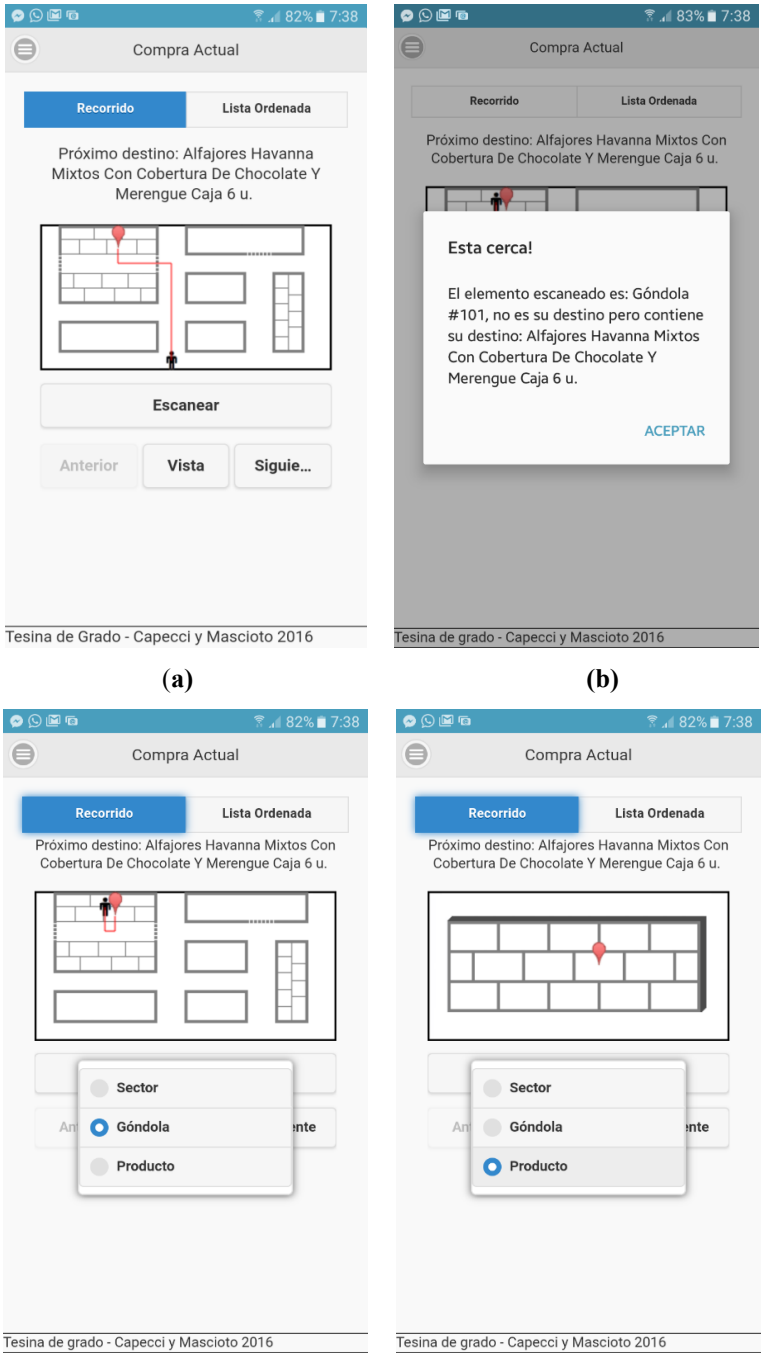
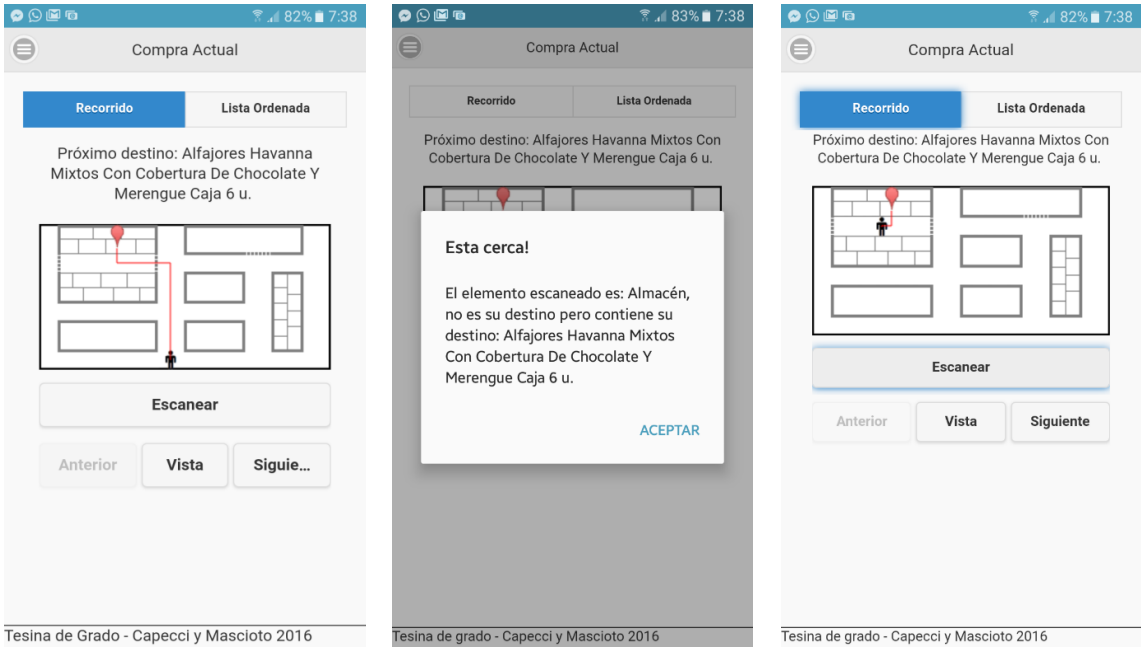


Figura 5.19: *(a)Destino Producto. (b) Contiene destino. (c) Vista Góndola. (d) Vista Producto.*

- *El destino era un producto y leyó el código QR del Sector que lo contiene*

  En el caso que el usuario tenga como próximo destino un producto (ver Figura 5.20.a), por ejemplo, *Alfajores Havanna Mixtos Con Cobertura de Chocolate y Merengue Caja 6u.*, pero escanea el sector *Almacén,* que tiene como particularidad que contiene dicho producto, el prototipo muestra un mensaje como se ve en la Figura 5.20.b y actualizará el mapa con la nueva posición del usuario y el camino al destino (ver Figura 5.20.c).

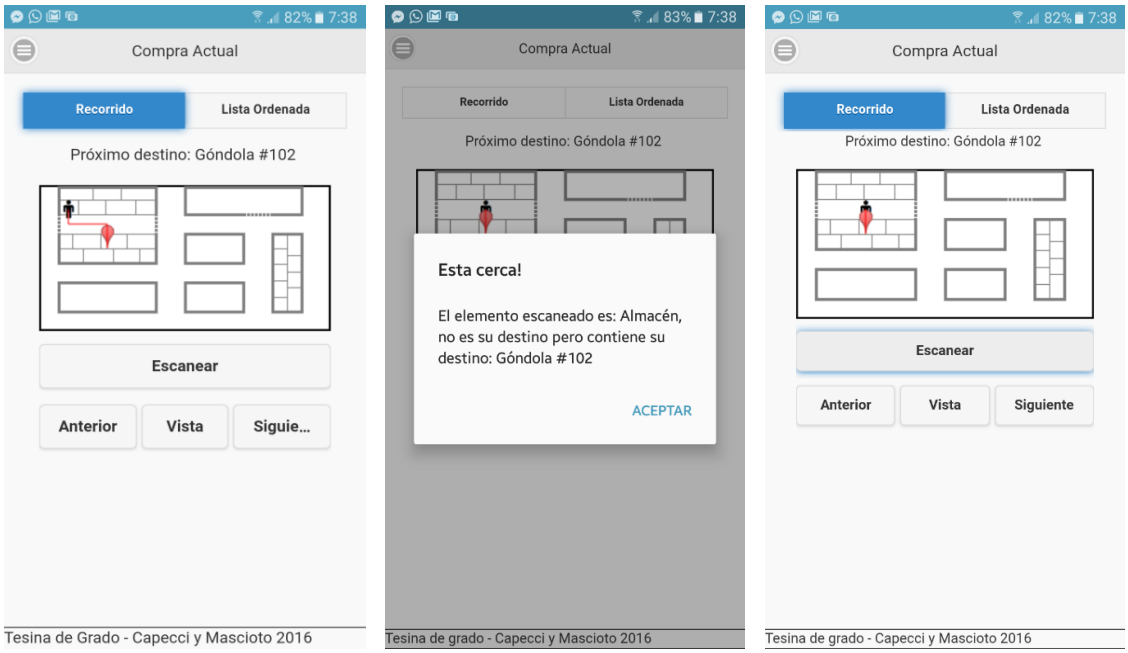


(a) **(b)** **(c)**

Figura 5.20: (a)Destino Producto. (b)Sector Contiene destino (c) Mapa actualizado

- *El destino era una Góndola y leyó el código QR del Sector que la contiene*

Supongamos que el usuario tiene como destino la *Góndola* con código "102" como se puede ver en la Figura 5.21.a, pero en vez de escanear el código destino escanea el código del sector *Almacén* que tiene como particularidad que contiene la góndola destino, el prototipo informa al usuario que se encuentra cerca de su destino (ver Figura 5.21.b) y actualiza el mapa con la posición del usuario y el camino a dicho destino como se puede observar en la Figura 5.21.c. Notar que no se cuenta con zoom, acorde a esto parece imperceptible el camino.



(a) **(b)** **(c)**

Figura 5.21: (a)Destino Producto. (b)Sector Contiene destino (c) Mapa actualizado

- *Lectura del código de Entrada del supermercado, teniendo como destino cualquier tipo de elemento.*

  Si el usuario escanea en cualquier momento de su compra el código QR de la entrada del supermercado, el prototipo le informará dicha situación con un mensaje tipo *popup* como se puede apreciar en la Figura 5.22 y actualiza la posición del usuario, brindándole un camino actualizado acorde a su nueva posición.

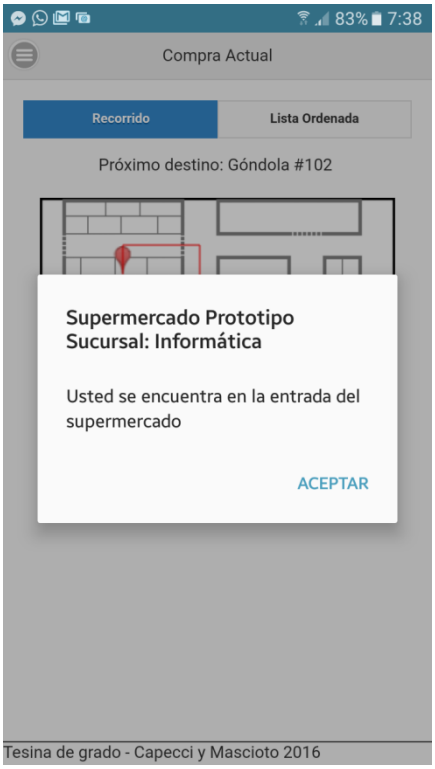
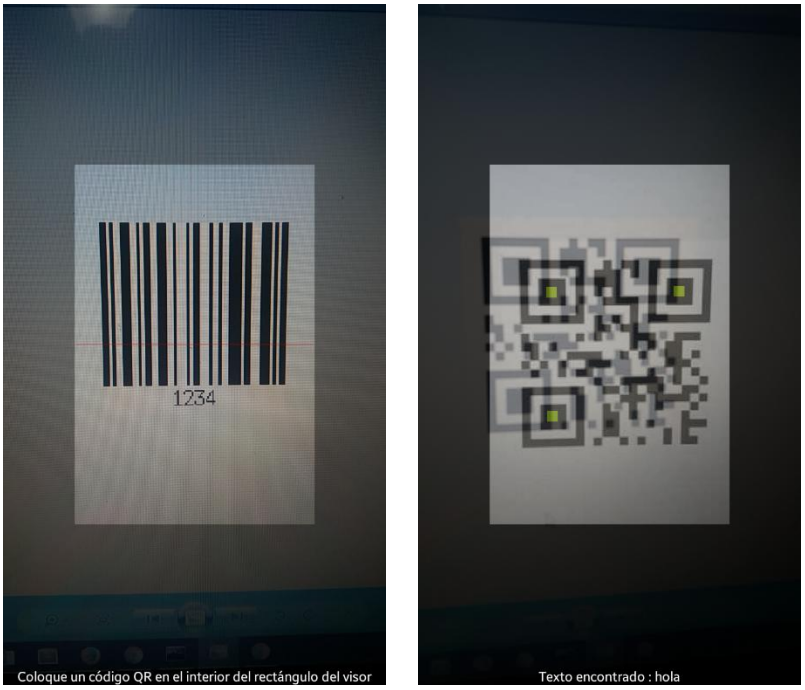


Figura 5.22: Lectura del código de entrada del supermercado mientras se está comprando.

A continuación se analiza como se comporta el protipo cuando se realiza la lectura de códigos QR que no son parte del supermercado.

Supongamos que el usuario a través de la funcionalidad "*Escanear Código*" mostrada en la Figura 4.26 (del Capítulo 4) escanear un código de barras, esto no producirá efecto alguno, ya que el *BarcodeScanner* está configurado para identificar solo códigos del tipo QR y no se procesará en código de barra como se puede apreciar en la Figura 5.23.a. (en la parte de abajo de dicha figura indica que se coloque un código QR). Cabe recordar que el *BarcodeScanner* fue configurado de esta manera con el parámetro "*formats*".

En el caso de escanear un código QR que no pertenece al código de ningún elemento del *Supermercado*, como se puede apreciar en la Figura 5.23.b donde el texto del código es "*hola*" (esto se puede observar en la parte de abajo de dicha figura). Luego, el prototipo informa con una ventana tipo *popup* que el código leído es un "*código inválido*" como se mostró en la Figura 4.29 (del Capítulo 4).



(a) **(b)**

Figura 5.23: *(a) Código de Barras. (b) Texto de código QR que no pertenece al supermercado*

6. CONCLUSIONES Y TRABAJOS FUTUROS

En este capítulo se presentan las conclusiones que se obtuvieron al final de este trabajo y se mencionan posibles trabajos futuros.
En conclusión podríamos decir que con el auge de los teléfonos celulares inteligentes (smartphones), se han desarrollado gran cantidad de aplicaciones para estos dispositivos. Debido a la inclusión de tecnología GPS en los smartphones, se puede ofrecer a los usuarios información o servicios de su entorno a partir de su posición. La problemática surge cuando la tecnología GPS falla y no es precisa en espacios interiores, donde las paredes del edificio atenúan y dispersan la señal del GPS aumentando el error de medición. Por su parte, las tecnologías de posicionamiento en interiores existentes carecen de soporte para smartphones o de precisión suficiente, requiriendo además la instalación de un hardware específico con un costo importante.

La problemática de posicionamiento en interiores ha sido objeto de un intenso estudio e investigación durante los últimos años. Hasta ahora, ninguna de las soluciones propuestas ha conseguido el éxito que han alcanzado los sistemas de posicionamiento y navegación análogos empleados en exteriores. La mayoría de las aplicaciones móviles de asistencia indoor son creadas ad hoc, es decir, no cuentan con un modelo de solución asociado para definir toda la información de los productos o servicios que brindan.

En esta tesis hemos presentado un modelo de solución para brindar asistencia a los usuarios dentro de un supermercado, escalable para comercios con similares características, por ejemplo, tiendas del estilo *"Hágalo usted mismo"*. Es decir, si bien fue probado con una cantidad acotada de sectores, góndolas y productos, estos podrían ser más. Más aún, el modelo podría ser extendido para soportar otro tipo de funcionalidades, como pueden ser las mencionadas en el Capítulo 2, por ejemplo, las recomendaciones alimenticias.

En base al modelo propuesto, se desarrolló un prototipo funcional. Para este prototipo se usaron códigos QR para identificar los elementos del supermercado, los cuales tienen una posición dentro del mismo. Tenemos en cuenta que este método no realiza un posicionamiento continuo ya que requiere la interacción del usuario para escanear el código, pero sería de bajo costo, y de esta manera el usuario puede recibir asistencia en un espacio indoor.

En cuanto en la generación de caminos cabe destacar que la solución elegida, donde se eligen algunos puntos estratégicos para a partir de allí buscar el camino, permite asistir al usuario para generarle caminos dinámicos sin importar donde se encuentre posicionado actualmente y hacia dónde tenga que ir. Si bien esta solución se usó para el supermercado, la misma estrategia podría ser usada en otros espacios indoor. La generación de estos puntos estratégicos (como se mostró en el Capítulo 4) depende mucho de por dónde se pueda transitar, en este ejemplo elegido eran solo algunos pasillos a considerar por lo tanto los puntos involucrados no eran muchos, pero en supermercados más grandes habría que evaluar cómo esta solución se pueda escalar. Por ejemplo, teniendo una herramienta visual que permita definir los pasillos transitables, y a partir de esos se generen los puntos dinámicamente, facilitando así usar un enfoque como el desarrollado para esta tesis.

A continuación se especifican algunos posibles trabajos futuros que se desprenden de la tesis propuesta.

- Si bien el prototipo se instanció para un único usuario, un trabajo futuro sería ampliar a varios usuarios, como hay información compartida en este caso habría que analizar para

contar con una solución cliente-servidor. Y más aún, se podría extender para contar con varios supermercados que compartan información, por ejemplo, cadenas de supermercados donde los precios son iguales para todos. Esto impactaría en el modelo pero el mayor impacto se daría en el prototipo, ya que se tendría que cambiar como se mencionaba anteriormente a una arquitectura cliente-servidor, requiriendo sincronización entre toda la información compartida.

- Asociado al punto anterior, se podría contar con un front-end de gestión de datos. Cada supermercado realiza la carga de sus datos y administra sus productos, góndolas y sectores. Pero también podría haber información compartida entre las cadenas de supermercados y esto permitiría no tener que cargar la misma información. Para el prototipo se tienen los datos precargados, pero se podría contar con un back-end con los datos almacenados en servidores. Asimismo, cada dispositivo móvil podría contar con una base de datos local que se sincronice con la centralizada para gestionar datos más específicos localmente, y no requerir consumir los mismos todo el tiempo.

- Otro aspecto que quedaría como línea futura, es implementar aquellas funcionalidades que se mencionaron en el Capítulo 4, que por simplificación sólo se implementaron como *template* a modo visual. Por ejemplo, la funcionalidad de registración.

- Se podría extender el modelo y el prototipo para que calcule y estime el tiempo que llevaría realizar la compra de una de las listas seleccionadas, tomando como variables de estimación un tiempo de caminata propendió dentro del supermercado acorde a la cantidad de usuarios logueados en estado comprando (no es lo mismo moverse en un supermercado vacío que con muchas personas).

- El modelo podría ser extendido para soportar otro tipo de funcionalidades, como se mencionó anteriormente. Se podrían agregar algunas de las características específicas de los trabajos relevados en el Capítulo 2, por ejemplo, las recomendaciones alimenticias relacionadas a cada producto, entonces en vez de ver solo información del producto, el usuario vería información alimenticia. Esto también impactaría en el prototipo funcional como puede apreciarse con esta descripción.

- Al permitir al usuario cargar su lista en casa, el modelo y el prototipo también se podrían extender con la funcionalidad para brindar supermercados cercanos. Entonces cuando el usuario tiene una lista de compras pendientes, recordárselo cuando pase, por ejemplo, por un supermercado o por el supermercado preferido de compra.

- El modelo podría ser extendido para considerar ofertas, y asistir a los usuarios considerando las mismas. Por un lado hay que modelar este concepto, y aspectos como la durabilidad de la oferta o las características de la misma, por ejemplo, "*llevando dos productos se lleva uno gratis*" o podría ser una oferta del estilo "*10% con pago en efectivo*". Las ofertas podrían estar asociadas a un producto o ser generales a todas las compras. Este tipo de extensión requiere un análisis previo al modelado para lograr que la oferta pueda representarse de manera general para cualquier situación para la que pudiera aplicarse. Esta extensión además de modificar el modelo también impactaría en el prototipo. Donde si son, por ejemplo, ofertas de un producto, al estar este posicionado, podría el usuario recibir ofertas cuando está posicionado cerca de los mismos, porque así si lo quiere comprar lo tiene cerca. Es decir, algunas ofertas podrían ser posicionadas y el usuario las recibe al estar cerca de las mismas.

- Una de las líneas de continuación de esta tesis, podría ser ampliar el dominio. Si bien el modelo y el prototipo fue desarrollado para supermercados, es escalable para tiendas con características similares. Hoy en día la mayoría de las personas utilizan servicios de posicionamiento para obtener información de lo que hay a su alrededor. Así por ejemplo, se podría transpolar a shoppings donde todos los aspectos de asistencia en la movilidad podrían ser reusados, y se tendrían que definir aspectos puntuales del dominio tales como tiendas (en otros tipos de conceptos de este dominio). Pero conceptos como la lista de compras o estrategias de cómo comprar podrían reusarse para este nuevo domino.

- Otro punto que requiere más análisis es el cálculo de caminos complejos y cómo mostrar los mismos, por citar un caso, para supermercados que tienen más de un piso. Habrá que seguir analizando trabajos del estilo de [Zlatanova et al., 2013] y [Jung and Lee, 2015] para mejorar la representación del espacio indoor. Dado los tiempos de la tesis, se eligió algunos puntos estratégicos conectados para a partir de allí buscar el camino (como se describió en el Capítulo 4). Como se mencionó anteriormente en las conclusiones, la generación de estos puntos estratégicos depende mucho de por dónde se pueda transitar físicamente, para transpolar la solución propuesta en esta tesis a otros supermercados, sería recomendable contar con una herramienta visual que permita definir los pasillos transitables, y a partir de esos se generen los puntos dinámicamente, facilitando así usar un enfoque como el desarrollado para esta tesis.

- Asociado al punto anterior, los planos (o imágenes) del supermercado (o de los supermercados) usados para la asistencia en la movilidad podrían ser cargados por los administradores del sistema del supermercado. Por ejemplo, se podría cargar para los productos el plano frontal de dónde están ubicados en la góndola. Hay que pensar que el prototipo fue implementado para pocos productos pero cargar todas las imágenes de todos los tipos de productos podría no ser una tarea trivial, más aun que los mismos se podrían cambiar de góndolas frecuentemente.

- Como una mejora al punto anterior, un trabajo futuro podría ser usar realidad aumentada en vez de planos de dónde está ubicado cada producto en la góndola. Esto agilizaría los datos a cargar ya que con tener la posición (x,y,z) de los productos se pude calcular qué mostrar con realidad aumentada. Por otro lado, esto requerirá un análisis de las capacidades necesarias de los dispositivos móviles para poder brindar realidad aumentada.

- La utilización del mecanismo de sensado a través de códigos QR tal vez no sea la mejor opción, ya que se necesita de la intervención del usuario para conocer su posición. Es decir, es un mecanismo indirecto pero tiene la ventaja de tener poco costo. Para contar con mecanismos directos de sensado, donde el usuario no tenga que estar leyendo códigos todo el tiempo, requiere un análisis de nuevas tecnologías que empiezan a surgir en el mercado, como la solución presentada en [Dickinson et al., 2016] donde se usan *Bluetooth Low Energy Beacons.* Cabe destacar que este trabajo fue encontrado una vez que ya se tenía la implementación desarrollada, pero nos parece interesante destacar como otra alternativa para contar con un sensado directo, al menos para góndolas y sectores. Pareciera que los códigos QR para los productos podría ser útil, aunque por ahí también se podría analizar usar el código de barra del producto para no tener que incorporar nada extra al mismo. Es interesante notar que cada vez más productos usan códigos QR para redireccionar al usuario a sus páginas, estos se podrían usar para identificar de que producto se trata.

Bibliografía

[Asthana et al., 1994] Asthana, A., Cravatts, M., & Krzyzanowski, P. (1994, December). An indoor wireless system for personalized shopping assistance. In Mobile Computing Systems and Applications, 1994. Proceedings., Workshop on (pp. 69-74). IEEE.

[Bai et al., 2014] Bai, Y. B., Wu, S., Ren, Y., Ong, K., Retscher, G., Kealy, A., ... & Zhang, K. (2014, October). A new approach for indoor customer tracking based on a single Wi-Fi connection. In Indoor Positioning and Indoor Navigation (IPIN), 2014 International Conference on (pp. 239-245). IEEE.

[Dalmasso et al., 2013] Dalmasso, I., Datta, S. K., Bonnet, C., Nikaein, N.: Survey, comparison and evaluation of cross platform mobile application development tools. In: 9th International Wireless Com-munications and Mobile Computing Conference, pp. 323-328. IEEE (2013)

[Dickinson et al., 2016] Dickinson, P., Cielniak, G., Szymanezyk, O., & Mannion, M. (2016, November). Indoor positioning of shoppers using a network of Bluetooth Low Energy beacons. In Indoor Positioning and Indoor Navigation (IPIN), 2016 International Conference on (pp. 1-8). IEEE.

[Emmanouilidis et al., 2013] Emmanouilidis, C., Koutsiamanis, R. A., & Tasidou, A. (2013). Mobile guides: Taxonomy of architectures, context awareness, technologies and applications. Journal of Network and Computer Applications, 36(1), 103-125.

[Heitkötter et al., 2012] Heitkötter, H., Hanschke, S., Majchrzak, T.A.: Evaluating cross-platform development approaches for mobile applications. In: Web information systems and technologies, pp. 120-138. Springer-Verlag Berlin Heidelberg (2012)

[Johnson et al., 1995] Johnson, R., Gamma, E., Helm, R., & Vlissides, J.:Design patterns: Elements of reusable object-oriented software. Boston, Massachusetts: Addison-Wesley, 1995.

[Jung and Lee, 2015] Jung, H., & Lee, J.: Indoor Subspacing to implement indoorGML for indoor navigation. International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences, 2015, 40.

[Lim, 2015] Lim, S.: Experimental Comparison of Hybrid and Native Applications for Mobile Sys-tems. Int. J. Multimed. Ubiquitous Eng 10(3), 1-12 (2015)

[López-de-Ipiña et al., 2011] López-de-Ipiña, D., Lorido, T., & López, U. (2011, June). Indoor navigation and product recognition for blind people assisted shopping. In International Workshop on Ambient Assisted Living (pp. 33-40). Springer Berlin Heidelberg.

[Muralidharan et al., 2014] Muralidharan, K., MISRA, A., & BALAN, R. K. (2014). Where Am I?: Studying Users' Indoor Navigation Location Needs.

[Put et al., 2014] Put, A., Dacosta, I., Milutinovic, M., De Decker, B., Seys, S., Boukayoua, F., & Martens, L. (2014, June). inshopnito: An advanced yet privacy-friendly mobile shopping application. In 2014 IEEE World Congress on Services (pp. 129-136). IEEE.

[Stamopoulos et al., 2014] Stamopoulos, S. F., Komninos, A., & Garofalakis, I. (2014, October). A Mobile shopping assistant to support product domesticity in consumer decisions. InProceedings of the 18th Panhellenic conference on informatics (pp. 1-6). ACM.

[Waltner et al., 2015] Waltner, G., Schwarz, M., Ladstätter, S., Weber, A., Luley, P., Bischof, H., & Paletta, L. (2015). MANGO-Mobile Augmented Reality with Functional Eating Guidance and Food Awareness. In New Trends in Image Analysis and Processing--ICIAP 2015 Workshops (pp. 425-432). Springer International Publishing.

[Yang et al., 2008] Yang, W. S., Cheng, H. C., & Dia, J. B. (2008). A location-aware recommender system for mobile shopping environments. Expert Systems with Applications, 34(1), 437-445.

[Zlatanova et al., 2013] Zlatanova, S., Liu, L., & Sithole, G.: A conceptual framework of space subdivision for indoor navigation. In Proceedings of the Fifth ACM SIGSPATIAL International Workshop on Indoor Spatial Awareness. ACM, 2013. p. 37-41.

Anexo A: BarcodeScanner

El plugin *BarcodeScanner* escanea códigos, utilizando la cámara del dispositivo móvil, y obtiene los datos codificados en ellos para procesarlos (más detalles de este plugin se puede consultar en *https://github.com/phonegap/phonegap-plugin-barcodescanner*). Actualmente soporta los siguientes tipos de códigos para *Android*:

- QR_CODE
- DATA_MATRIX
- UPC_E
- UPC_A
- EAN_8
- EAN_8
- CODE_128
- CODE_39
- CODE_93
- CODABAR
- ITF
- RSS14
- RSS_EXPANDED

Cada uno de estos tipos de códigos tiene su propia codificación, en particular para esta tesis solo nos focalizaremos en los códigos QR.

Desde *JavaScript* el plugin es invocado usando el nombre "cordova.plugins.barcodeScanner", esto en tiempo de ejecución se transforma en un objeto de la clase *BarcodeScanner* la cual es una clase dentro de los plugins provistos por *PhoneGap*.

Uno de los métodos que se pueden invocar al objeto *BarcodeScanner* es el método `scan(success, fail, options)`, donde tanto success y fail son funciones callback. Estas funciones son las que se invocan tanto si la lectura del código fue exitosa o no respectivamente.

En el Código A.1 se puede apreciar un ejemplo de cómo usar el método `scan(success, fail, options)`, esta sería la forma de usarlo desde *JavaScript*. Se puede apreciar las dos funciones pasadas como parámetros de dicho métodos, estas se ejecutarán dinámicamente acorde a si la lectura del código fue o no exitosa.

```
cordova.plugins.barcodeScanner.scan(
      function (result) {
          alert("We got a barcode\n" +
                                "Result: " + result.text + "\n" +
                                "Format: " + result.format + "\n" +
                               "Cancelled: " + result.cancelled);
      },
      function (error) {
          alert("Scanning failed: " + error);
      },
      {
          "preferFrontCamera" : true, // iOS and Android
          "showFlipCameraButton" : true, // iOS and Android
          "showTorchButton" : true, // iOS and Android
          "disableAnimations" : true, // iOS
          "prompt" : "Place a barcode inside the scan area", //
                                              supported on Android only
          "formats" : "QR_CODE,PDF_417", // default: all but PDF_417
                                               and RSS_EXPANDED
          "orientation" : "landscape" // Android only
                                (portrait|landscape), default unset so
                                it  rotates with the device
      }
  );
```

Código A.1: Ejemplo uso del método scan() de la clase *BarcodeScanner*

Otro método que se puede invocar del objeto *BarcodeScanner* es encode(type, data, success, fail). Los tipos (type) de codificación soportados por este método son:

- TEXT_TYPE
- EMAIL_TYPE
- PHONE_TYPE
- SMS_TYPE

En el Código A.2 se puede apreciar un ejemplo de cómo usar el método encode(type, data, success, fail), esta sería la forma de usarlo desde *JavaScript*.

```
cordova.plugins.barcodeScanner.encode(
        cordova.plugins.barcodeScanner.Encode.TEXT_TYPE, "http://www.nytimes.com",
        function(success) {
        alert("encode success: " + success);
    },
        function(fail) {
       alert("encoding failed: " + fail);
    }
);
```

Código A.2: Ejemplo uso del método encode() de la clase *BarcodeScanner*

Anexo B: Configuración del archivo config.xml del Prototipo

Uno de los archivos que se deben configurar para que funcione PhoneGap es el archivo config.xml. Este archivo tiene la configuración global de la aplicación, es un archivo XML independiente de la plataforma en donde se definen, por ejemplo, los plugins a usar y todo lo referente a la configuración específica de la plataforma.

Se puede apreciar en el Código B.1 la configuración realizada para el archivo config.xml del Prototipo. Se puede ver cómo se definió el nombre, la página de inicio (index.html), diferentes preferencias, se puede observar además que dentro de los plugin se especifica el cordova-plugin-barcodescanner (el cual fue descripto en el Anexo A). Y al final de todo el archivo se puede ver la plataforma Android especificada.

A continuación se especifican más detalles de cada uno de los tags usados en este archivo para clarificar cada uno de ellos:

- Atributo id del elemento <widget> proporciona identificador de reversa-dominio de la aplicación y la versión de su número de versión completa expresada en notación de mayor/menor/parche. La etiqueta widget también puede tener atributos que especifican las versiones alternativas, a saber versionCode para *Android*.
- El elemento <name> especifica nombre formal de la aplicación, como aparece en la pantalla principal del dispositivo y dentro de la tienda app interfaces.
- El elemento <description> y <author> especifican metadatos e información de contacto que puede aparecer en anuncios de la tienda app.
- Opcional <content> elemento define la página de inicio de la aplicación en el directorio web de alto nivel de activos. El valor predeterminado es index.html que aparece en el directorio de nivel superior www de nuestro proyecto.
- Elementos <access> definen el conjunto de dominios externos que puede comunicarse con la aplicación.
- La etiqueta <preference> establece varias opciones como pares nombre/valor de atributos. De cada preferencia name es sensible a mayúsculas.
- La etiqueta <plugin> se utiliza para habilitar APIs de nivel de dispositivo y plugins externos.

```
<?xml version='1.0' encoding='utf-8'?>
<widget id="com.phonegap.supermercadoInteligente" version="1.0.0" xmlns="http://www.w3.org/ns/widgets"
xmlns:gap="http://phonegap.com/ns/1.0">
        <name>Supermercado Inteligente</name>
        <description>
                Prototipo Tesina de Grado 2016
        </description>
        <author email="support@phonegap.com" href="http://phonegap.com">
                PhoneGap Team
        </author>
        <content src="index.html" />
        <preference name="permissions" value="none" />
        <preference name="orientation" value="portrait" />
        <preference name="target-device" value="universal" />
        <preference name="fullscreen" value="false" />
        <preference name="webviewbounce" value="true" />
        <preference name="prerendered-icon" value="true" />
        <preference name="stay-in-webview" value="false" />
        <preference name="detect-data-types" value="true" />
        <preference name="exit-on-suspend" value="false" />
        <preference name="show-splash-screen-spinner" value="true" />
        <preference name="auto-hide-splash-screen" value="true" />
        <preference name="disable-cursor" value="false" />
        <preference name="android-minSdkVersion" value="14" />
        <preference name="android-installLocation" value="auto" />
        <plugin name="cordova-plugin-camera" source="npm"/>
        <plugin name="cordova-plugin-console" source="npm" />
        <plugin name="cordova-plugin-device" source="npm" />
        <plugin name="cordova-plugin-dialogs" source="npm" />
        <plugin name="cordova-plugin-inappbrowser" source="npm" />
        <plugin name="cordova-plugin-media" source="npm" />
        <plugin name="cordova-plugin-network-information" source="npm" />
        <plugin name="cordova-plugin-barcodescanner" source="npm" />
        <icon src="www/res/icon/icon.png" />
        <icon gap:platform="android" gap:qualifier="ldpi" src="www/res/icon/android/icon-36-ldpi.png" />
        <icon gap:platform="android" gap:qualifier="mdpi" src="www/res/icon/android/icon-48-mdpi.png" />
        <icon gap:platform="android" gap:qualifier="hdpi" src="www/res/icon/android/icon-72-hdpi.png" />
        <icon gap:platform="android" gap:qualifier="xhdpi" src="www/res/icon/android/icon-96-xhdpi.png" />
        <icon gap:platform="blackberry" src="www/res/icon/blackberry/icon-80.png" />
        <icon gap:platform="blackberry" gap:state="hover" src="www/res/icon/blackberry/icon-80.png" />
        <icon gap:platform="webos" src="www/res/icon/webos/icon-64.png" />
        <icon gap:platform="winphone" src="www/res/icon/windows-phone/icon-48.png" />
        <icon gap:platform="winphone" gap:role="background" src="www/res/icon/windows-phone/icon-173-tile.png" />
        <gap:splash gap:platform="android" gap:qualifier="port-ldpi" src="www/res/screen/android/screen-ldpi-portrait.png" />
        <gap:splash gap:platform="android" gap:qualifier="port-mdpi" src="www/res/screen/android/screen-mdpi-portrait.png"
        />
        <gap:splash gap:platform="android" gap:qualifier="port-hdpi" src="www/res/screen/android/screen-hdpi-portrait.png" />
        <gap:splash gap:platform="android" gap:qualifier="port-xhdpi" src="www/res/screen/android/screen-xhdpi-portrait.png"
        />
        <gap:splash gap:platform="blackberry" src="www/res/screen/blackberry/screen-225.png" />
        <gap:splash gap:platform="winphone" src="www/res/screen/windows-phone/screen-portrait.jpg" />
          <access origin="*" />
        <plugin name="cordova-plugin-whitelist" version="1" />
        <allow-navigation href="http://*/*" />
        <allow-intent href="http://*/*" />
        <allow-intent href="https://*/*" />
        <allow-intent href="tel:*" />
        <allow-intent href="sms:*" />
        <allow-intent href="mailto:*" />
        <allow-intent href="geo:*" />
        <platform name="android">
            <allow-intent href="market:*" />
        </platform>

</widget>
```

Código B.1: Definición del archivo config.xml del Prototipo

Anexo C: Invocación del *BarcodeScanner* dentro del Prototipo

En este anexo se presenta cómo se realiza la lectura de códigos QR desde el prototipo, se puede apreciar en el código C.1 la función scanCode(), donde se invoca al cordova.plugins.barcodeScanner.scan() acorde a los descripto en el Anexo A, dependerá de cómo fue la lectura y del estado del usuario es cómo se comporta el mismo.

```
function scanCode(event){
 try{
  cordova.plugins.barcodeScanner.scan(
   function (result){
    console.log("Result: " + result.text + " Format: " + result.format + " Cancelled: " + result.cancelled);

    if (result.cancelled){
                navigator.notification.alert("Scanner cancelado", null, "Error", "Aceptar");
    }else{
                /* los unicos estados posibles para escanear son Mirando y Comprando */
                var info;
                if (clienteLogueado.getEstadoActual() != "Comprando"){
                        /* estado  actual: mirando */
                        clienteLogueado.estadoActual = new Mirando();
                        info = supermercado.escanear(result.text, clienteLogueado);
                        if(info == false){
                                navigator.notification.alert("Código inválido", null, "Error", "Aceptar");
                        }else{
                                switch (info[0]){
                                 case "producto":
                                   showResultadoProducto(info);
                                   break;
                                 case "gondola":
                                   showResultadoGondola(info);
                                   break;
                                 case "sector":
                                   showResultadoSector(info);
                                   break;
                                 case "entrada":
                                   showResultadoEntrada(info);
                                   break;
                                }
                        }
                }else{
                        /* estado actual: comprando */
                        //comenzó la compra o click en boton Escanear - procesa el elemento
                        info = supermercado.escanear(result.text, clienteLogueado);
                        if(info == false){
                                navigator.notification.alert("Código inválido", null, "Error", "Aceptar");
                        }else{
                            if(info[0] == "entrada"){
                                drawMap(clienteLogueado.posicionActual, new PositionPoint3D($('#x-destino').val(), $('#y-
                                destino').val(),  $('#z-destino').val()));
                                 navigator.notification.alert(info[3], null, info[2], "Aceptar" );
                            }
                            if (info[0] != -1){
                                //inicio de compra o se procesó con exito el elemento
                                if(info[0] == 0){
                                        //inicio de compra
                                        $("#compra-anterior-button").attr("disabled", true);
                                }else{
                                        $("#compra-anterior-button").attr("disabled", false);
                                }
                                if(info[2] != null){
                                        $("#elementoDestino").text("Próximo destino: " + info[2].getNombre()).trigger( "create" );
                                        $('#tipoDestino').val(info[2].getInformacion()[0]).trigger( "change" );
                                        changeDestino();
                                        $('#x-destino').val(info[2].getUbicacion().getPoint().x);
                                        $('#y-destino').val(info[2].getUbicacion().getPoint().y);
                                        $('#z-destino').val(info[2].getUbicacion().getPoint().z);

                                        $("#subtotal").text(clienteLogueado.recorridoActual.subtotal).trigger( "create" );
                                        drawMap(clienteLogueado.posicionActual,info[2].getUbicacion().getPoint());
                                        $.mobile.changePage("#compraActual");
```

```
                        }else{
                                //se procesó el último elemento
                                clienteLogueado.recorridoActual = null;
                                clienteLogueado.listaActual = null;
                                drawMap(clienteLogueado.posicionActual,null);
                                //deshabilitar botones
                                $("#compra-scan-button").attr("disabled", true);
                                $("#compra-vista-button").attr("disabled", true);
                                $("#compra-comprar-button").attr("disabled", true);
                                $("#compra-anterior-button").attr("disabled", true);
                                $("#compra-siguiente-button").attr("disabled", true);

                                $("#elementoDestino").text("Compra Finalizada").trigger( "create" );
                                navigator.notification.alert("Compra finalizada!", null, "", "Aceptar");
                        }
                    }else{
                        // muestra el mapa, sugiere comprar si es producto o seguir el orden de la lista de compra
                        if(info[2]){
                                navigator.notification.alert("El elemento escaneado es: "+ info[3].getNombre() + ", no es su
                                destino pero contiene su destino: " + info[4].getNombre(), null, "Esta cerca!", "Aceptar");
                        }else{
                                if(info[3].getInformacion()[0] == "producto"){
                                        navigator.notification.confirm("El producto escaneado es: "+ info[3].getNombre()
                                        + ", no es su destino.\nPuede comprarlo o seguir con el proximo elemento.",
                                        confirmCallback , "", "Comprar, Seguir" );
                                }else{
                                        navigator.notification.alert("El elemento escaneado es: "+ info[3].getNombre() + ",
                                        no es su destino.\nPróximo elemento: " + info[4].getNombre(), null, "", "Aceptar");
                                }
                        }
                        $("#elementoDestino").text("Próximo destino: " + info[4].getNombre()).trigger( "create" );
                        $('#tipoDestino').val(info[4].getInformacion()[0]).trigger( "change" );
                        changeDestino();
                        $('#x-destino').val(info[4].getUbicacion().getPoint().x);
                        $('#y-destino').val(info[4].getUbicacion().getPoint().y);
                        $('#y-destino').val(info[4].getUbicacion().getPoint().z);
                        drawMap(clienteLogueado.posicionActual,info[4].getUbicacion().getPoint());
                        $.mobile.changePage("#compraActual");
                    }
                }
            }
        }
    },
    function (error){
        navigator.notification.alert("Scanning failed: " + error, null, "Error", "Aceptar");
    },
    {
      "orientation": "portrait",
      "prompt" : "Coloque un código QR en el interior del rectángulo del visor",
      "formats" : "QR_CODE"
    }
   );
  } catch (e){
    console.log('---------------------------- ERROR');
    console.log(e);
  }
}
```

Código C.1: Invocación del *BarcodeScanner* dentro del propotipo.