UNIVERSIDAD AUTÓNOMA METROPOLITANA UNIDAD AZCAPOTZALCO

DIVISIÓN DE CIENCIAS BÁSICAS E INGENIERÍA

LICENCIATURA EN INGENIERÍA EN COMPUTACIÓN

APLICACIÓN MÓVIL (CANBAS CHEAP) PARA ADQUIRIR LA CANASTA BÁSICA A MENOR PRECIO EN LAS TIENDAS MÁS CERCANAS

PROYECTO TECNOLÓGICO

TRIMESTRE 2017-INVIERNO

ÁLVAREZ CORDERO MARÍA LUISA

2122000179

einobyrem@hotmail.com

ASESORA: GONZÁLEZ BRAMBILA SILVIA BEATRIZ

DRA. EN CIENCIAS DE LA COMPUTACIÓN

PROFESORA TITULAR

DEPARTAMENTO DE SISTEMAS

sgb@correo.azc.uam.mx

Fecha de entrega: 25 de Abril de 2017

DECLARATORIA

Yo, Dra. Silvia Beatriz González Brambila, declaro que aprobé el contenido del presente Reporte de Proyecto de Integración y doy mi autorización para su publicación en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.

González Brambila

Yo, María Luisa Álvarez Cordero, doy mi autorización a la Coordinación de Servicios de Información de la Universidad Autónoma Metropolitana, Unidad Azcapotzalco, para publicar el presente documento en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.

María V.

RESUMEN

De acuerdo a la información que proporciona el INPC (Índice Nacional de Precios al Consumidor) el INEGI (Instituto Nacional de Geografía e Informática) ha declarado que en los últimos años se ha presentado un aumento en los precios de los productos que conforman la canasta básica, además de que este puede variar dependiendo del lugar en que se adquieran dichos productos.

Actualmente todas las personas o la gran mayoría cuentan con un teléfono celular en él cuál se pueden descargar diversas aplicaciones con distintos fines, algunas aplicaciones permiten precisamente consultar los precios de los productos de interés o incluso mostrar ofertas en las distintas tiendas.

En el presente proyecto se desarrolla una aplicación móvil multiplataforma en la cual los usuarios participan activamente para registrar productos con sus respectivos precios en distintos establecimientos, además de consultar los productos que forman parte de su canasta básica y seleccionar el radio de distancia para la búsqueda de tiendas que cuenten con estos productos al mejor precio.

La aplicación consta de cinco módulos, donde en cada uno, los usuarios pueden realizar diversas acciones:

- Registro/Login: registrarse para posteriormente acceder a la aplicación y hacer uso de sus funciones.
- Ver Perfil: consultar sus puntos, su credibilidad y seleccionar su ubicación actual.
- Agregar Ubicación: registrar una ubicación.
- Registro de Productos: registrar tiendas y productos con su precio permitiendo así la clasificación de los productos.
- Búsqueda de Canasta: seleccionar los productos que forman parte de su canasta básica, seleccionar la cantidad de estos y colocar el radio de distancia en el que les gustaría realizar la búsqueda de tiendas cercanas.
  Este módulo también le permite al usuario visualizar la ruta a seguir para llegar a la tienda seleccionada previamente.

Todas estas operaciones son invocadas por medio de servicios web, lo que implica que sea el servidor el que se encargue de acceder a la base de datos y realizar los registros y consultas correspondientes para dar respuestas correctas a las solicitudes que hace la aplicación cliente. Esta se encuentra programada en lenguaje Java y JavaFX, hace uso de Gluon Mobile y Scene

Builder para el desarrollo y la interfaz de usuario. Por su parte la aplicación del servidor está programada en lenguaje Java además de hacer uso de Apache Tomcat para el correcto despliegue de los servicios web.

Para las vistas que hacen uso de Google Maps se emplea GMapsFX que trabaja con Java y no con JavaScript, lo que hace que el desarrollo de la aplicación cliente sea más uniforme, dado que la mayoría del código fuente está programado en Java.

Esta aplicación es útil puesto que realiza correctamente la búsqueda de tiendas más cercanas que cuenten con los productos seleccionados por el usuario, actualmente se tienen aplicaciones de este tipo pero con interfaces algo complicadas, por lo que CanbasCheap podría ser una buena opción para aquellos usuarios interesados en este tipo de aplicaciones.

Actualmente no se considera una aplicación comercializable ya que aún no es funcional la parte que muestra las vistas de Google Maps en dispositivos móviles, pero podría ser la base para el desarrollo de una aplicación que pueda distribuirse en el mercado y con ello contribuir a que los usuarios realicen sus compras en las tiendas que tengan los productos más baratos.

Tabla de contenido

Índice de Figuras

Índice de Tablas

1.-INTRODUCCIÓN

"La canasta básica es un conjunto de bienes y servicios indispensables para que una familia pueda satisfacer sus necesidades básicas de consumo a partir de su ingreso." [1].

Según el estudio [2] que hace el INEGI (Instituto Nacional de Geografía e Informática), se tienen 82 productos en la canasta básica, organizados de acuerdo a categorías y determinados en base a la Encuesta Nacional de Ingresos y Gastos de los Hogares (ENIGH) [1], con los cuáles se puede calcular el costo promedio de cada producto y construir el INPC (Índice Nacional de Precios al Consumidor) en el cual queda establecido el peso (ponderación) de cada producto. De acuerdo a la información que proporciona el INPC, el INEGI ha declarado que en los últimos años se ha presentado un aumento en los precios de los productos de la canasta básica; esto sin tomar en cuenta que el precio de los productos puede variar de acuerdo a la tienda en que se adquieran.

Se sabe que todos los productos comercializados tienen un código de barras asociado, el cuál es usado para identificarlos, saber de qué país provienen y de qué empresa.

En México y otros países [3] se utiliza el sistema EAN/UCC[1] para difundir y administrar los estándares de identificación de productos. A los códigos para unidades de consumo y expedición, se les conoce también como GTINs (*Global Trade Item Numbers*) los cuáles identifican de manera única cada producto o servicio ofrecido en el mercado, los GTINs son asignados localmente, pero son únicos a nivel mundial; estos códigos son representados por códigos de barras.

El proyecto consiste en desarrollar una aplicación que permita a los usuarios adquirir su canasta básica a menor precio y seleccionar el radio de distancia para encontrar las tiendas más cercanas, haciendo uso de la tecnología de Google Maps y búsqueda de productos a través del código GTIN.

[1] EAN/UCC (European Article Numbering Association)/ (Uniform Code Council): Unión de dos organizaciones que manejan estándares compatibles de código de comercio.

1.1.- ANTECEDENTES

1.1.2.- Trabajos Internos

Aplicación móvil para la recomendación de productos en el comercio electrónico. [4]. Es una aplicación que recomienda productos de cómputo, haciendo uso de la ponderación de cada producto, según la valoración que den los consumidores y en base al algoritmo LSP (Logic Scoring of Preferences).

1.1.3.- Trabajos externos

¿Quién es quién en los precios? [5]. Esta aplicación propuesta por la PROFECO (Procuraduría Federal del Consumidor) compara los precios del producto de la canasta básica que el usuario desea consumir en las diferentes tiendas, para poder ofrecerle los precios más baratos, así como también permite al usuario introducir el radio de distancia en que se hará la búsqueda de las tiendas más cercanas.

Radarprice [6]**.** Es un rastreador web y móvil que, tras escanear el código de barras de un artículo con cualquier dispositivo móvil permite: conocer su precio al instante en más de 1,200 plataformas web y encontrar las tiendas más cercanas con el mejor precio.

Idealo [7]. Ofrece una búsqueda de productos por nombre, escribiendo el código de barras o escaneándolo. Tiene una base de datos de más de 4,000 tiendas online. Incluye numerosas categorías de producto que sirven como filtros.

Análisis, diseño e implementación de un sistema de control de ingreso de vehículos basado en visión artificial y reconocimiento de placas en el parqueadero de la universidad politécnica Salesiana-Sede Cuenca [8]**.** Consiste en un sistema de reconocimiento de placas vehiculares que utiliza algoritmos de Reconocimiento Óptico de caracteres (OCR) para poder generar un mejor control, estadísticas y personal de quienes ingresan a la Universidad Politécnica Salesiana-Sede Cuenca.

Reconocimiento de caracteres mediante imágenes en contadores de gas en entornos reales [9]**.** Consiste en la automatización del proceso de lectura de contadores de gas en una aplicación informática que permita además reconocimiento de caracteres en imágenes, para interpretar y asociar las lecturas del consumo de gas, así como su relación con el número de suministro contratado.

1.2.- JUSTIFICACIÓN

Según estudios realizados por el INEGI la canasta básica tuvo un aumento de precios significativo [10], se sabe que los precios pueden variar de acuerdo a la tienda; por lo que si se tuviera conocimiento de las tiendas en dónde los productos son más baratos, el usuario tendría la posibilidad de adquirir su canasta básica a menor precio.

Con la aplicación que se desarrollará en este proyecto se conformará una comunidad de usuarios en la cual podrán realizar aportaciones y consultar datos.

Los usuarios que realicen aportaciones serán aquellos que envíen una foto de la etiqueta del producto junto con su precio para ser procesados por la aplicación y actualizar la base de datos con los precios y la tienda en la que se encuentra dicho usuario.

Aquellos usuarios que sólo deseen realizar una consulta, bastará co n que ingresen el nombre del producto y la aplicación les proporcionará los diferentes precios del producto en las tiendas consideradas dentro de un rango definido.

La aplicación contará también con un apartado en dónde se encuentren los productos que conforman la canasta básica, dándole al usuario la posibilidad de marcar la cantidad de aquellos que desea comprar y calculando el precio total de la canasta básica establecida.

El usuario será libre de elegir el radio de distancia en el cuál se realizará la búsqueda de tiendas, esto con la finalidad de que encuentre los productos de la canasta básica que va a consumir a un mejor precio y sin tener que ir más lejos de lo que él decida; además la aplicación le mostrará al usuario la ruta que debe seguir para llegar a la tienda seleccionada.

En esta aplicación los usuarios podrán interactuar para dar a conocer precios de productos en tiendas específicas y opiniones, de tal forma que los consumidores estén mejor informados para tomar decisiones.

1.3.- OBJETIVOS

1.3.1.-Objetivo General

- Desarrollar una aplicación móvil que permita elegir la tienda más conveniente de acuerdo al precio de su canasta básica y la distancia, utilizando tecnología de Google Maps.

1.3.2.-Objetivos Específicos

- Diseñar e implementar el módulo de registro de usuarios considerando criterios de usabilidad en la interfaz gráfica.

  Todas las vistas implementadas en la aplicación cliente consideran criterios de usabilidad.
  Este objetivo se cumple satisfactoriamente puesto que la aplicación cliente establece una correcta comunicación con el servidor y envía correctamente los datos del usuario a registrar.
  Para la parte de Login, la aplicación cliente envía el usuario y la contraseña y el servidor se encarga de verificar que exista en la base de datos un usuario con este nombre de usuario y contraseña, si es así envía el objeto usuario en formato JSON[2]*.*

- Diseñar e implementar un módulo para reconocer el código GTIN y buscarlo en la base de datos.

  Este objetivo se cumple haciendo uso del plugin ScanBarcode que proporciona Gluon Charm Down, el cual se encarga precisamente de leer el código de barras de distintos productos para posteriormente ser enviado al servidor y hacer correctamente el registro, además de permitir también la búsqueda del producto a través de este código.

- Diseñar e implementar un módulo para clasificar los precios de acuerdo a los niveles: bajo, promedio o alto y determinar su credibilidad.

  Este objetivo se cumple puesto que antes de realizar un registro de producto en tienda, primero verifica que no exista ningún registro de este, si es así guarda el producto como primer registro y coloca en el rango el valor de 'Medio', por otra parte, si encuentra registros existentes del producto se encarga de comparar los precios e irlos clasificando en 'Bajo' o 'Alto'.

- Diseñar e implementar un módulo para ubicar a las personas más activas.

  Este objetivo se cumple puesto que el servidor cuenta con una función encargada de modificar los puntos del usuario y su credibilidad cada

[2] JSON, acrónimo de JavaScript Object Notation, es un formato ligero para el intercambio de datos.

vez que este hace una búsqueda o registro de productos, por lo que si se requiere consultar quienes son los más activos solo se tienen que buscar a aquellos con la mayor credibilidad.

- Diseñar e implementar un módulo que permita al usuario ingresar el radio de distancia para la búsqueda de tiendas.

  Este objetivo se cumple gracias a que la aplicación cliente le da la posibilidad al usuario de seleccionar el radio de distancia en el que desea hacer la búsqueda de tiendas y este dato es enviado al servidor el cual tiene una función que por medio de la fórmula de Haversine (se describe en la sección 1.4.2) puede determinar la distancia entre la ubicación actual del usuario y las diferentes tiendas y en base a ello determinar si se encuentran dentro del rango de distancia o no.

- Diseñar e implementar un módulo que permita buscar en las tiendas encontradas los diferentes productos solicitados por categoría o nombre, utilizando la base de datos con productos y tiendas relacionados.

  Este objetivo se cumple puesto que en la aplicación cliente, una vez que el usuario selecciona Búsqueda de Canasta, se envía una petición al servidor para crear una nueva lista, la cual se utiliza después para comparar los productos que contiene con los productos existentes en las distintas tiendas cercanas y con base en ello determinar cuáles disponen de todos los productos que solicita el usuario y enviar la lista resultante a la aplicación cliente.

- Diseñar e implementar un módulo para seleccionar la tienda y mostrar al usuario la ruta a seguir.

  Este objetivo se cumple gracias a que una vez mostradas al usuario las tiendas que se encuentran dentro del rango de distancia y que además tienen todos los productos solicitados, el usuario puede seleccionar una y con base en ello la aplicación hace uso de GMapsFX y aplica la funcionalidad de Directions API para obtener y mostrar la ruta a seguir desde la ubicación actual del usuario a la tienda seleccionada.

1.4.- MARCO TEÓRICO

A continuación, se presentan los conceptos más importantes que son manejados a lo largo del presente reporte para explicar el desarrollo de Canbas-Cheap.

1.4.1.- Google Maps

Es una aplicación y sitio web que permite a los usuarios desplazarse por los mapas de todo el mundo e ir ampliando las zonas de interés para apreciar fotos satelitales de gran calidad, mapeos vectoriales, mapas con calles y rutas, etc. [11]

Google Maps cuenta con diferentes APIs tales como:

*Google Maps Android API

*Google Maps JavaScript API

*Google Maps Geocoding API

*Google Maps Directions API

*Google Maps Places API Web Service

Canbas-Cheap no hace uso de todas las APIs de las que dispone Google Maps, pero sí hace uso de:

Google Maps Geocoding API

Esta herramienta tiene la funcionalidad de convertir direcciones en coordenadas geográficas (latitud, longitud). Por lo que se puede aplicar en diversas situaciones, como por ejemplo para localizar algún lugar en la superficie terrestre, esto sería una tarea imposible si solo conoces la dirección, pero gracias a Google Geocoding API es posible convertir esta dirección en coordenadas y en base a ello tener la posición exacta de este lugar.

Para emplear esta herramienta es necesario hacer uso de los siguientes *parámetros obligatorios* [12].

- ***Address:*** la dirección que quieres geocodificar, en el formato utilizado por el servicio postal nacional del país correspondiente.
- ***Components:*** un filtro de componente para el que quieres obtener un geocódigo.
- ***key:*** la clave de API de tu aplicación.

Geocoding API tiene también la funcionalidad de convertir coordenadas geográficas en direcciones que pueden ser entendidas por los usuarios (***geocodificación inversa).*** Es decir, si solo se conocen las coordenadas de algún lugar, pero se quiere saber la dirección, conociendo la calle, la colonia, el código postal y la ciudad en que se encuentra, se puede fácilmente hacer uso de esta API y obtener así la dirección.

Para hacer uso de la geocodificación inversa, los *parámetros obligatorios* [12] que se deben usar en la solicitud son los siguientes:

Ya sea, ***latlng:*** los valores de latitud y longitud que especifican la ubicación para la que se quiere obtener la dirección más cercana en lenguaje natural.

O bien, ***place_id:*** el id. de sitio del sitio para el que se quiere obtener la dirección en lenguaje natural.

Google Maps Directions API

Esta herramienta da la posibilidad de conocer la ruta a seguir para llegar a un determinado lugar, tomando en cuenta la posición actual u origen. Google Directions API muestra, si así se desea, las indicaciones a tomar para llegar al lugar de destino.

Para hacer uso de esta herramienta es necesario incluir los siguientes *parámetros obligatorios* [13] en la solicitud:

- ***Origin:*** la dirección, el valor de latitud/longitud textual o el id. de sitio desde el que se quiere calcular las indicaciones.
- ***Destination:*** la dirección, el valor de latitud/longitud textual o el id. de sitio hasta el que se quiere calcular las indicaciones.

Para que Google Directions API muestre correctamente las indicaciones es necesario que proporcionar el modo (***mode***) de viaje que se usará para llegar al destino.

Existen distintos modos de viaje tales como:

- ***driving*** (predeterminado) brinda indicaciones de manejo estándar usando la red de carreteras.
- ***walking*** solicita indicaciones de traslado a pie por sendas peatonales y veredas (cuando estén disponibles).
- ***bicycling*** solicita indicaciones para el traslado en bicicleta por ciclovías y calles preferidas (cuando estén disponibles).
- ***transit*** solicita indicaciones por rutas de transporte público (cuando estén disponibles).

1.4.2. FÓRMULA DE HAVERSINE

Esta es una fórmula que permite calcular la distancia entre dos puntos sobre la superficie de la Tierra, haciendo uso de la Latitud y Longitud de cada punto.

La fórmula es:

$$haversin\left(\frac{d}{R}\right) = haversin(\varphi 1 - \varphi 2) + \cos(\varphi 1)\ \cos(\varphi 2) haversin(\Delta\lambda)$$

[14] Dónde:

haversin es la función haversine, haversin (θ) = sen 2 (θ /2) = (1-cos (θ))/2.

d es la distancia entre dos puntos (sobre un círculo máximo de la esfera)

R es el radio de la esfera,

φ1 es la latitud del punto 1,

φ2 es la latitud del punto 2, y

Δ λ es la diferencia de longitudes

El argumento a la función haversine se supone que debe darse en radianes, si se da en grados, el haversin (d/R) de la fórmula se convertiría en haversin (180 · d /π R).

Entonces se puede resolver mediante la función arcoseno:

$$d = R\ haversin^{-1}(h) = 2Rarcsin\left(\sqrt{h}\right)$$

Esta fórmula resultante es la que se aplica en el Servidor de Canbas-Cheap para hacer el cálculo de distancias.

Donde:

h es haversin (d / R).

1.4.3.- RESTFUL WEB SERVICE

(REST) es un estilo arquitectónico que especifica las restricciones, como la interfaz uniforme, que, si se aplica a un servicio web inducir propiedades deseables, tales como el rendimiento, la escalabilidad y la capacidad de modificación, que hacen posibles servicios que funcionan mejor en la Web [15].

Los servicios web REST se utilizan a frecuencia gracias a que permiten una comunicación sencilla con el cliente que los consume, REST está basado en cuatro principios fundamentales que son: la identificación de recursos a través de URI, interfaz uniforme ya que usa métodos estándar como PUT, POST, GET, DELETE; Auto-descriptiva mensajes e Interacciones con estado a través de hipervínculos.

2. DESARROLLO DEL PROYECTO

2.1.-DISEÑO DEL SISTEMA

2.1.1.-DIAGRAMA DE DOMINIO DE CANBAS-CHEAP

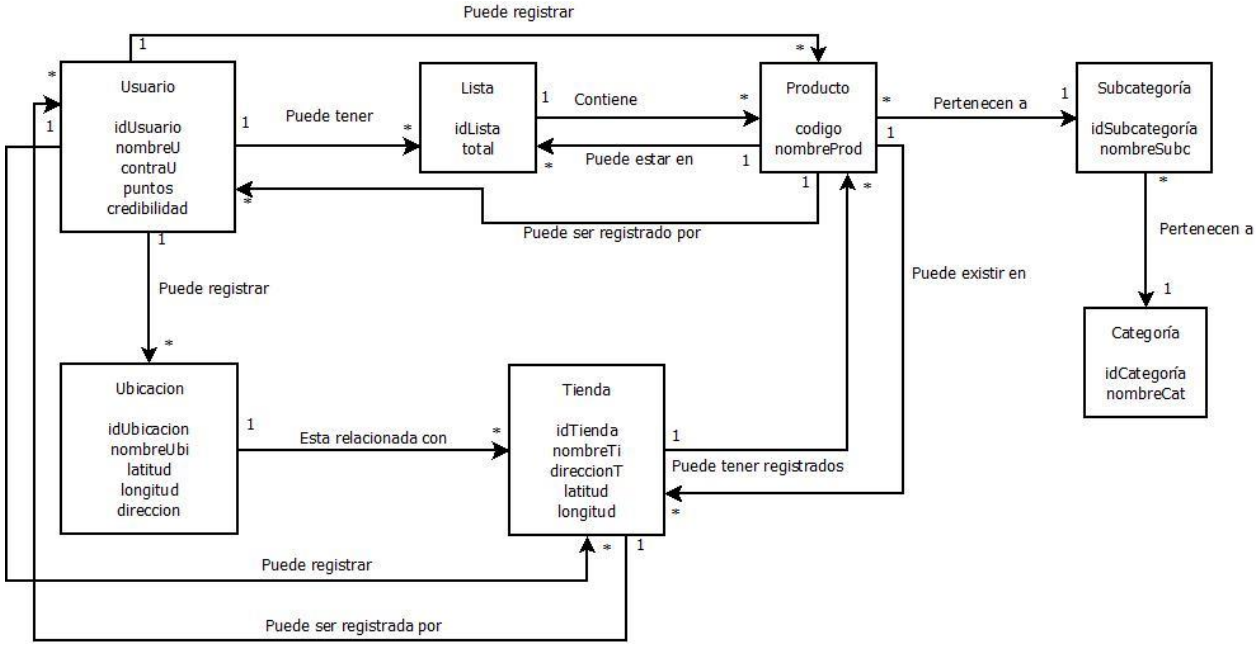


Figura 1 DIAGRAMA DE DOMINIO

El diagrama de dominio (Figura 1) muestra las clases principales del sistema.

A continuación, se explica a grandes rasgos como están relacionadas las entidades que conforman el sistema.

- *Usuario*. Esta clase se utiliza para registrar a la persona que hace uso de la aplicación, permitiéndole a través de la corroboración de sus datos acceder a la aplicación.
- *Ubicación*. Se utiliza para poder hacer el registro de tiendas y para hacer una correcta búsqueda de tiendas cercanas en base a la latitud y longitud de esta.

- ***Producto***. Se utiliza para hacer el registro de productos o búsqueda de canasta básica.
- ***Categoría y Subcategoría.*** Estás clases se utilizan para hacer una clasificación de los productos que el usuario desea registrar o agregar a su búsqueda de canasta básica.
- ***Lista***. Se utiliza para relacionar al usuario con los productos que ha seleccionado en la búsqueda de canasta.

2.1.2.-DIAGRAMA DE CLASES DE CANBAS-CHEAP

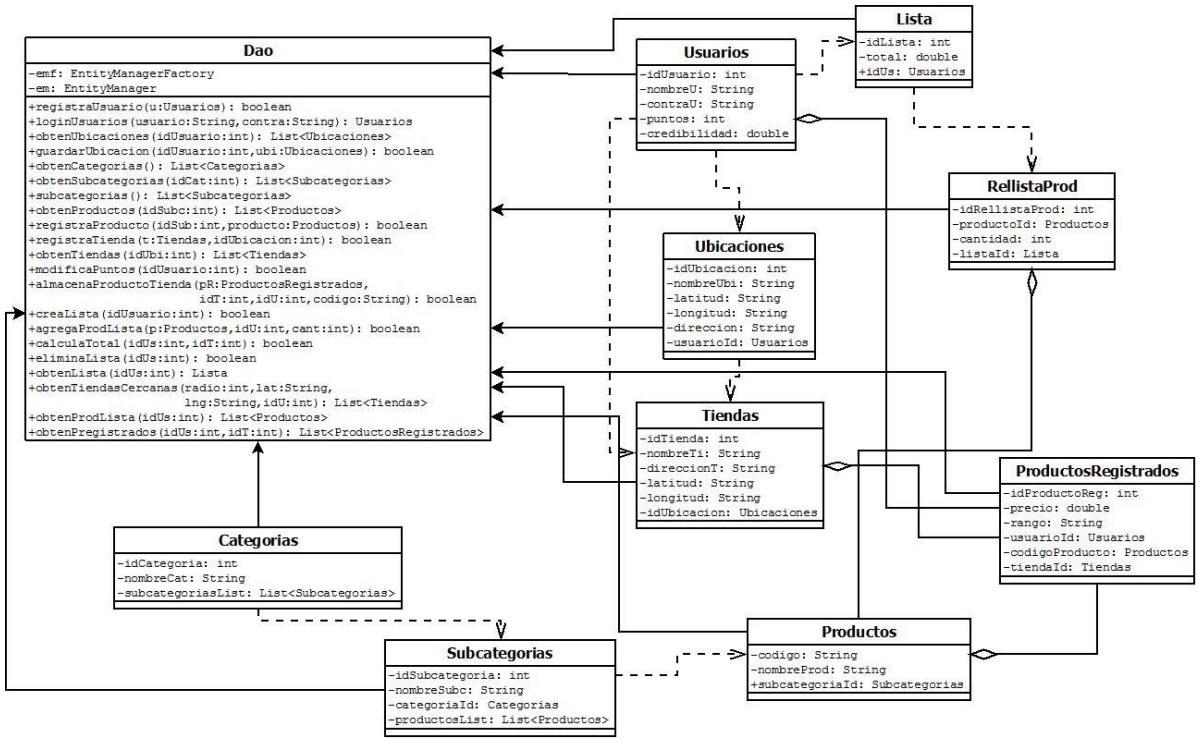


Figura 2 DIAGRAMA DE CLASES

La Figura 2 muestra el diagrama de clases del sistema, en el cuál aparecen las clases entidad, junto con dos clases auxiliares *RellistaProd* y *ProductosRegistrados* además de una clase *Dao.*

- ***RellistaProd.*** Esta clase es la que relaciona una lista perteneciente a cada usuario con los productos seleccionados por este, permitiendo así agregar a ella la cantidad de productos y con ello hacer correctamente el cálculo del total a pagar.
- ***ProductosRegistrados.*** Es la clase que se encarga de relacionar a los usuarios con los productos que ha registrado, proporcionando también la tienda en la que se encuentran y a qué precio.
- Finalmente, la clase ***Dao***, esta se encarga de realizar las consultas, modificaciones y registros a la base de datos necesarios para el funcionamiento del sistema.

2.1.3.-ARQUITECTURA DE CANBAS CHEAP

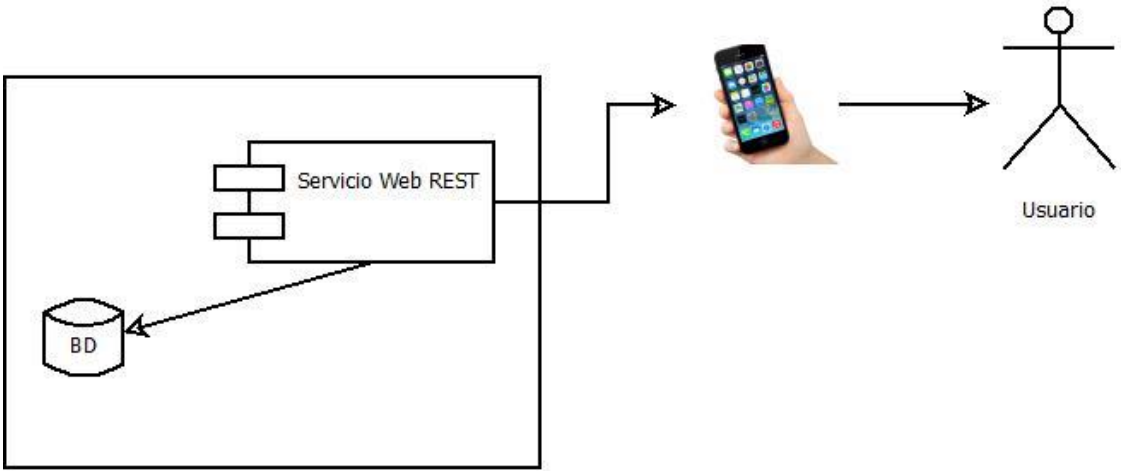


Figura 3 DIAGRAMA DE LA ARQUITECTURA DE CANBAS-CHEAP

Canbas-Cheap cuenta con un Servidor Web que se encarga de recibir y enviar datos a la aplicación cliente, usando persistencia de la información a través de una base de datos (Ver Figura 3).

A continuación, se explica cómo se desarrollaron cada uno de los componentes de CANBAS-CHEAP:

2.2.- HARDWARE

Para el desarrollo de Canbas-Cheap se utilizó una computadora de escritorio con las siguientes características:

- Un procesador Intel® Pentium Dual de 2.00 GHz
- 2 GB de memoria RAM
- Disco duro de 500 GB
- Un monitor de 15.6 pulgadas

2.3.- PERSISTENCIA DE DATOS

Para el correcto funcionamiento de Canbas-Cheap es necesario almacenar la información, por lo que se generó una base de datos la cual está montada en el Servidor de la aplicación, permitiéndole así dar respuesta a las consultas y peticiones que realiza la aplicación cliente.

Software

- MySQL Workbench versión 6.3.

MySQL Workbench es una herramienta que te permite generar y gestionar de manera rápida bases de datos.

2.3.1.-MODELO ENTIDAD-RELACIÓN

El modelo Entidad-Relación de la base de datos se muestra en la Figura 4, en la cual se puede apreciar cómo se encuentran almacenados los datos y como se relacionan las tablas que conforman la base de datos. Su descripción se muestra en la Tabla 1.

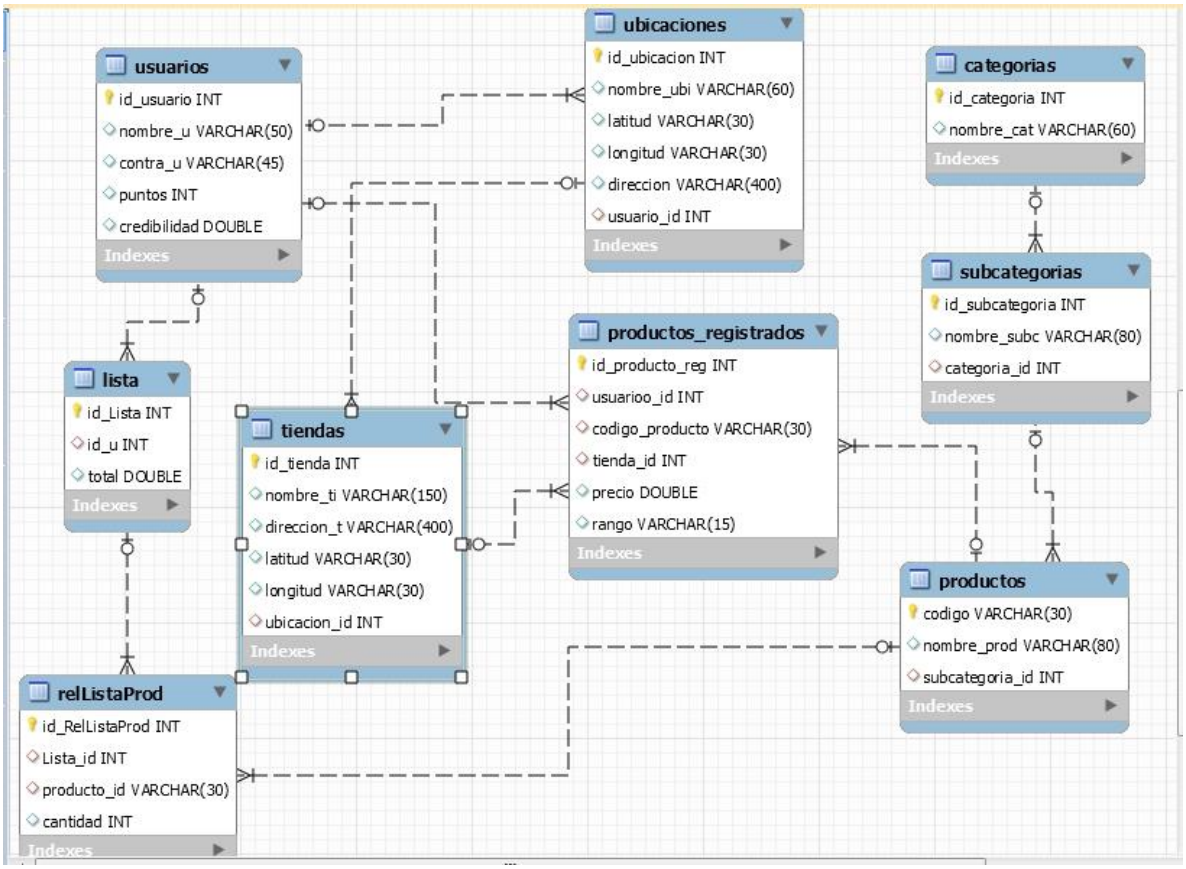


Figura 4 Modelo Entidad-Relación de la base de datos

usuarios	En esta tabla se encuentran almacenados los datos de los usuarios, cuenta con campos como: id_usuario de tipo INT, este campo es utilizado como llave primaria, nombre_u y contra_u de tipo String, puntos de tipo INT y credibilidad de tipo Double.
ubicaciones	Aquí se guarda la información de las ubicaciones que posee cada usuario, para ello cuenta con los siguientes campos: id_Ubicacion de tipo INT, utilizado como llave primaria, nombre_ubi, latitud, longitud y dirección de tipo String, además del campo usuario_id de tipo INT, el cual es utilizado como llave foránea ya que permite

	relacionar a los usuarios con sus ubicaciones registradas.
categorias	Esta tabla almacena las distintas categorías en que se pueden clasificar los productos, cuenta con campos como: Id_categoria de tipo INT, utilizado como llave primaria y nombre_cat de tipo String.
subcategorias	Sirve para almacenar las subcategorías para la correcta clasificación de productos, esta tabla está relacionada con Categorías por lo que posee el campo: categoría_id de tipo INT, el cual es utilizado como llave foránea, cuenta también con los campos: Id_subcategoria de tipo INT, utilizado como llave primaria y nombre_subc de tipo String.
productos	Almacena información relacionada con los productos, por lo que sus campos son: código de tipo String, utilizado como llave primaria, nombre_prod de tipo String y por último el campo subcategoría_id de tipo INT el cual se emplea como llave foránea, permitiendo así clasificar los productos en base a subcategorías.
tiendas	Guarda información de las tiendas que registra el usuario en determinada ubicación, por lo que posee campos como: Id_tienda de tipo INT, utilizado como llave primaria, nombre_ti, dirección_t, latitud, longitud de tipo String y por último ubicación_id de tipo INT, utilizado como llave foránea para la correcta relación de Tiendas-Ubicaciones.
productos_registrados	Relaciona las tablas usuario, tienda y producto, por lo que cuenta con los campos: id_producto_reg de tipo INT, utilizado como llave primaria, usuario_id, código_producto y tienda_id de tipo INT, utilizados como llaves foráneas que hacen referencia a las tablas correspondientes, además de precio de tipo Double y rango de tipo String.
lista	Aquí se almacenan las listas que genera cada usuario, posee los siguientes campos: id_Lista de tipo INT, utilizado como llave primaria, total de tipo Double y por último id_u de tipo INT, utilizado como llave foránea para la relación entre Usuario-Lista.

rellistaprod	Relaciona las tablas Lista-Productos, posee los siguientes campos: id_Rellistaprod de tipo INT, utilizado como llave primaria, Lista_id de tipo INT, producto_id de tipo String, utilizados como llaves foráneas que hacen referencia a las tablas correspondientes y por último Cantidad de tipo INT.

Tabla 1. TABLAS DE LA BASE DE DATOS Y SU DESCRIPCIÓN

2.4.- SERVIDOR WEB

2.4.1.- GESTION DE DATOS

Software

- Hibernate-JPA 2.1.

JPA (Java Persistence API) establece una interfaz común que es implementada por un proveedor de persistencia (TopLink, EclipseLink, Hibernate, entre otros) [16].

Hibernate-JPA, son herramientas que permiten el mapeo-objeto-relacional (ORM), es decir, permiten el mapeo de tablas de una base de datos a objetos en Java y viceversa.

Para el correcto funcionamiento de Hibernate fue necesario agregar las bibliotecas mostradas en la Figura 5.

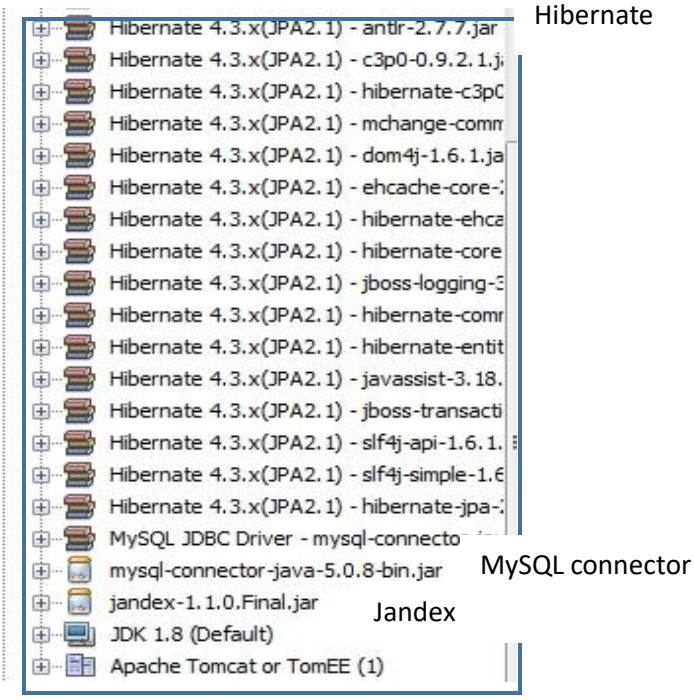


Figura 5 Bibliotecas necesarias para el uso de Hibernate

- MySQL Connector es necesario para la correcta comunicación con la base de datos.
- Jandex-1.1.0, si no se tiene agregada esta biblioteca puede ocurrir que se muestre un error al momento de utilizar Hibernate-JPA por lo que es necesario descargarla y añadir el jar correspondiente.
  Esta biblioteca es utilizada para la generación de metadatos.

Para realizar el mapeo de tablas de la base de datos a objetos entidad fue necesario primero definir una unidad de persistencia (PU), ya que gracias a ella se permite la comunicación con la base de datos y con ello la persistencia de la información.

Una vez creada la unidad de persistencia, se procedió a crear las clases entidad desde la base de datos, estas clases poseen como atributos los campos de sus respectivas tablas, además de sus métodos *set*, *get* y otros implementados por el API JPA.

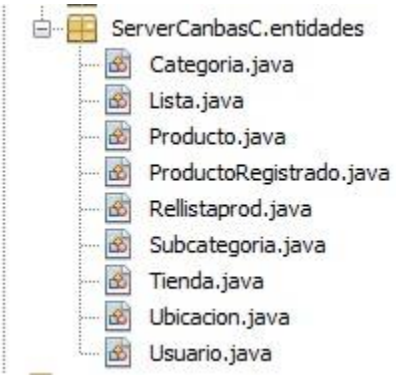


Figura 6 Clases entidad generadas a partir de la base de datos

Estas clases entidad son utilizadas por la clase Dao mostrada en la Figura 7, la cual se encarga de realizar todas las funciones que proporcionan los servicios.

La clase Dao realiza la transacción de la información entre la base de datos y el servidor.

Dao

```
-emf: EntityManagerFactory
-em: EntityManager
+registraUsuario(u:Usuarios): boolean
+loginUsuarios(usuario:String,contra:String): Usuarios
+obtenUbicaciones(idUsuario:int): List<Ubicaciones>
+guardarUbicacion(idUsuario:int,ubi:Ubicaciones): boolean
+obtenCategorias(): List<Categorias>
+obtenSubcategorias(idCat:int): List<Subcategorias>
+subcategorias(): List<Subcategorias>
+obtenProductos(idSubc:int): List<Productos>
+registraProducto(idSub:int,producto:Productos): boolean
+registraTienda(t:Tiendas,idUbicacion:int): boolean
+obtenTiendas(idUbi:int): List<Tiendas>
+modificaPuntos(idUsuario:int): boolean
+almacenaProductoTienda(pR:ProductosRegistrados,
                        idT:int,idU:int,codigo:String): boolean
+creaLista(idUsuario:int): boolean
+agregaProdLista(p:Productos,idU:int,cant:int): boolean
+calculaTotal(idUs:int,idT:int): boolean
+eliminaLista(idUs:int): boolean
+obtenLista(idUs:int): Lista
+obtenTiendasCercanas(radio:int,lat:String,
                      lng:String,idU:int): List<Tiendas>
+obtenProdLista(idUs:int): List<Productos>
+obtenPregistrados(idUs:int,idT:int): List<ProductosRegistrados>
```

Figura 7 Clase Dao

En la clase Dao se define el atributo *EntityManagerFactory* el cuál se encarga de gestionar las clases entidad que define la unidad de persistencia, para llevar a cabo esta funcionalidad se hace uso del atributo *EntityManager.*

A continuación, se explican los métodos que conforman la clase Dao.

- registraUsuario: Este método recibe como parámetro un objeto de tipo usuario que será registrado en la base de datos, si el registro es correcto devuelve un boolean con valor *true,* de lo contrario el valor devuelto es *false.*
- loginUsuarios: Este método recibe como parámetros dos String, uno con el nombre del usuario y otro con la contraseña del mismo, hace la búsqueda en la base de datos de algún registro de usuario que coincida con estos valores y si la búsqueda es exitosa se obtienen los datos de este usuario para posteriormente ser retornados en un objeto de tipo usuario.
- obtenUbicaciones: Este método recibe como parámetro el id de usuario de tipo entero, este es utilizado para corroborar que exista un usuario con ese id, si es así, se procede con la recuperación de ubicaciones registradas por este usuario, las cuales serán devueltas en un List<Ubicaciones>.
- guardarUbicacion: Este método recibe como parámetro el id del usuario de tipo entero y el objeto de tipo ubicación que se desea registrar, para ello primero se busca al usuario con ese id, si existe se procede con el registro, si este es exitoso devuelve un boolean con valor *true*, de lo contrario el valor devuelto es *false*.
- obtenCategorias: Este método no recibe ningún parámetro ya que solo se encarga de obtener todas las categorías registradas en la base de datos y retornarlas en un List<Categorias>
- obtenSubcategorias: Este método recibe como parámetro el id de categoría de tipo entero y en base a este se obtienen las subcategorías, para esto primero se busca la categoría con este id, si existe se procede a buscar las subcategorías que estén relacionadas con esta, para posteriormente ser retornadas en un List<Subcategoria>.
- Subcategorías: Este método no recibe ningún parámetro ya que se encarga de obtener todas las subcategorías registradas en la base de datos y retornarlas en un List<Subcategorias>.
- obtenProductos: Este método recibe como parámetro el id de subcategoría de tipo entero y en base a este se obtienen los productos registrados relacionados con esta subcategoría, para esto primero se busca la subcategoría con este id, si existe se procede a buscar los productos y posteriormente retornarlos en un List<productos>.

- registraProducto: Este método recibe como parámetros el id de subcategoría de tipo entero y el objeto de tipo producto que será registrado, para hacer el registro primero se busca en la base de datos la existencia de la subcategoría con ese id y posteriormente se procede a hacer el registro del producto relacionándolo con dicha subcategoría, si el registro es existoso se devuelve un boolean con valor *true*, de lo contrario el valor devuelto es *false*.
- registraTienda: Este método recibe como parámetros el id de ubicación de tipo entero y el objeto de tipo tienda que será registrado, para ello primero se verifica que exista un registro de ubicación con este id, si es así se hace el correspondiente registro de la tienda, si el resultado es exitoso se devuelve un boolean con valor *true*, en caso contrario el valor devuelto es *false*.
- obtenTiendas: Este método recibe como parámetro el id de ubicación y en base a ello se obtienen las tiendas relacionadas con esta ubicación, para esto primero se hace la búsqueda de una ubicación con este id y posteriormente se obtienen las tiendas relacionadas a esta, estas son devuelvas por un List<Tienda>.
- modificaPuntos: Este método recibe como parámetro el id de usuario de tipo entero, para ello primero se busca en la base de datos al usuario con este id, posteriormente se procede a sumar diez puntos a los ya registrados y en base a estos puntos se calcula la credibilidad del usuario, se realizan los cambios y son enviados a la base de datos para su correcta persistencia, si el proceso fue exitoso se devuelve un boolean con valor *true,* en caso contrario el valor devuelto será *false.*

  La clase Credibilidad se muestra en la Figura 8, se encuentra en el paquete *ServerCanbasC.Util,* ya que no forma parte de las clases entidad y solo es utilizada para el cálculo correcto de la credibilidad.

Figura 8 Clase Credibilidad

El método *CalculaCredibilidad* recibe como parámetro un número entero que representa los puntos que tiene registrados el usuario en la base de datos y aplica una regla de tres para obtener el porcentaje de credibilidad y el cual es retornado por este método.

Se toman quinientos puntos que equivalen al cien por ciento de credibilidad, cada que el usuario registra productos o realiza una

búsqueda de canasta se suman diez puntos y se procede a hacer el cálculo de la credibilidad.

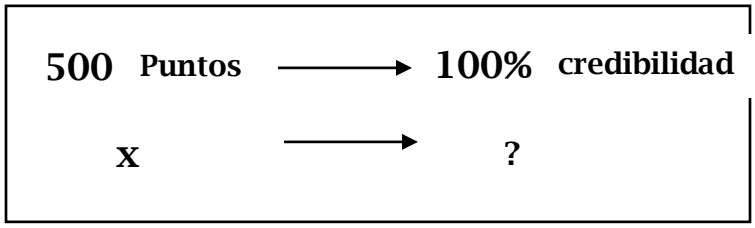


Figura 9 Regla de 3 para obtener credibilidad

- almacenaProductoTienda: Este método recibe como parámetros un objeto de tipo ProductosRegistrados el cual tiene el precio del producto, el id de la tienda e id de usuario ambos de tipo entero y código de tipo String, para su correcto funcionamiento primero se busca en la base de datos al usuario con ese id, la tienda con ese id y el producto con ese código, posteriormente estos son fijados por medio de los métodos set del objeto productosRegistrados, después verifica que no se tenga algún registro de este producto en la misma tienda, si no hay dicho registro se fija el valor de rango en *'Medio'*, de lo contrario se comparan los precios de los productos encontrados y en base a ello se fija el nuevo rango del producto a registrar, ya hecho esto se procede a hacer el registro de este producto, si este fue exitoso se devuelve un boolean con valor *true*, de lo contrario el valor devuelto será *false*.
- creaLista: Este método recibe como parámetro el id de usuario de tipo entero, se procede a corroborar que exista un registro de usuario con este id y si se encuentra se crea un objeto de tipo Lista fijando por medio del método set al usuario que registra dicha lista, estos datos son enviados a la base de datos para su correcto registro, si el resultado es exitoso se devuelve un boolean con valor *true*, de lo contrario el valor devuelvo será *false*.
- agregaProdLista: Este método recibe como parámetros un objeto de tipo Productos, el id de usuario y cantidad ambos de tipo entero, se procede a corroborar que exista un registro de usuario con este id, si es así se obtiene la lista registrada por este, posteriormente se crea un objeto de tipo relListaProd y se fija la lista previamente recuperada, el producto y la cantidad de este y se realiza el registro, si este fue exitoso se devuelve un boolean con valor *true*, de lo contrario el valor devuelto será *false*.
- calculaTotal: Este método recibe como parámetros el id de usuario y el id de tienda, ambos de tipo entero, se procede a corroborar que exista un registro de usuario con este id, si es así se obtiene la lista

registrada por este, con ello se obtienen los productos registrados en relListaProd y se buscan en los registros de la base de datos aquellos que estén en la tienda con ese id y tengan el rango *Medio*, una vez hecho esto se obtienen los precios, los cuales son sumados uno a uno y en base a ello se obtiene el total el cual es fijado en el objeto de tipo lista por medio del set y se persisten los cambios, si este proceso fue exitoso se devuelve un boolean con valor *true*, de lo contrario el valor devuelto será *false*.

- eliminaLista: Este método recibe como parámetro el id de usuario de tipo entero, se procede a corroborar que exista un registro de usuario con ese id, si es así se obtiene la lista registrada para posteriormente ser eliminada, si este proceso fue exitoso se devuelve un boolean con valor true, de lo contrario el valor devuelto será false.
- obtenLista: Este método recibe como parámetro el id de usuario de tipo entero, se procede a corroborar que exista un registro de usuario con ese id, si es así se obtiene esta y es devuelta por el objeto de tipo Lista.
- obtenTiendasCercanas: Este método recibe como parámetros: radio de tipo entero, latitud y longitud de tipo String y el id de usuario de tipo entero, para el funcionamiento de este método primero se busca en la base de datos un registro de usuario con este id, si existe se procede a la obtención de la lista registrada por este y en base a ella se obtienen los registros de relListaProd para guardar en un List<Productos> los productos encontrados en esta, posteriormente se crea una instancia de la clase Distancia mostrada en la Figura 10.

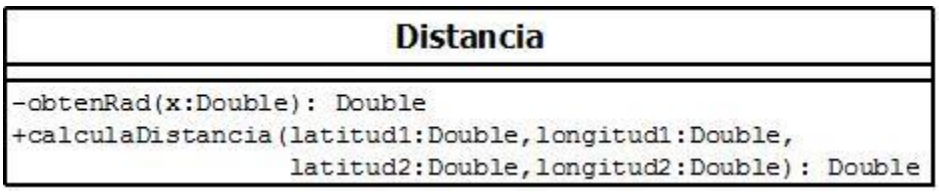


Figura 10 Clase Distancia

Se obtienen los registros de tiendas de la base de datos y en base a la latitud y longitud proporcionados y la latitud y longitud de cada tienda se aplica el método de CalculaDistancia el cual hace uso de la fórmula de Haversine previamente explicada en la sección 1.4.2 del Marco Teórico, se compara el resultado de este método con el radio proporcionado y si el resultado es menor o igual a este se agrega esta tienda a un List<tiendas>. Una vez recuperadas las tiendas cercanas se procede a la obtención de los productos registrados en cada tienda

los cuales se almacenan en un List<Productos>, se crea una instancia de la clase ComparaList mostrada en la Figura 11.

Figura 11 Clase ComparaList

Esta clase compara dos List<Productos>, pSu que hace referencia a los productos seleccionados por Usuario y peT productos en tienda, para ello hace una comparación uno a uno de cada uno de los elementos de pSu en peT, si todos los productos en pSu están en peT el método compara devuelve un boolean con valor *true*, en caso contrario devuelve un *false*.
El método obtenTiendasCercanas corrobora que el resultado sea true y almacena la tienda que tiene todos los productos en un nuevo List<Tiendas> tiendasc el cual es devuelto por este método.

- obtenProdLista: Este método recibe como parámetro el id de usuario de tipo entero, se procede a corroborar que se tenga un registro de usuario con este id, si es así se obtiene la Lista registrada por este usuario y en base a ello se obtiene relListaProd y se almacenan en un List<Productos> los productos obtenidos, este List<Productos> es el devuelto por este método.
- obtenPregistrados: Este método recibe como parámetros el id de usuario y el id de tienda de tipo entero, se hace la búsqueda en la base de datos de aquellos productos Registrados que coincidan con el id de usuario y tienda proporcionados, estos son almacenados en un List<ProductosRegistrados> el cual es devuelto por este método.

2.4.2.- SERVICIOS WEB

El intercambio de información entre el servidor web y la aplicación cliente se realiza a través del protocolo REST.

Software

La aplicación Servidor se programó en su totalidad en lenguaje Java.

- Apache Tomcat versión 8 [17] para el despliegue del servidor.

Entorno de Desarrollo

- Netbeans IDE 8.2 [18]

Creación de Servicios

Para la creación de los servicios se debe hacer click derecho en el proyecto de la aplicación servidor -> Nuevo -> Other-> WebServices -> RESTful Web Services from Patterns, ver Figuras 12 a 14.

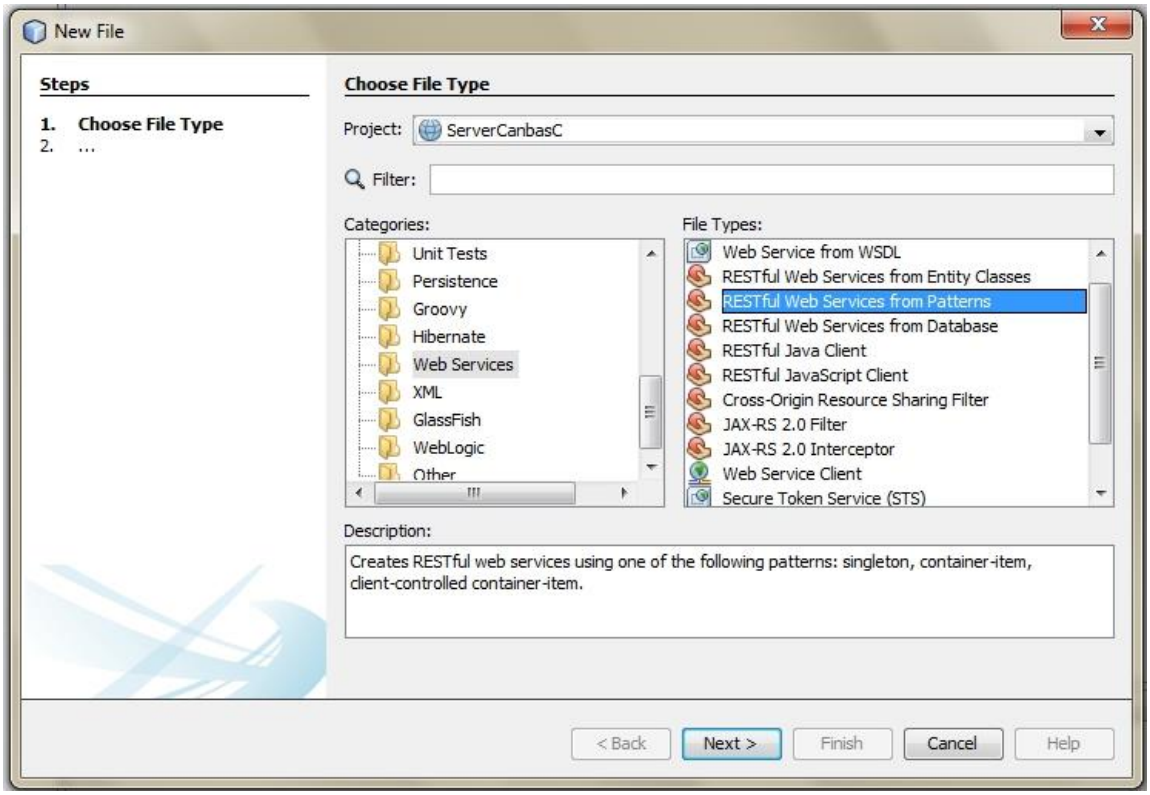


Figura 12 Creación de Servicios REST Paso 1

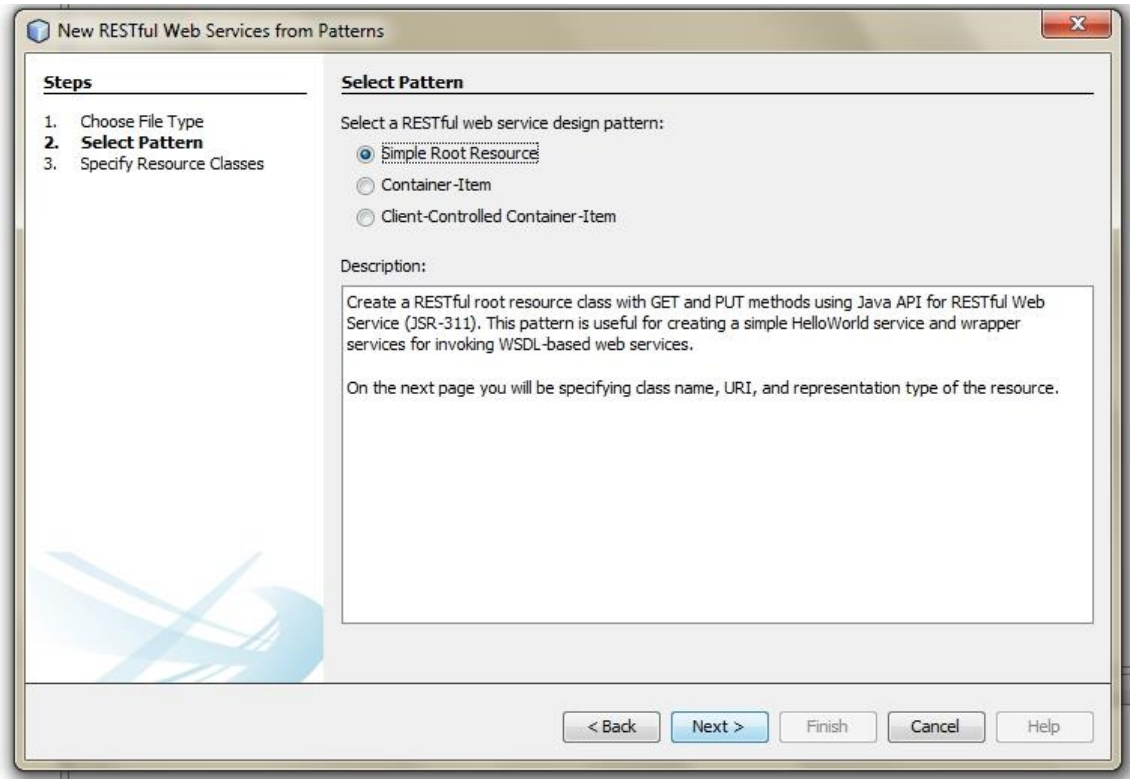


Figura 13 Creación de Servicios REST Paso 2

Por último se escribe un nombre de Path y de clase para este Servicio.

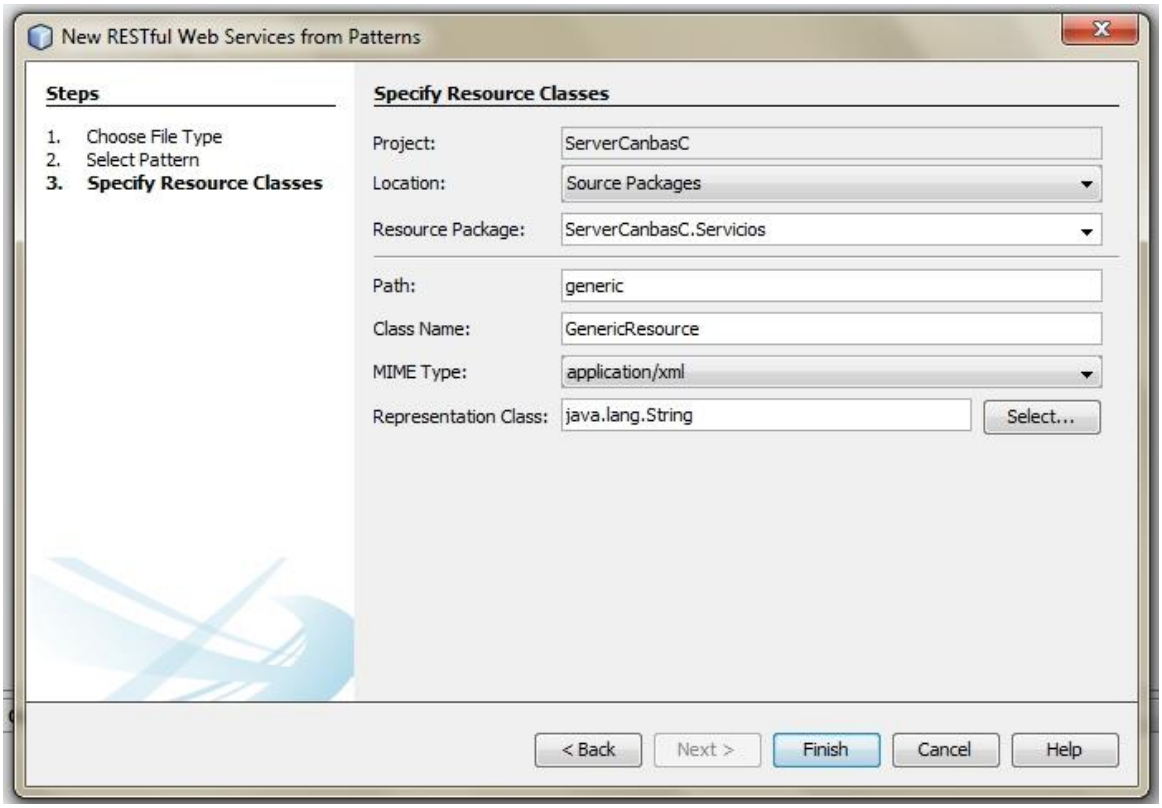


Figura 14 Creación de Servicios REST Paso 3

Una vez hecho esto se genera automáticamente una clase llamada *ApplicationConfig* la cual hace referencia a las clases que generan los servicios para el correcto despliegue de estos.

Para generar una clase de servicio, se crea una clase Java y se agregan las líneas de código mostradas en la Figura 15, la clase ApplicationConfig reconoce automáticamente estás clases y las agrega en su método AddRestResourcesClass ver Figura 16.

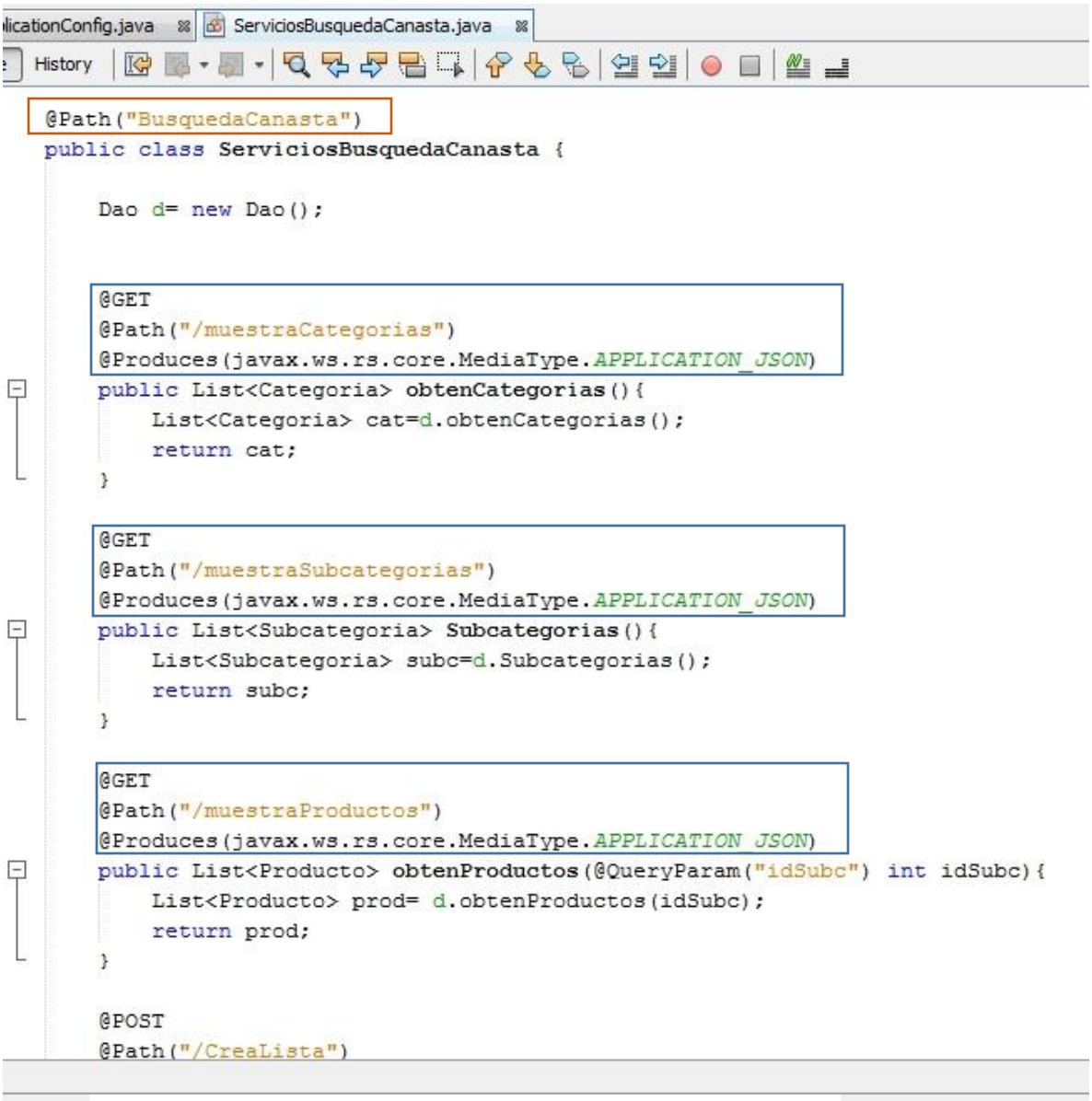


Figura 15 Ejemplo de una clase de Servicio

El primer *Path* hace referencia a la ruta de acceso relativa al servicio.

Para definir las funciones que tiene el Servicio se agregó a cada una la notación de una de las cuatro funciones: *@Get, @Post, @Put ó @Delete*, junto con su respectivo *Path* el cual, hace referencia al método al que se desea invocar en este servicio y por último la notación *@Produces* en caso de que el método devuelva algún objeto, ó *@Consumes* en caso de que se desee enviar un objeto o dato al servidor.

```
30      */
31      private void addRestResourceClasses(Set<Class<?>> resources) {
32          resources.add(ServerCanbasC.Servicios.ServiciosBusquedaCanasta.class);
33          resources.add(ServerCanbasC.Servicios.ServiciosRegProductos.class);
34          resources.add(ServerCanbasC.Servicios.ServiciosUsuario.class);
35      }
36
37  }
38
```

Figura 16 Código de la clase ApplicationConfig en que se agregan las clases Servicio

Paquetes

El servidor web está formado por cuatro paquetes ver Figura 17.

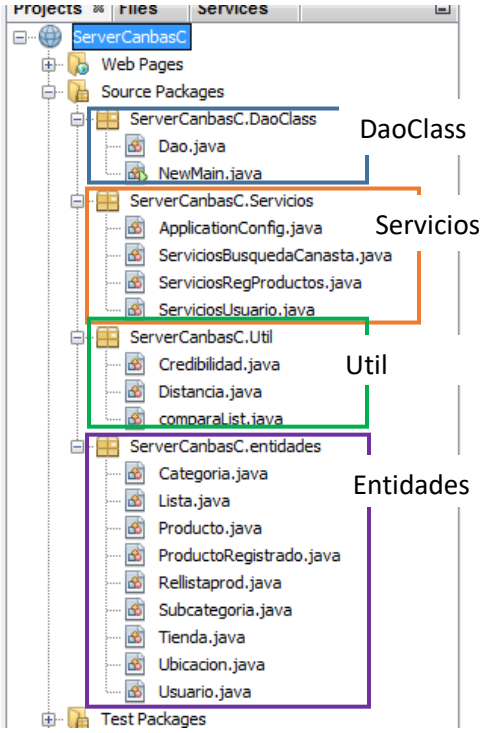


Figura 17 PAQUETES QUE CONFORMAN EL SERVICIO WEB

A continuación, se explica el contenido de cada uno de los paquetes que conforman la aplicación del servidor.

ServerCanbasC.DaoClass: Este paquete aloja la clase responsable de establecer conexión con la base de datos (Dao) explicada previamente en el apartado 2.4.1 de esta sección. Este paquete también tiene una clase Main, la cual se utilizó solamente para realizar pruebas a cada método de la aplicación y con ello corroborar su funcionamiento.

ServerCanbasC.Servicios: En este paquete se encuentran los servicios de los que puede disponer la aplicación cliente.

Los servicios fueron clasificados de acuerdo a las funciones más importantes que brinda cada uno, estos servicios hacen uso de los métodos de la clase Dao para dar espuestas a las consultas y peticiones de la aplicación cliente.

- *ServiciosBusquedaCanasta:* Este es el servicio que se encarga de realizar las tareas correspondientes a la búsqueda de Canasta por parte del usuario, por lo que hace uso de los siguientes métodos: obtenCategorias, Subcategorias, obtenProductos, creaLista, agregaProdLista, obtenTiendasCercanas, obtenProductosLista, obtenProductosRegistrados, calculaTotal y obtenLista.
- *ServiciosRegProductos:* Este es el servicio que se encarga de realizar las tareas correspondientes al registro de productos, por lo que hace uso de los siguientes métodos: obtenCategorias, obtenSubcategorias, registraProducto, registraTienda, almacenaProductoTienda y obtenTiendas.
- *ServiciosUsuario:* En este servicio se encuentran las operaciones encargadas de gestionar los datos del usuario, por lo que hace uso de los siguientes métodos: registraUsuario, loginUsuarios, guardarUbicacion, obtenUbicaciones y modificaPuntos.

ServerCanbasC.Util: En este paquete se encuentran las clases que se utilizan para ayudar a la clase Dao con las funciones requeridas (Credibilidad, ComparaList, Distancia).

El funcionamiento de estas clases se explicó previamente en el apartado 2.4.1 de esta sección.

***ServerCanbasC.entidades*:** En este paquete se encuentran las clases entidad que fueron mapeadas con Hibernate a través de la unidad de persistencia, por lo que sus atributos coinciden con los campos que tiene cada tabla en la base de datos.

- *Categoría:* Esta clase es utilizada para la correcta clasificación de productos.
- *Lista:* Esta clase es utilizada para representar la lista que está creando el usuario cada vez que realiza una búsqueda de canasta.
- *Producto:* Esta clase hace referencia al producto que se quiere registrar o consultar.
- *ProductoRegistrado:* Esta clase es la que se encarga de relacionar al producto registrado con el usuario que lo registra y la tienda en que se encuentra, además de permitirle al usuario introducir el precio del producto en dicha tienda.

- *RellistaProd:* Esta clase relaciona al usuario con su respectiva lista y con los productos que ha seleccionado.
- *Subcategoría:* Esta clase es utilizada para clasificar a los productos en base a subcategorías, las cuáles están clasificadas por categorías.
- *Tienda:* Esta clase hace referencia a la tienda que se quiere registrar o consultar.
- *Ubicación:* Esta clase hace referencia a la ubicación en la que se encuentra el usuario.
- *Usuario:* Esta clase hace referencia al usuario que está utilizando la aplicación.

2.5.- APLICACIÓN CLIENTE

La aplicación cliente se desarrolló haciendo uso de Java y JavaFX como lenguajes de programación. A continuación, se menciona el software y el entorno de desarrollo utilizado para la creación de esta.

Software

- Gluon Mobile [19]

La aplicación cliente fue planeada tomando como software para su desarrollo PhoneGap[20], se optó por cambiarlo por Gluon Mobile[19], puesto que Gluon trabaja con Java lo que lo hace un software más robusto.

Gluon Mobile[19] provee las herramientas necesarias para la creación de una aplicación Cliente, permite el desarrollo de un solo código que es funcional para dispositivos Android, Ios y aplicaciones de escritorio, con solo modificar los archivos de configuración de la plataforma en que se desea probar la aplicación.

Para las pruebas en dispositivos móviles se debe hacer uso de:

- ***Android***. Android SDK Manager, descargar la última versión de Build-Tools (23.0.1), SDK Platform API 21 y de Extras: Android Support Respository.
- ***Ios.*** Una Mac con MacOS X 10.11.5 o superior y Xcode 8.x o superior.

- Gradle versión 1.3.9.2.

Gluon Mobile[19] hace uso de Gradle para la construcción de las aplicaciones ya que permite añadir las dependencias que estas requieren para su funcionamiento.

- GMapsFX [21] versión 2.10.0.

GMapsFx es un wrapper del API Google Maps, en el que el código utilizado es Java y no JavaScript como en el API original.

GMapsFX no tiene aún todas las funciones de las distintas API's que proporciona Google Maps pero si cuenta con: Geocoding, Directions y Elevation.

Entorno de desarrollo

- Netbeans 8.2 [18], agregando los plugins de Gluon Mobile en su versión más reciente (2.5.0) y Gradle versión 1.3.9.2.
- SceneBuilder [22] 8.2.0.

Scene Builder [22] es una herramienta que permite la creación de interfaces de usuario de una manera sencilla.

A SceneBuilder se agregó el jar correspondiente a GMapsFX en su versión 2.10.0 para las vistas que utilizan Google Maps.

A continuación, se explica la estructura de la aplicación Cliente.

2.5.1.-ESTRUCTURA DE LA APLICACIÓN CLIENTE

Puesto que Gluon Mobile permite la creación de aplicaciones multiplataforma (Android, iOS, Escritorio), genera carpetas para cada una de estas permitiendo que se agregue código nativo si se requiere a la aplicación, ver Figura 18.

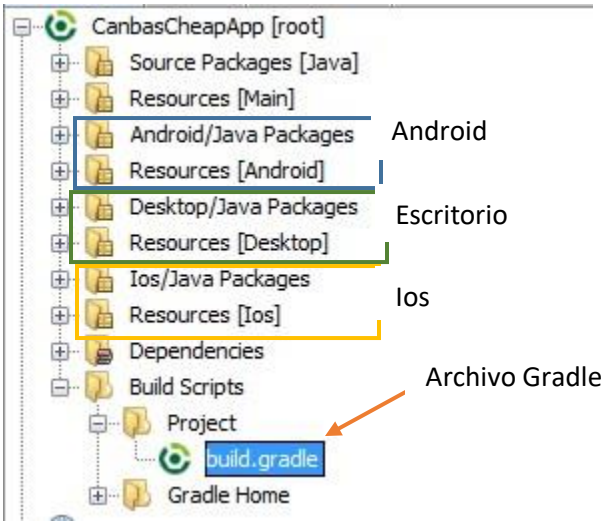


Figura 18 Estructura de la aplicación Cliente

Gluon Mobile cuenta con un componente llamado Gluon Charm Down [19] el cual, en tiempo de ejecución hace que se genere el código de la plataforma específica para proveer la funcionalidad correcta, por lo que en el presente

proyecto no fue necesario añadir código nativo a alguna carpeta de plataforma específica.

Es importante mencionar que cada aplicación generada con Gluon Mobile [19] posee un archivo *build.gradle* el cual permite la correcta construcción de la aplicación, descargando las dependencias que esta requiere.

De manera inicial el archivo *build.gradle* es generado automáticamente, este puede ser editado para agregar las dependencias requeridas.

Para hacer uso de GMapsFX [21] fue necesario editar este archivo, el cual se encuentra en la carpeta *BuildScripts*, ver Figura 18.

En el archivo *build.gradle* se añadió la línea de código mostrada en la Figura 19.

```
buildscript {
    repositories {
        jcenter()
    }
    dependencies {
        classpath 'org.javafxports:jfxmobile-plugin:1.3.0'
    }
}

apply plugin: 'org.javafxports.jfxmobile'

repositories {
    jcenter()
    maven {
        url 'http://nexus.gluonhq.com/nexus/content/repositories/releases'
    }
}

mainClassName = 'com.canbascheapapp.CanbasCheapApp'

dependencies {
    compile 'com.gluonhq:charm:4.3.0'
    compile group: 'com.lynden', name: 'GMapsFX', version: '2.10.0'
    compileNoRetrolambda 'com.airhacks:afterburner.mfx:1.6.3'
}

jfxmobile {
    javacEncoding='utf-8'
    downConfig {
        version = '3.2.0'
        // Do not edit the line below. Use Gluon Mobile Settings in your project context menu instead
        plugins 'barcode-scan', 'display', 'lifecycle', 'position', 'statusbar', 'storage'
```

GMapsFx

Figura 19 Archivo build.gradle

A continuación, se explican los paquetes que conforman la aplicación Cliente, ver Figura 20.

Paquetes

La aplicación cliente hace uso del framework Afterburner [23] por lo que funciona bajo la arquitectura Modelo-Vista-Presentador.

Cabe señalar que el cliente es pesado, ya que al momento de hacer las peticiones al servicio REST es necesario declarar el tipo de dato que será

enviado o recibido, por lo que se optó por crear en el cliente las clases entidad necesarias para el correcto intercambio de información el cuál se hace en formato JSON.

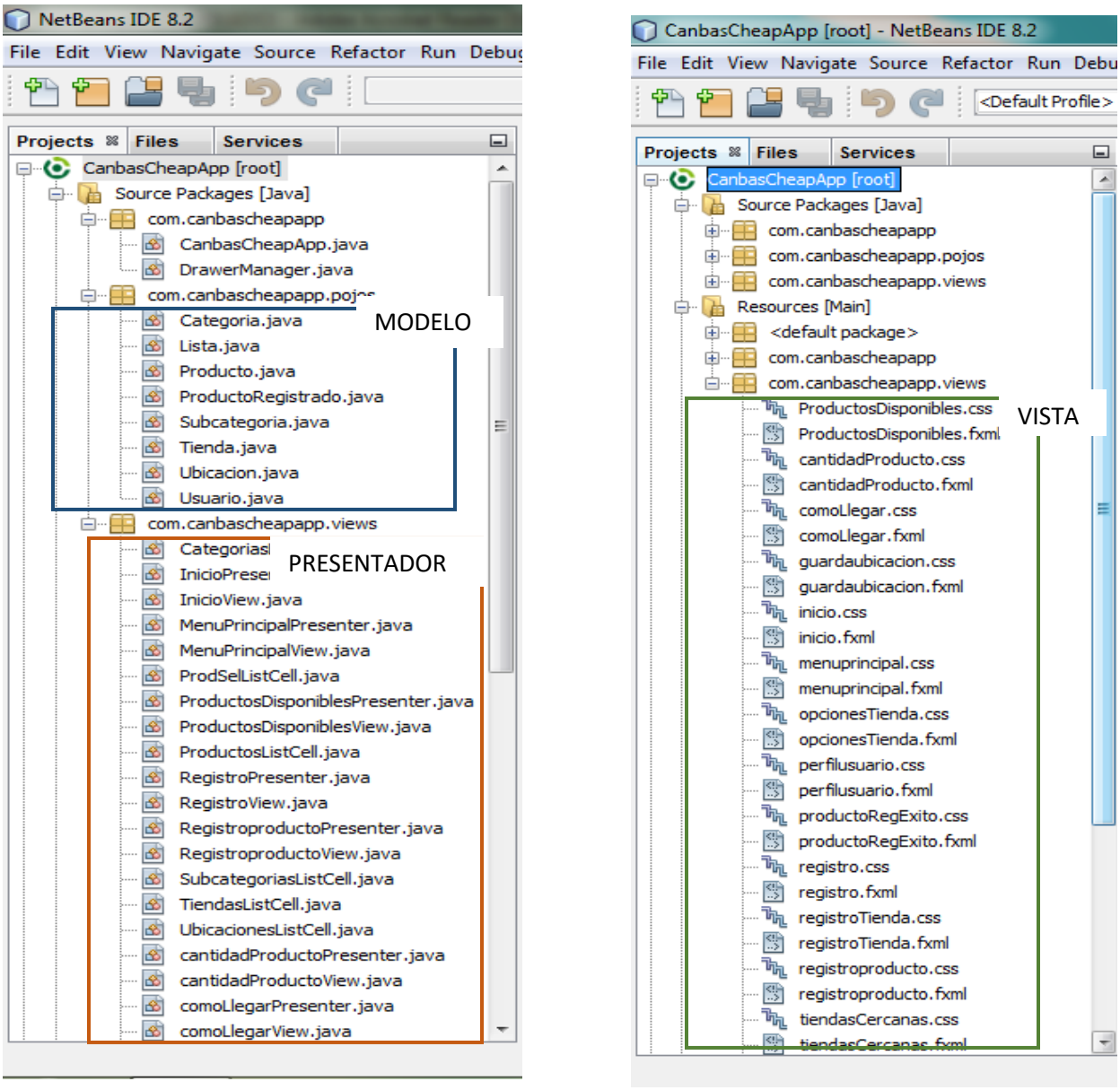


Figura 20 Paquetes de la aplicación Cliente

MODELO. Compuesto por las clases entidad necesarias para el intercambio de información con el servidor.

Cabe señalar que estas clases entidad deben tener los mismos atributos que en la aplicación del servidor, es decir, deben tener el mismo nombre en cada atributo.

VISTA. Compuesto por archivos FXML y CSS que son los que definen la interfaz de usuario.

Para la creación de las vistas se hizo uso de SceneBuilder [22] ya que permite generar vistas de manera rápida y sencilla sin la necesidad de codificar cada elemento a mostrar.

PRESENTADOR. Aquí se alojan clases Java con sus respectivas vistas y es aquí donde se definen las funciones que puede llevar a cabo la vista, además de hacer las solicitudes al servidor.

2.5.2.- FUNCIONAMIENTO

Para explicar el funcionamiento de la aplicación CANBAS-CHEAP se hará uso del diagrama de Casos de Uso General mostrado en la Figura 21.

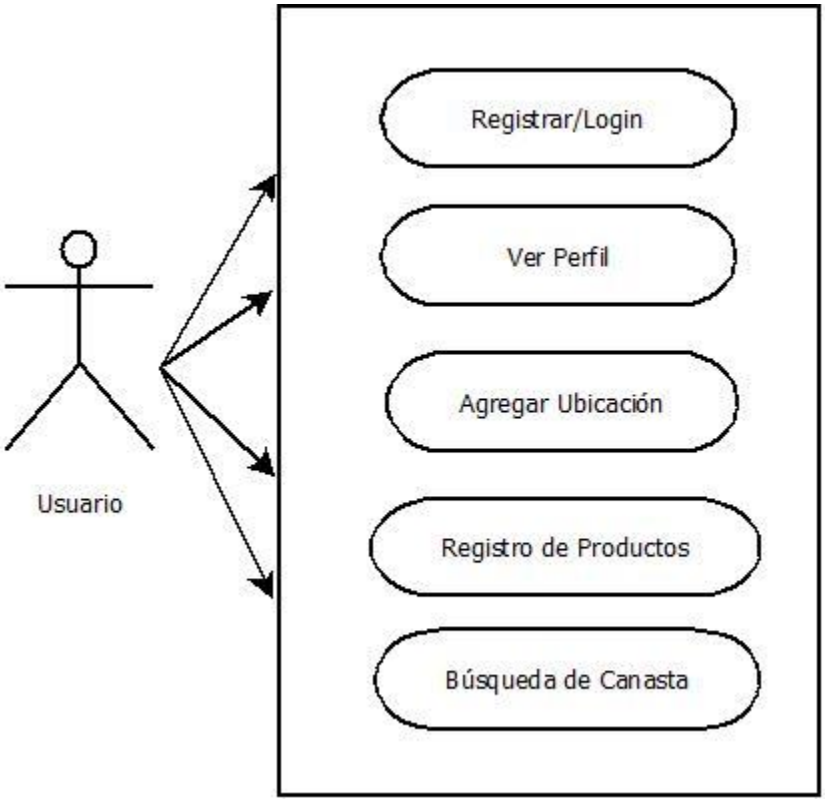


Figura 211 DIAGRAMA DE CASOS DE USO GENERAL

El actor primario es el "usuario", que interactúa todo el tiempo con la aplicación, se encarga de registrar productos, sus ubicaciones o buscar su canasta básica, para ello debe primero registrarse y posteriormente acceder a la aplicación.

Canbas-Cheap consta de cinco módulos como puede observarse en la Figura 21, estos módulos a su vez están compuestos de sub-módulos para realizar tareas particulares.

A continuación, se describe el desarrollo de las interfaces y clases que usa cada módulo:

- **Registrar/Login:** Este módulo le permite al usuario registrarse en la aplicación, donde podrá acceder a la pantalla de Login y si sus datos son correctos esta le permitirá acceder a sus funciones.
- **Ver Perfil:** Este módulo permite al usuario consultar su información y elegir una ubicación para la búsqueda de canasta o registro de productos.
- **Registrar Producto:** Este módulo permite al usuario registrar un producto, tomando en consideración la tienda en que se encuentra dicho producto y el precio, para ello el usuario debe tener al menos una tienda registrada, en caso contrario deberá acceder al registro de la misma.
- **Buscar Canasta Básica:** En este módulo el usuario puede seleccionar los productos que desea adquirir, proporcionar la cantidad de ellos, seleccionar el radio de distancia para la búsqueda de tiendas que dispongan de dichos productos y seleccionar de entre las tiendas cercanas la que le parezca mejor opción.
- **Agregar Ubicación:** Este módulo permite al usuario agregar diferentes ubicaciones, estás son consideradas al momento de buscar las tiendas cercanas y al momento de registrar tiendas.

Creación de vistas

El proceso para la creación de una vista es el mismo para todas.

A continuación, el proceso:

Primero se creó un archivo. fxml en la Carpeta *Resources [Main]*, paquete CanbasCheapApp.*views,* ver Figuras 22, 23 y 24.

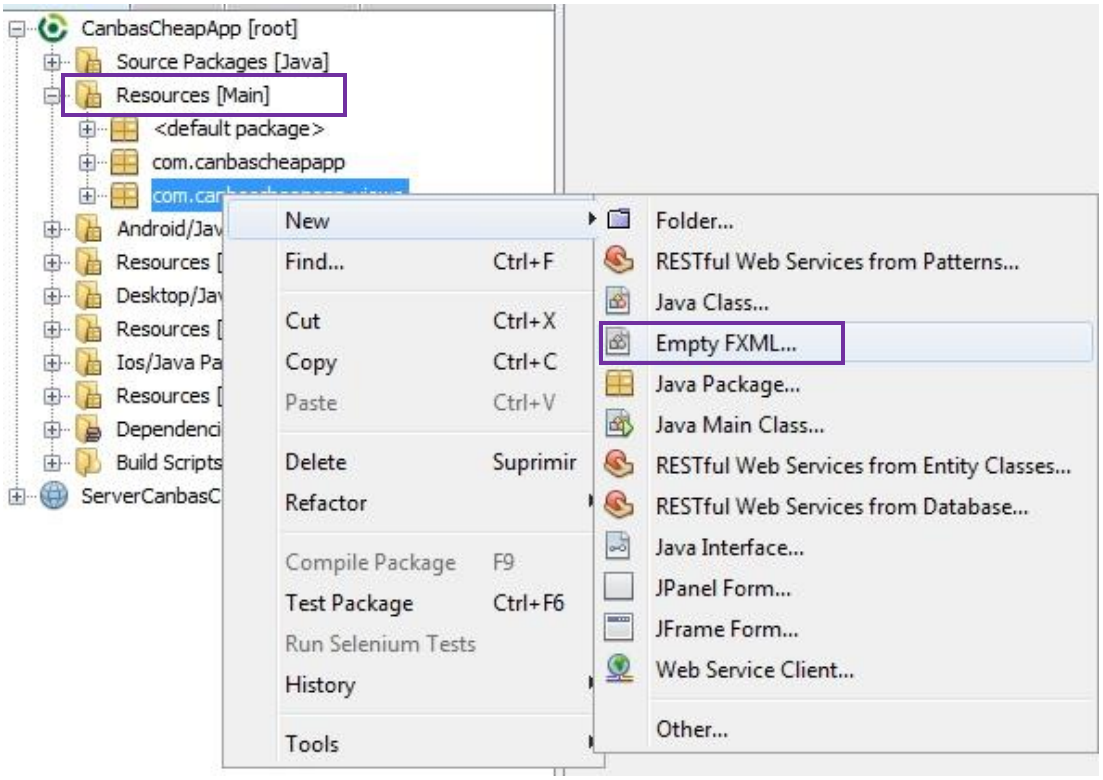


Figura 222 Creación de un archivo FXML Paso 1

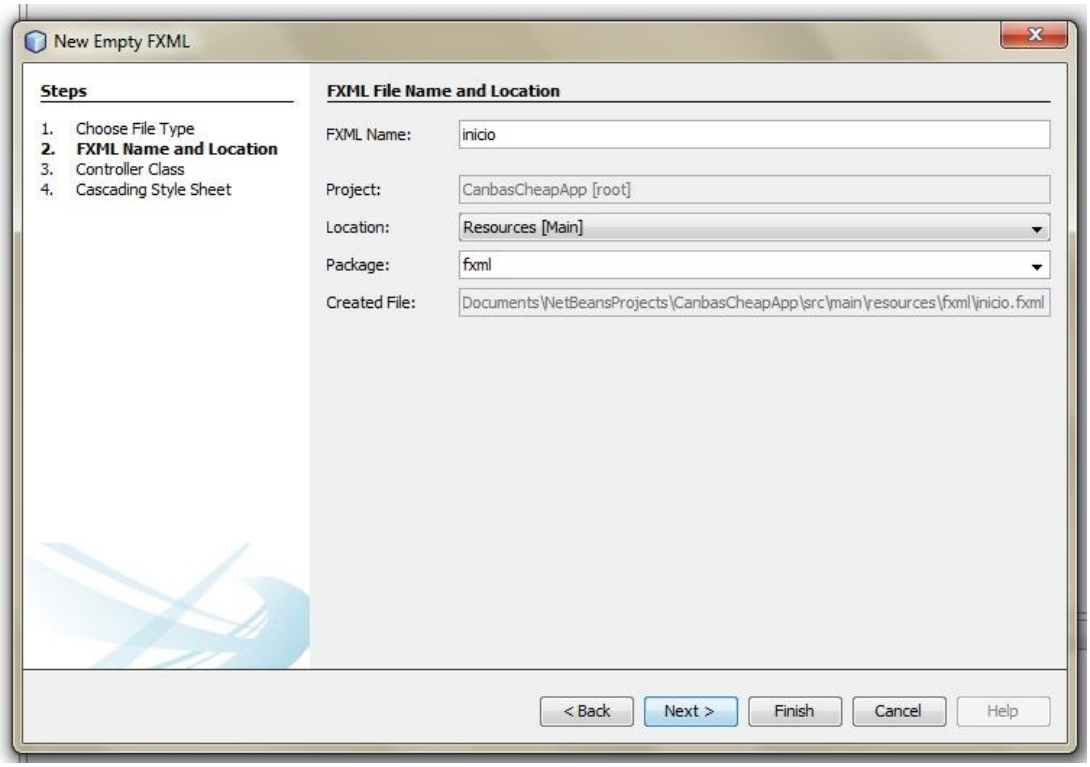


Figura 233 Creación de un archivo FXML Paso 2

Se colocó un nombre al archivo. fxml que se creó y por último se seleccionó la casilla para la creación de una hoja de estilo, ver Figura 24, por default esta tiene el mismo nombre que el archivo *.fxml.*

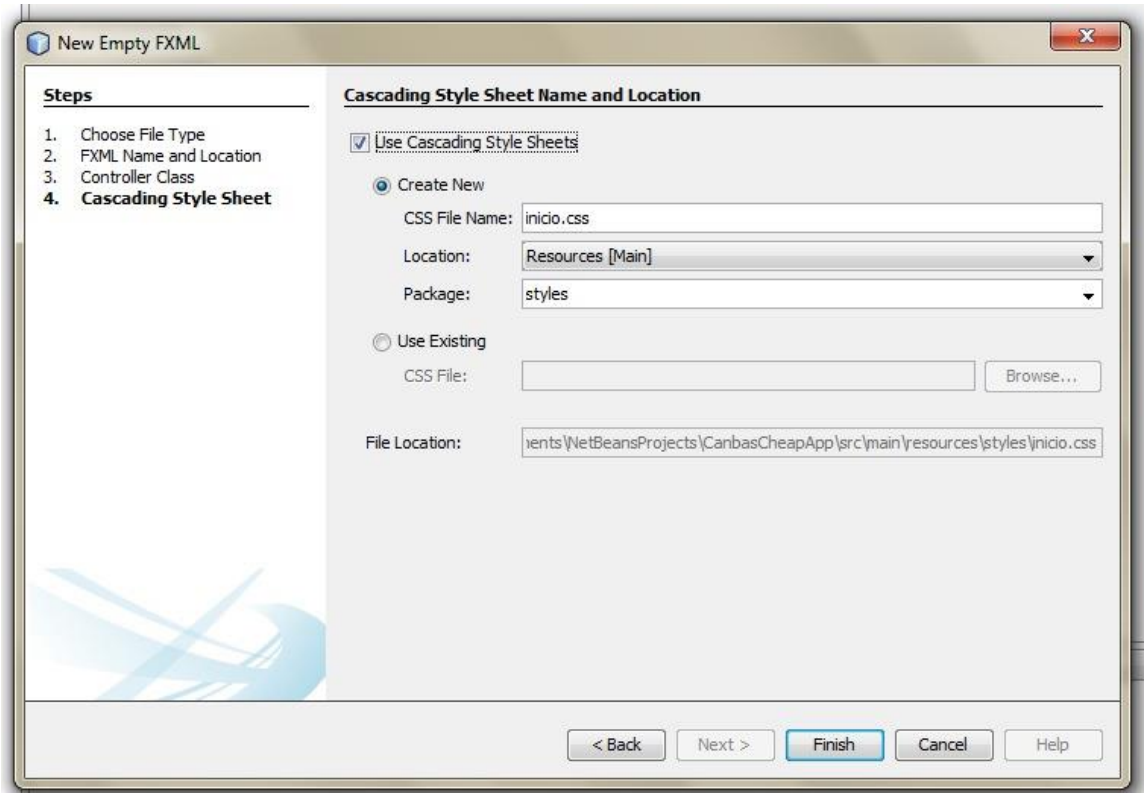


Figura 244 Creación de un archivo FXML Paso 3

Cada archivo *.css* correspondiente a una vista lleva como código el que se muestra en la Figura 25, este código puede ser cambiado si se desea para añadir más estilos a cada componente de la vista.

Figura 255 Código del archivo .css

Posteriormente se crearon dos clases Java en la carpeta Source Packages, paquete CanbasCheapApp.*views,* ver Figura 26. La primera clase es el Presentador y la otra una clase que extiende de FXMLView, estas clases deben llevar el nombre de la vista+*Presenter* y vista+*View*.

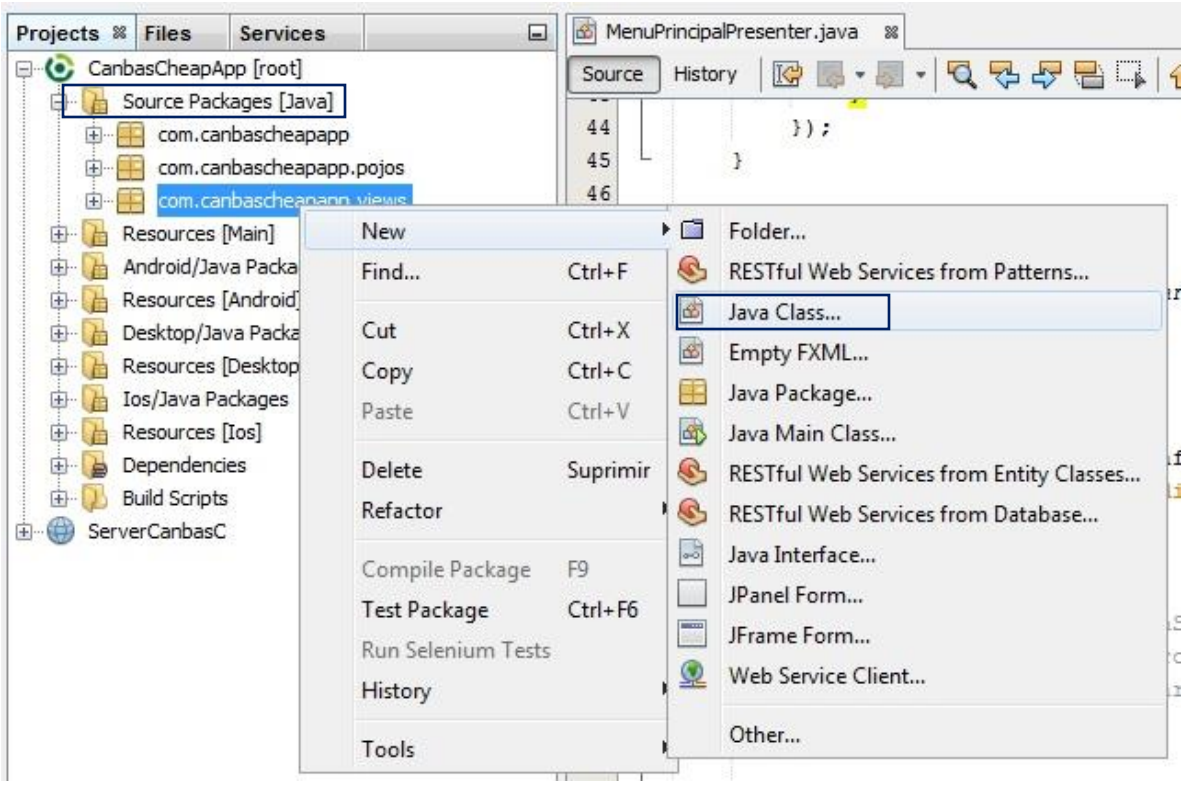


Figura 266 Creación de una clase Java como Presentador

La clase Presentador es la que implementa las funciones de los componentes que posee cada vista, a continuación, se explica cómo se desarrolló cada módulo de la aplicación cliente tomando como referencia el diagrama de casos de uso general mostrado en la Figura 21.

Implementación de cada Módulo

Para el diseño e implementación de cada vista se utilizó SceneBuilder [22], agregando los elementos necesarios para el correcto funcionamiento de la aplicación.

2.5.3.- Registrar/Login

La vista de inicio de la aplicación es la de ***Login***, en la que el usuario puede ingresar su nombre y contraseña, para ello se hizo uso de un *TextField,* un *PasswordField*, el botón de inicio y uno de registro, el diseño de esta vista en SceneBuilder se muestra en la Figura 27.

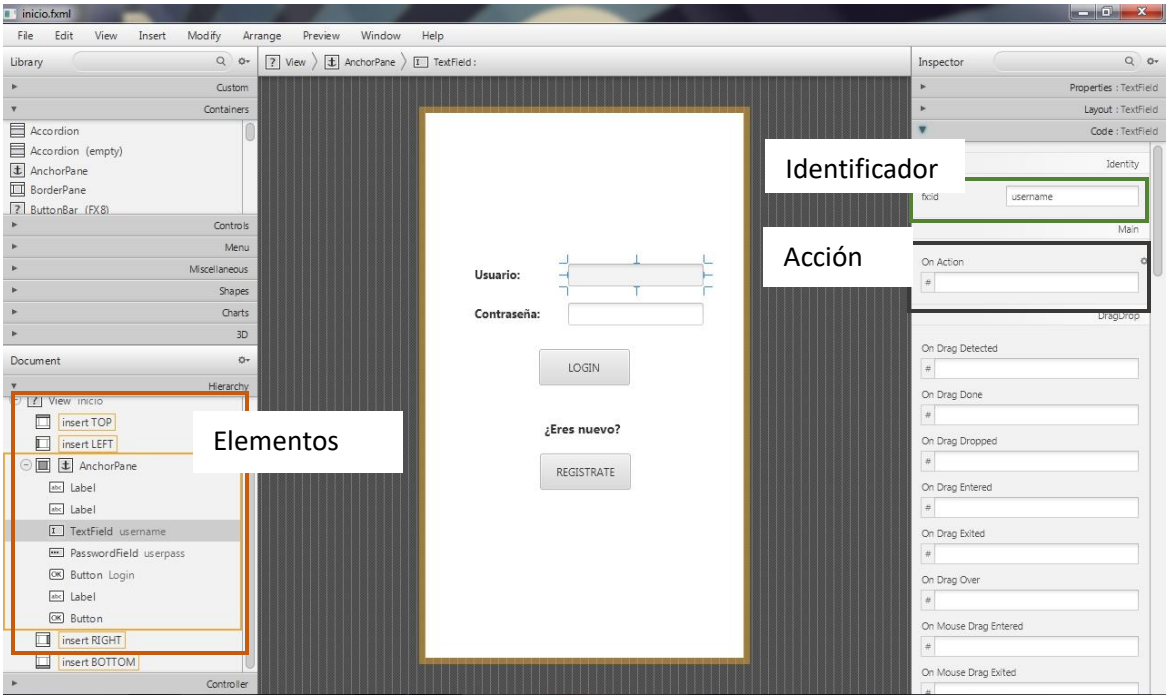


Figura 27 Diseño de la vista Login en SceneBuilder

SceneBuilder [22] trabaja con el archivo. fxml, basta con arrastrar y soltar el elemento que se desea agregar y este coloca automáticamente el código. Los elementos añadidos a la vista se encuentran en el apartado *Document,* por su parte el apartado *Inspector-> Code*, permite agregar un identificador y alguna acción a cada elemento.

En la clase InicioPresenter se agregaron los elementos previamente descritos, como se puede apreciar en la Figura 28, para ello se utilizó la notación @FXML la cual indica que estos están definidos en un archivo. fxml.

Como se puede observar fue necesario colocar un identificador a la vista para su correcta manipulación.

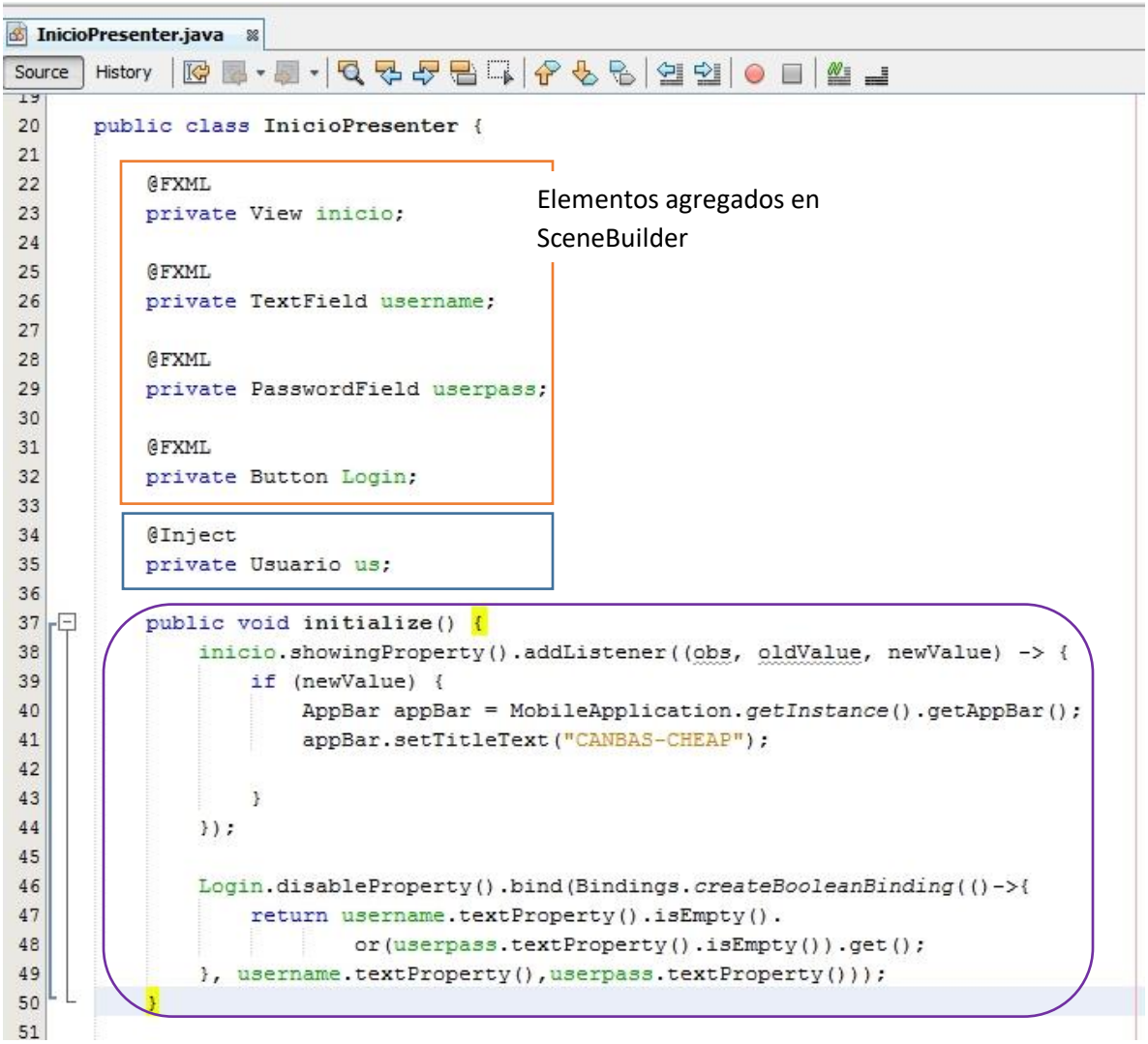


Figura 278 Código inicial de la clase InicioPresenter

La notación @Inject permite pasar un objeto de una vista a otra, en este caso el objeto de tipo usuario, el cual estará definido en todas las vistas de la aplicación excepto en la de Registro.

El método initialize() es el que se invoca siempre cuando se carga la vista, dentro de él se encuentra un escuchador , el cual cada vez que percibe un cambio muestra un AppBar con el título “Canbas-Cheap”, por otra parte se enlazó al botón Login con el TextField y con el PasswordField para asegurar que no estén vacíos, de lo contrario no se activará la funcionalidad del botón.

Esta vista tiene dos botones, el primero Login, este recupera los datos introducidos en el TextField y en el PasswordField y envía estos al servidor a través de una petición vía ***Rest,*** el segundo Registrar sólo sirve para cambiar a la vista de Registro**.**

Para poder consumir un servicio Rest, Gluon Mobile hace uso de la librería ***Gluon Connect*** [24].

Para ello primero se creó una función loginUsuario(), en la que se realiza la petición al servicio por medio del código que se muestra en la Figura 29.

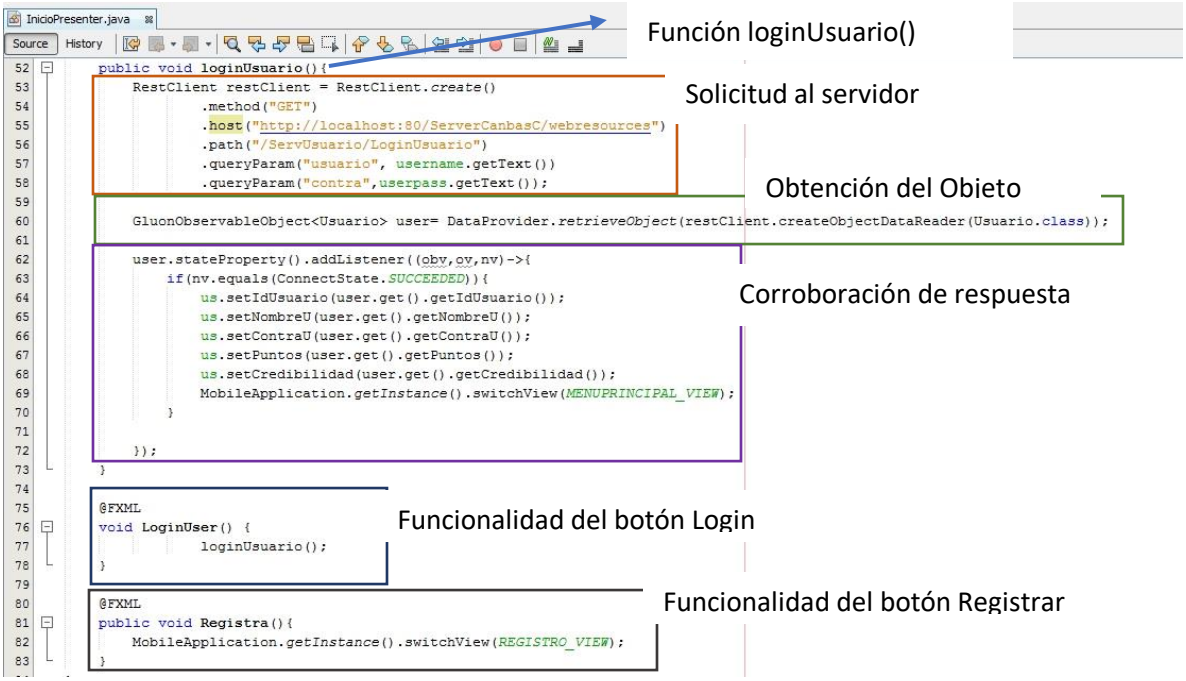


Figura 29 Funcionalidad de los botones de la vista inicio

A continuación se explica el código mostrado:

- *Solicitud al servidor:* Se creó un objeto de tipo *RestClient*, a este se le colocaron sus respectivas funciones, añadiendo los parámetros requeridos, como *method*, en este caso get porque se requiere recuperar un objeto de tipo usuario, *host,* la dirección del servidor (“http://localhost:80/ServerCanbasC/webresources”), *path*, la dirección relativa al servicio que se desea invocar y la función de este (“/ServUsuario/LoginUsuario”), *queryParam* en este caso dos: “usuario” y “contra” , los cuales reciben los datos proporcionados en el TextField “username” y el PasswordField “contra”.
- *Obtención del objeto:* Gluon Connect proporciona la clase DataProvider a través de la cual se pueden obtener instancias de GluonObservableObject, para ello se hace uso del método retrieveObject indicándole el tipo de objeto que se obtendrá.
- *Corroboración de respuesta*: al hacer uso de GluonObservableObject se puede hacer uso del método stateProperty y addListener para saber si ha ocurrido un cambio en el objeto, es decir, si se recuperó de la

solicitud al servidor, si este indica que la solicitud fue exitosa se fijan los valores al objeto usuario a través de los métodos set.

- *Funcionalidad del botón Login*: Llama al método loginUsuario() explicado previamente y posteriormente cambia a la vista MenuPrincipal.
- *Funcionalidad del botón Registrar*: Invoca al método SwitchView de GluonMobileApplication para cambiar a la vista de Registro.

Vista Registro

La vista Registro está compuesta por un TextField, que permite al usuario ingresar su nombre, dos PasswordField en los que debe ingresar su contraseña y un botón de Registro, el diseño de esta vista en SceneBuilder se muestra en la Figura 30.

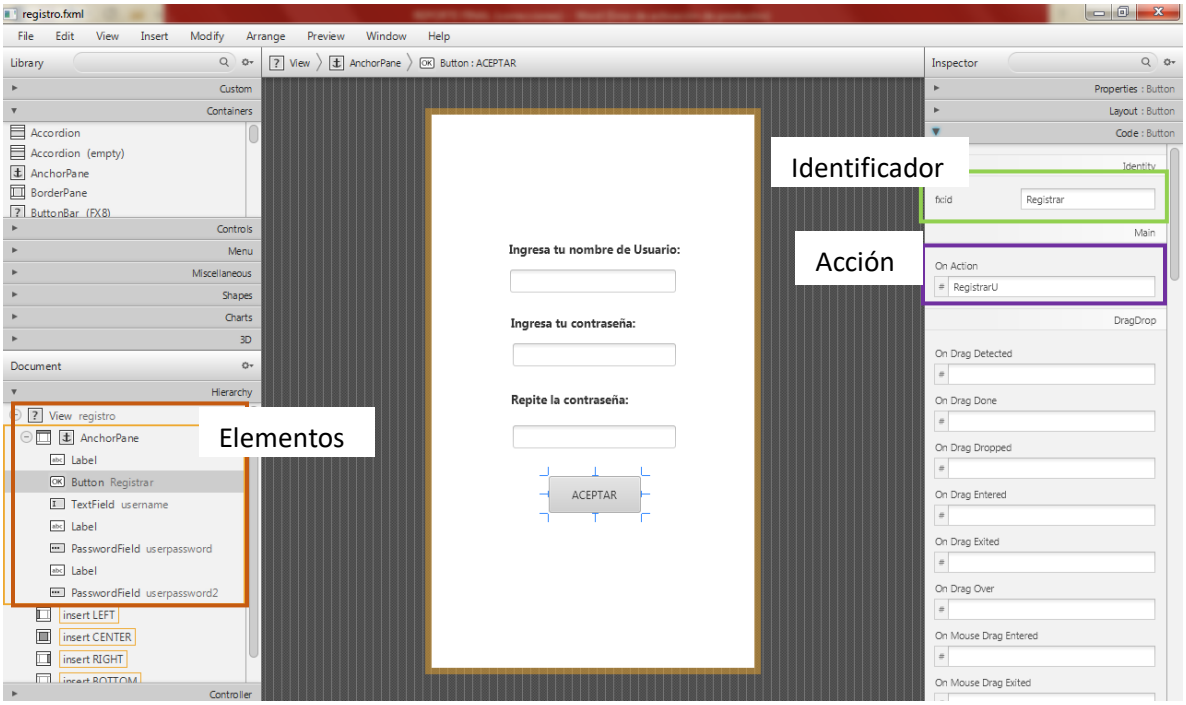
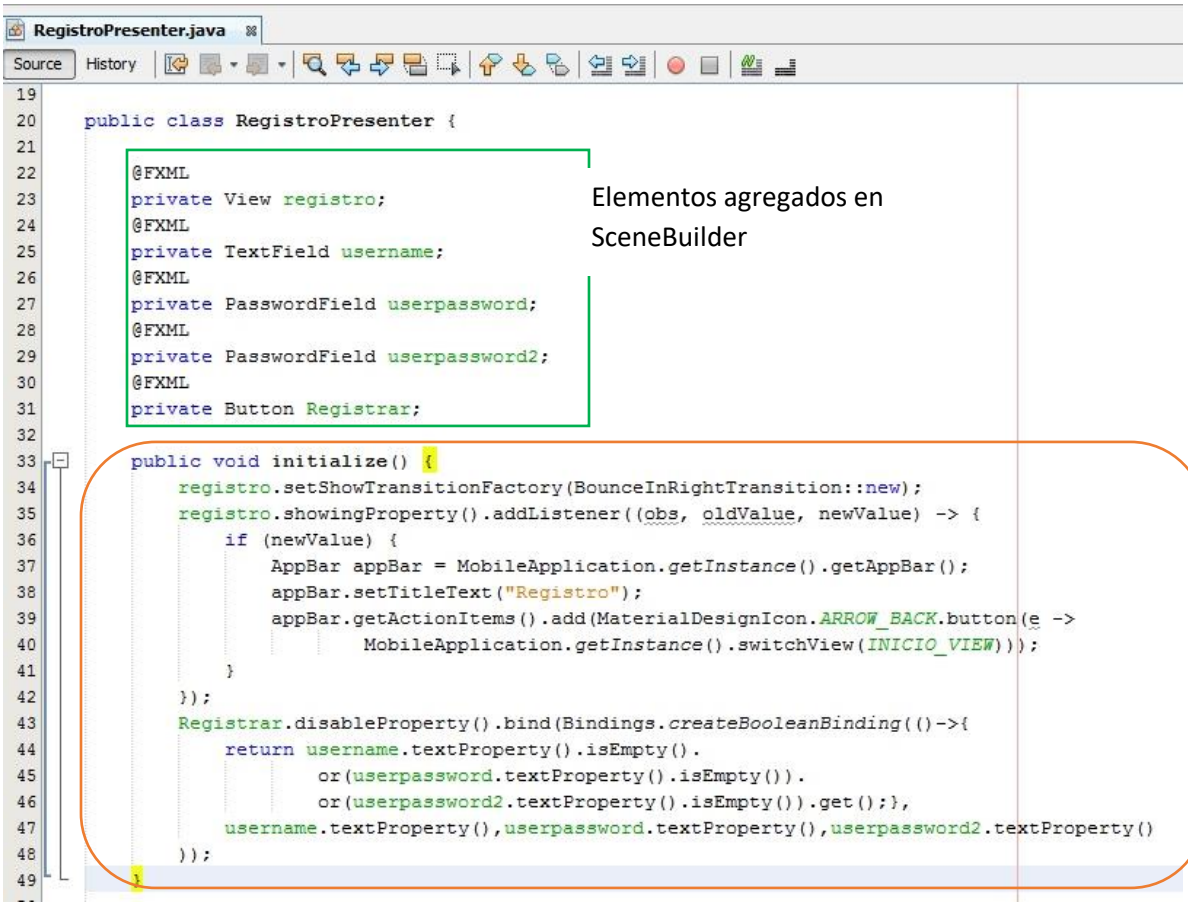


Figura 30 Diseño de la vista Registro en SceneBuilder

En la clase RegistroPresenter mostrada en la Figura 31, se añadieron los elementos previamente descritos para su manipulación.




```
public class RegistroPresenter {

    @FXML
    private View registro;
    @FXML
    private TextField username;
    @FXML
    private PasswordField userpassword;
    @FXML
    private PasswordField userpassword2;
    @FXML
    private Button Registrar;

    public void initialize() {
        registro.setShowTransitionFactory(BounceInRightTransition::new);
        registro.showingProperty().addListener((obs, oldValue, newValue) -> {
            if (newValue) {
                AppBar appBar = MobileApplication.getInstance().getAppBar();
                appBar.setTitleText("Registro");
                appBar.getActionItems().add(MaterialDesignIcon.ARROW_BACK.button(e ->
                        MobileApplication.getInstance().switchView(INICIO_VIEW)));
            }
        });
        Registrar.disableProperty().bind(Bindings.createBooleanBinding(()->{
            return username.textProperty().isEmpty().
                    or(userpassword.textProperty().isEmpty()).
                    or(userpassword2.textProperty().isEmpty()).get();},
            username.textProperty(),userpassword.textProperty(),userpassword2.textProperty()
        ));
    }
```

Figura 31 Código inicial de la clase RegistroPresenter

En el método initialize() se agregó una transición a la vista haciendo uso del método *setShowTransitionFactory()* para hacer más notorio el cambio entre vistas, también se agregó la propiedad del escuchador a esta para mostrar un AppBar con el título “Registro” y un botón el cual permite volver a la vista inicio, por otra parte se enlazó al botón Registrar con el TextField y los dos PasswordField esto para corroborar que ninguno de los tres este vacío, de lo contrario no se activa la función del botón.

Esta vista posee un solo botón el cual tiene la funcionalidad de enviar los datos al servidor para crear un nuevo registro de usuario, para ello primero se creó la función registraUsuario(), ver Figura 32.

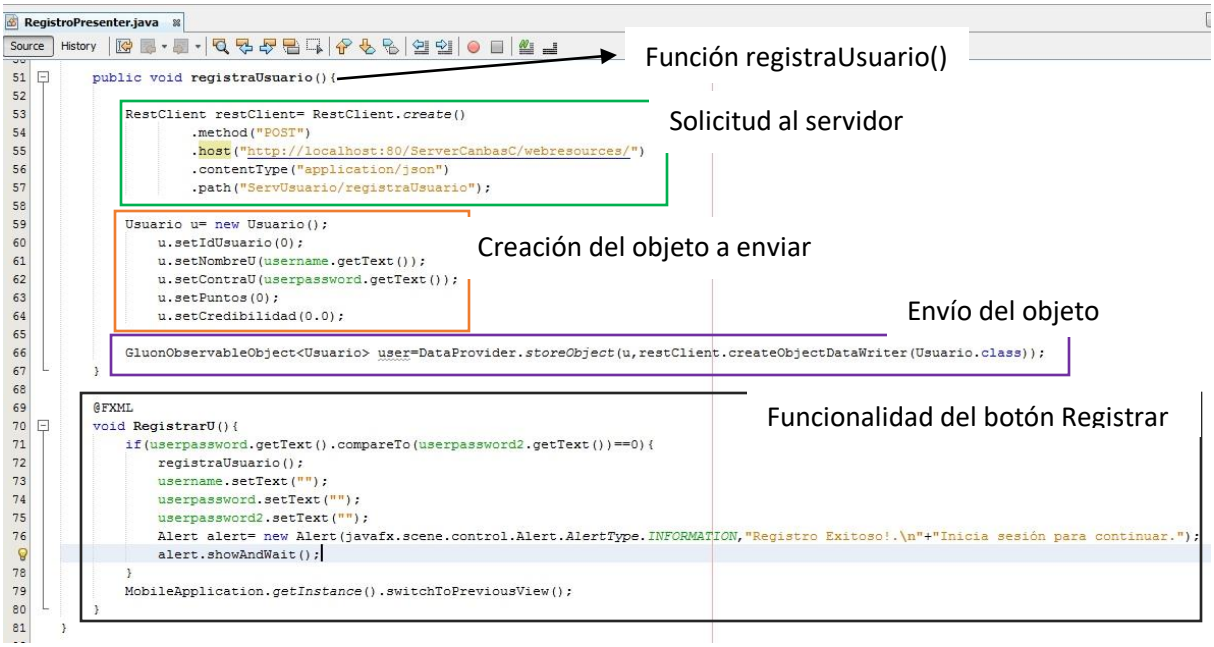


Figura 32 Funcionalidad del botón Registrar de la vista Registro

A continuación se explica el código mostrado:

- *Solicitud al Servidor:* Se creó un objeto RestClient proporcionado por la librería GluonConnect, se colocaron sus respectivas funciones añadiendo los parámetros necesarios, *method*, en este caso POST, puesto que se desea enviar un objeto al Servidor para la creación de un nuevo registro de usuario en la base de datos, *host*, la dirección del servidor (“http://localhost:80/ServerCanbasC/webresources/”), *contentType*, el tipo de contenido a enviar, en este caso un objeto en formato JSON (“application/json”), por último *path*, haciendo referencia a la dirección del servicio y la función a invocar.
- *Creación del objeto a enviar*: Se creó una nueva instancia del objeto Usuario, colocándole los datos proporcionados recuperados del TextField y PasswordField, tales como nombre de usuario TextField “username”, contraseña PasswordField “contra”, puntos y credibilidad están en blanco puesto que estos datos los coloca la función LoginUsuario del servidor.
- *Envío del objeto:* Se hace uso de GluonObservableObject para definitir el objeto a enviar y de DataProvider con la función storeObject en la que se coloca como parámetro inicial el objeto a enviar y posteriormente el tipo de objeto a enviar.
- *Funcionalidad del botón Registrar:* Primero se hizo una comparación para corroborar que la contraseña proporcionada sea la misma en ambos PasswordField, si es así, se invoca la función registraUsuario(), la cual permite envío de datos al Servidor, una vez hecho esto se

colocan los TextField y PasswordField en blanco y se muestra una alerta al usuario indicando que el registro se realizó de manera exitosa. Posteriormente se hace el cambio a la vista de inicio, haciendo uso de GluonMobileApplication.switchToPreviousView.

Vista Menu Principal

Esta vista muestra las diferentes funciones que proporciona la aplicación, ver Figura 33.

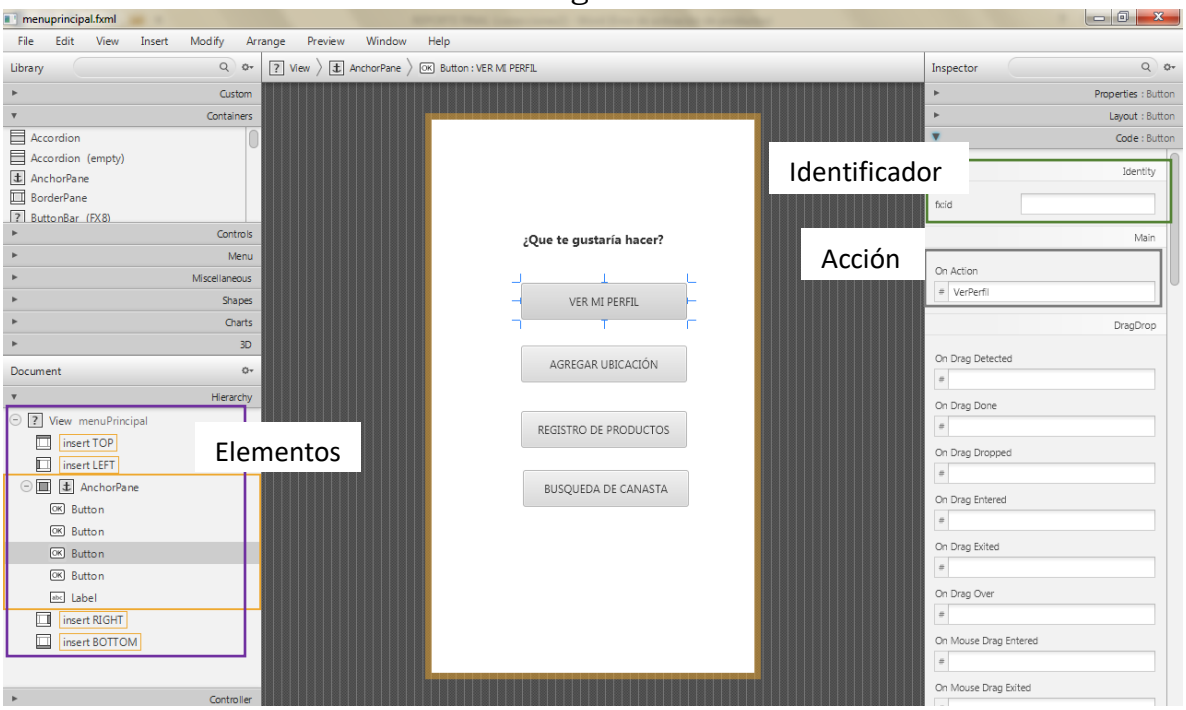


Figura 283 Diseño de la vista MenuPrincipal en SceneBuilder

Para cumplir los objetivos de esta vista, se hizo uso de cuatro botones los cuáles permiten el acceso a las funciones de la aplicación, tales como *Ver Mi Perfil*, *Agregar Ubicación*, *Registro de Productos* y *Búsqueda de Canasta*, para ello se definió una acción a cada uno en el apartado Code-> On Action de SceneBuilder, cabe mencionar que el nombre de esta acción debe encontrarse en la vista MenuPrincipalPresenter.

En la vista MenuPrincipalPresenter, ver Figura 34, se implementó cada una de las funciones que puede llevar a cabo cada botón mostrado en esta vista.

Figura 294 Código inicial de la clase MenuPrincipalPresenter

En la clase MenuPrincipalPresenter se agregaron los elementos como la vista con su respectivo Identificador, un objeto de tipo Usuario y uno de tipo Producto con la notación @Inject, el usuario es el que se creó desde la vista de inicio, por otro lado el Producto se creó desde esta vista puesto que como pruebas iniciales se hizo uso del plugin BarcodeScanService[25] de GluonCharmDown [19] para leer el código de barras de los productos y en base a ello pasar el código del producto a la vista de registroProducto, esta funcionalidad se explica a detalle con la función Registrar Producto.

En el método Initialize() se colocó una transición para hacer más notorio el cambio entre vistas, se colocó también un escuchador a la vista, en el que cada vez que este recibe un valor nuevo, es decir, cada vez que se cargue la vista, ejecutará el código de crear un AppBar con el título “Bienvenido a Canbas-Cheap”.

A continuación, se muestra y se explica el código que lleva acabo cada uno de los botones, ver Figuras 35 y 36:

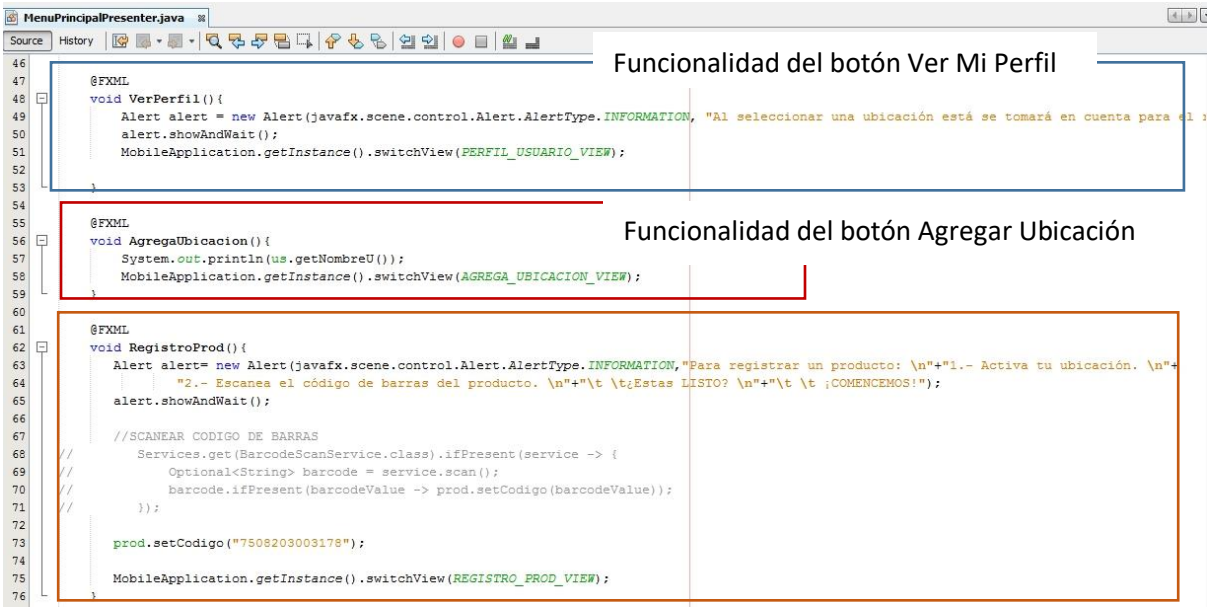


Figura 305 Código de los botones en la vista MenuPrincipal

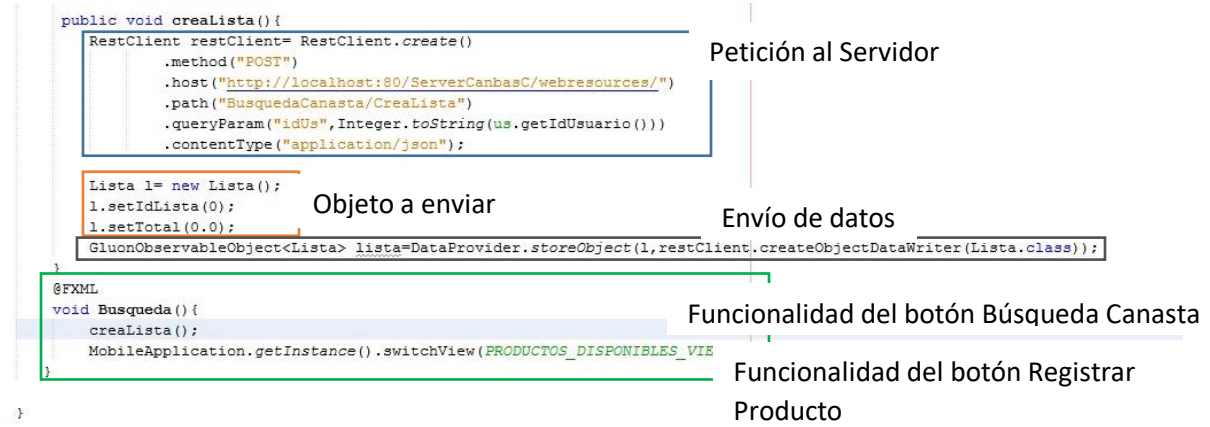


Figura 316 Código del botón Búsqueda canasta en la vista MenuPrincipal

- *Funcionalidad del botón Ver Mi Perfil:* Primero muestra una alerta indicando al usuario que la ubicación que seleccione dentro de la vista Perfil será utilizada cada vez que registre un producto o consulte las tiendas cercanas, una vez hecho esto cambia a la vista PerfilUsuario haciendo uso de MobileApplication.getInstance().switchView(PERFIL_USUARIO_VIEW.
- *Funcionalidad del botón Agregar Ubicación:* Cambia a la vista AgregarUbicación haciendo uso de MobileApplication.getInstance().switchView(AGREGA_UBICACION_VIEW).
- *Funcionalidad del botón Registrar Producto:* Muestra una alerta indicando los pasos a seguir para el correcto registro de un producto:

1.- Activa tu ubicación, 2.- Escanea el código de barras del producto. Posteriormente fija el código al producto a través del método set y hace el cambio a la vista de registro de producto haciendo uso de MobileApplication.getInstance().switchView(REGISTRO_PROD_VIEW)
.

Cabe señalar que el código comentado es funcional para dispositivos móviles, en este caso, se probó en un dispositivo Android versión 4.2.2, para hacer uso de este plugin se colocó el código mostrado en la Figura 37, en el archivo AndroidManifest.xml encontrado en la carpeta CanbasCheapApp\src\android.

```
xmlns:android="http://schemas.android.com/apk/res/android"
package="com.canbascheapapp" android:versionCode="1"
android:versionName="1.0">
        <supports-screens android:xlargeScreens="true"/>
<uses-permission android:name="android.permission.CAMERA"/>
        <uses-permission
android:name="android.permission.INTERNET"/>
        <uses-permission
android:name="android.permission.READ_EXTERNAL_STORAGE"/>
        <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
        <uses-sdk android:minSdkVersion="4"
android:targetSdkVersion="21"/>
        <application android:label="CanbasCheapApp"
android:name="android.support.multidex.MultiDexApplication"
android:icon="@mipmap/ic_launcher">
<activity
android:name="com.gluonhq.impl.charm.down.plugins.android.scan.z
xing.CaptureActivity"
                android:screenOrientation="sensorLandscape"
                android:clearTaskOnLaunch="true"
                android:stateNotNeeded="true"
                android:windowSoftInputMode="stateAlwaysHidden">
        <intent-filter>
<action
android:name="com.gluonhq.charm.down.plugins.android.scan.SCAN"/
>
          <category
android:name="android.intent.category.DEFAULT"/>
        </intent-filter>
      </activity>
```

Figura 327 Modificación al archivo AndroidManifest.xml para hacer uso del plugin BarcodeScanService

Cabe señalar que algunos plugins proporcionados por GluonCharmDown[19] son funcionales solo para dispositivos móviles, en este caso BarcodeScanService [25], por lo que este código no fue funcional para pruebas de la aplicación en su versión Web, teniendo que fijar con el método set algún código de manera manual.

- *Funcionalidad del botón Búsqueda Canasta:* Llama a la función creaLista(), la cual envía una instancia de esta al servidor para que este genere un nuevo registro para el usuario que está usando la aplicación, para ello hace uso de RestClient y sus funciones como create() el cual crea una instancia de este, host la dirección del servidor (http://localhost:80/ServerCanbasC/webresources/), path la dirección del servicio y su función a invocar ("BusquedaCanasta/CreaLista"), method en este caso "POST", contentType el tipo de dato a enviar ("application/JSON") y por último queryParam con el id de usuario, una vez hecho esto, se utilizó GluonObservableObject<Lista> y DataProvider con su función storeObject() a la cual se le colocaron como parámetros el objeto a enviar y el tipo de dato de este, por lo que primero se instancio el objeto Lista, para el envío al servidor, posteriormente se realiza el cambio a la vista ProdDisponibles haciendo uso de MobileApplication.getInstance().switchView(PRODUCTOS_DISPONIBLES_VIEW).

2.5.4.-Ver Perfil

La vista perfilusuario, ver Figura 38, permite al usuario revisar sus datos como su nombre, sus puntos y su credibilidad acumulados, así mismo permite también visualice sus ubicaciones registradas y con ello seleccione una para la búsqueda de tiendas y registro de productos.

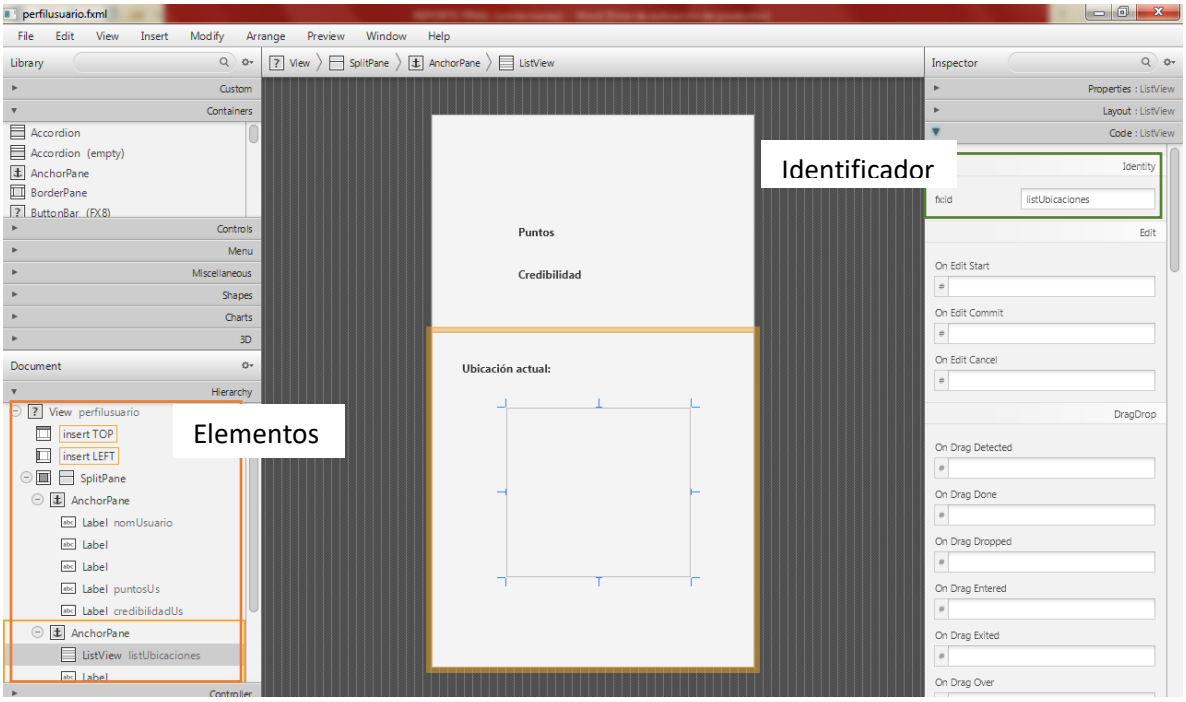


Figura 338 Diseño de la vista perfilusuario en SceneBuilder

Para cumplir con las tareas que debe proporcionar esta vista, se utilizó un SplitPane vertical que permite agrandar el espacio de uno de los dos AnchorPane en donde se colocaron los elementos. En el AnchorPane superior se agregaron tres elementos Label, uno para colocar el nombre del usuario, otro para colocar sus puntos y el último para colocar su credibilidad. En el AnchorPane inferior se agregaron un ListView que muestra las ubicaciones que tiene registradas el usuario y un Label que muestra la ubicación seleccionada.

En la vista perfilUsuarioPresenter se agregaron los elementos mencionados previamente, ver Figura 39.

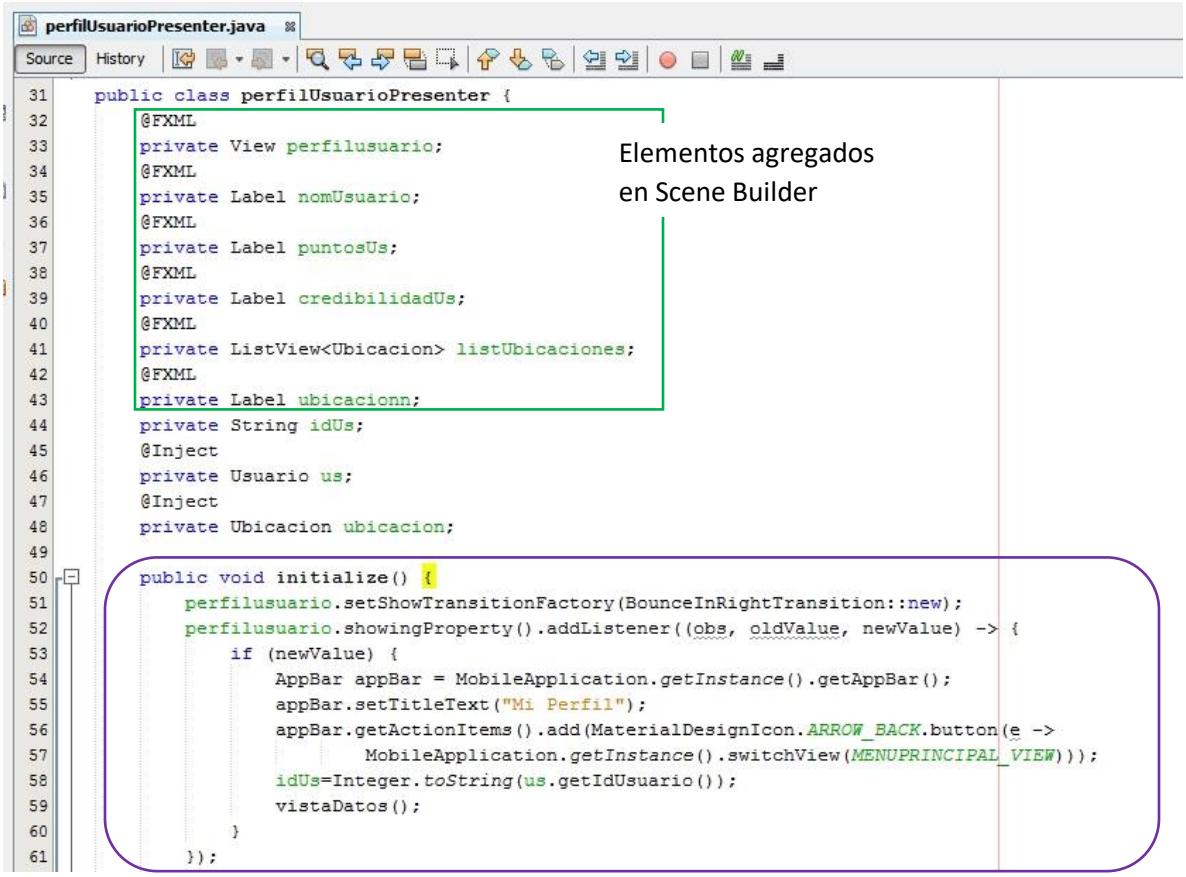


Figura 39 Código inicial de la clase perfilUsuarioPresenter

Dentro del método *initialize(),* se agregó una transición a la vista para hacer más notorio el cambio entre estas, haciendo uso de *setShowTransitionFactory* ,también se hizo uso de un escuchador, el cual cada que carga la vista, muestra un AppBar con el título “Mi Perfil”, un botón el cual permite volver la vista de MenúPrincipal y llama a la función vistaDatos(), por último y ya fuera del escuchador, se agregó una alerta para indicarle al usuario que si aún no tiene registradas ubicaciones registre al menos una, este código se muestra en la Figura 40.

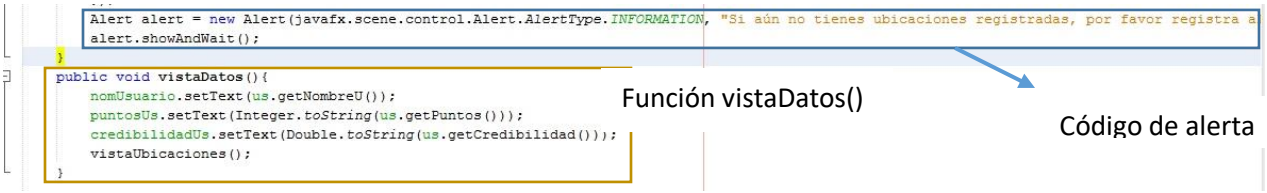


Figura 340 Código de alerta y función vistaDatos en la vista perfilusuario

La función *vistaDatos()* se encarga de mostrar los datos del usuario previamente recuperado en la vista de inicio con la función del botón Login, además llama al método *vistaUbicaciones()* el cual para mostrar las ubicaciones registradas por el usuario hace una petición al Servidor vía REST, este código se muestra en la Figura 41.

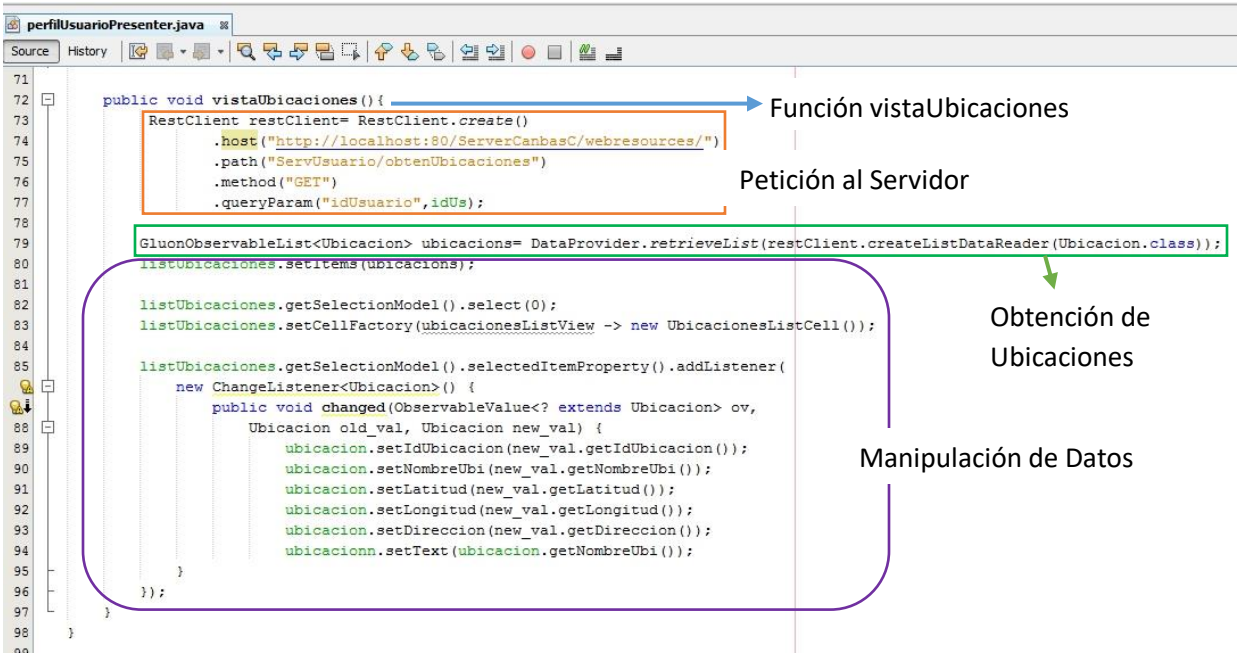


Figura 351 Código vistaUbicaciones de la clase perfilUsuarioPresenter

A continuación se explican cada uno de los componentes de la función vistaUbicaciones:

- *Petición al Servidor:* Se creó un objeto de tipo RestClient y se colocó cada una de sus funciones añadiendo los parámetros requeridos, host, la dirección del servidor (http://localhost:80/ServerCanbasC/webresources), path la dirección relativa al servicio y la función de este que se desea invocar ("servUsuario/obtenUbicaciones"), method en este caso GET puesto que se requiere la obtención de una lista de objetos tipo Ubicación y por último queryParam, "idUsuario" idus, un String al que se le otorgó

el valor del id de usuario del objeto us, haciendo un cast a este de Double a String, desde el método initialize().

- *Obtención de Ubicaciones:* Se hizo uso de GluonObservableList<ubicacion> y de DataProvider haciendo uso de su función retrieveList proporcionándole el tipo de objeto de List que se obtendrá.
- *Manipulación de Datos:* Se asignaron los datos obtenidos de la petición al servidor a la ListView "listUbicaciones", esto a través de la función *setItems()* de ListView, se creó una clase *UbicacionesListCell* para mostrar de manera correcta el nombre de cada ubicación, posteriormente se agregó un escuchador a listUbicaciones para mostrar en la etiqueta ubicacionn el nombre de la ubicación seleccionada, haciendo uso del código *getSelectionModel().selectedItemProperty.addListener(new ChangeListener<Ubicacion>()),* cabe señalar que la ubicación seleccionada es la que se usa para el registro de productos y para la búsqueda de canasta, por lo que es utilizada en otras vistas, para ello al inicio de la clase perfilUsuarioPresenter se declaró un objeto tipo ubicación con la notación *@Inject*, el cual al ocurrir un cambio en la selección es instanciado con la ubicación seleccionada.

La clase UbicacionesListCell se muestra en la Figura 42, esta permite mostrar en cada Item de la lista el nombre de cada ubicación, ya que sin el uso de esta clase se muestra como tal la referencia del objeto y esto no sería entendible para el usuario.

UbicacionesListCell.java

Source | History

```
15  public class UbicacionesListCell extends ListCell<Ubicacion>{
16      @Override
17      protected void updateItem(Ubicacion ubicacion, boolean empty) {
18          super.updateItem(ubicacion, empty);
19
20          if(empty || ubicacion == null) {
21
22              setText(null);
23              setGraphic(null);
24
25          } else {
26              setText(ubicacion.getNombreUbi());
27              setGraphic(null);
28              }
29          }
30  }
```

Figura 362 Código de la clase UbicacionesListCell

2.5.5.- Agregar Ubicación

Cabe señalar que la planeación inicial de la aplicación fue que esta reconociera la posición del usuario en base a su GPS y aplicar la geocodificación a esta para obtener la dirección, para ello se utilizó el plugin PositionService[26] de GluonCharmDown[19], por lo que se modificó el archivo AndroidManifest.xml como se muestra en la Figura 43 y se realizaron pruebas de su correcta funcionalidad en un dispositivo móvil con sistema Android 4.2.2, el resultado fue exitoso, pero dado que GMapsFX[21] no funciona en dispositivos móviles, la versión final de la aplicación fue Web, con la dificultad de que el plugin PositonService[26] solo es funcional para móvil, se optó por cambiar el reconocimiento de la posición del usuario por la introducción directa de la dirección y en base a ella realizar la geocodificación y obtener las coordenadas que son necesarias para el cálculo de distancia entre tienda-ubicación.

```
<?xml version="1.0" encoding="UTF-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
package="com.canbascheapapp" android:versionCode="1"
android:versionName="1.0">
        <supports-screens android:xlargeScreens="true"/>
<uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.CAMERA"/>
        <uses-permission
android:name="android.permission.INTERNET"/>
        <uses-permission
android:name="android.permission.READ_EXTERNAL_STORAGE"/>
        <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
        <uses-sdk android:minSdkVersion="4"
android:targetSdkVersion="21"/>
        <application android:label="CanbasCheapApp"
android:name="android.support.multidex.MultiDexApplication"
android:icon="@mipmap/ic_launcher">
<activity
android:name="com.gluonhq.impl.charm.down.plugins.android.scan.zx
ing.CaptureActivity"
                android:screenOrientation="sensorLandscape"
                android:clearTaskOnLaunch="true"
                android:stateNotNeeded="true"
                android:windowSoftInputMode="stateAlwaysHidden">
        <intent-filter>
<action
android:name="com.gluonhq.charm.down.plugins.android.scan.SCAN"/>
          <category
android:name="android.intent.category.DEFAULT"/>
```

Figura 373 Modificación al archivo AndroidManifest.xml para hacer uso del plugin PositionService

La vista guardaubicacion, ver Figura 44, permite al usuario registrar una nueva ubicación, para ello se hizo uso de GoogleMapView y dos TextField uno perteneciente a la dirección y él otro al nombre de ubicación.

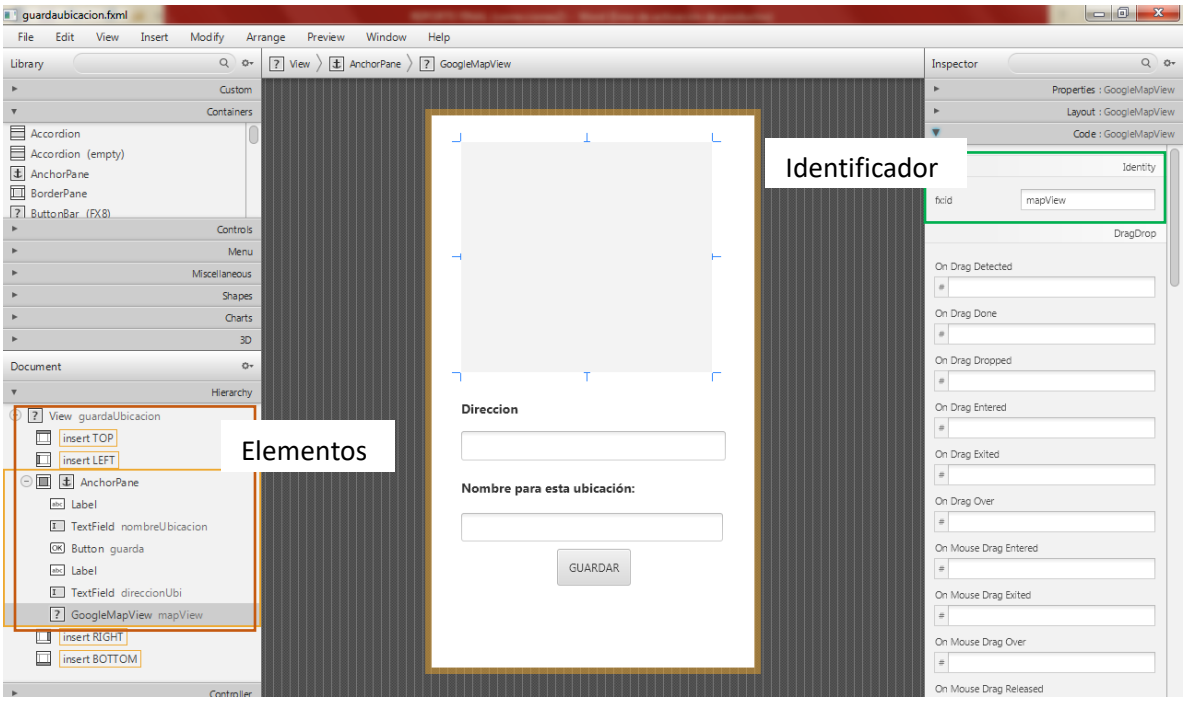
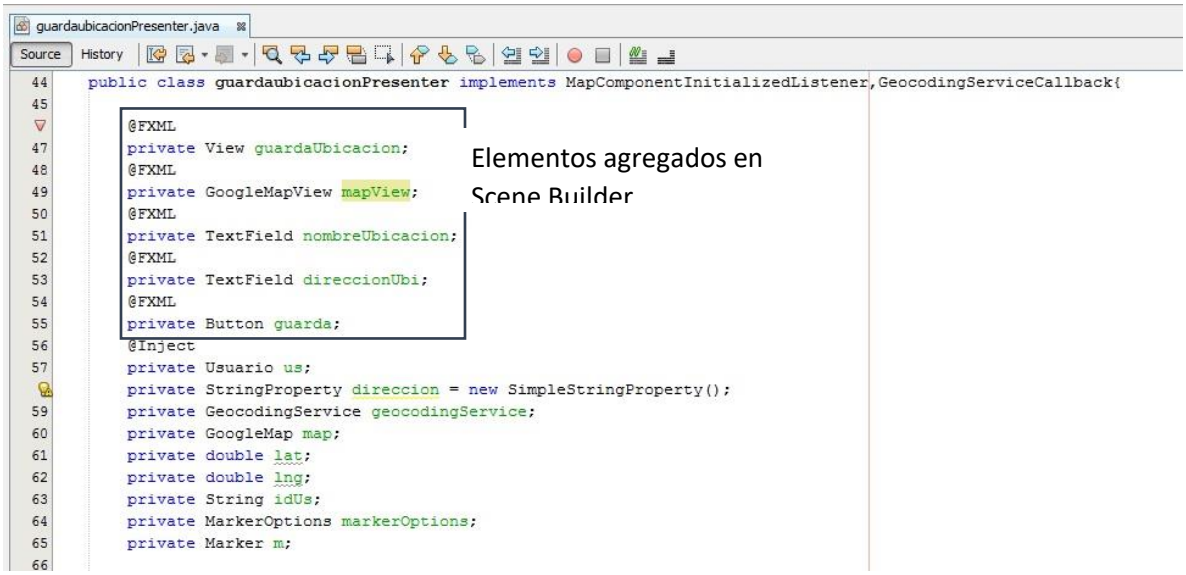


Figura 384 Diseño de la vista guardaubicacion en SceneBuilder

Para hacer uso de GoogleMapView se añadió primero el jar de GMapsFX[21] a SceneBuilder[22], este elemento permite mostrar el mapa y hacer uso de las funciones de Google Maps.

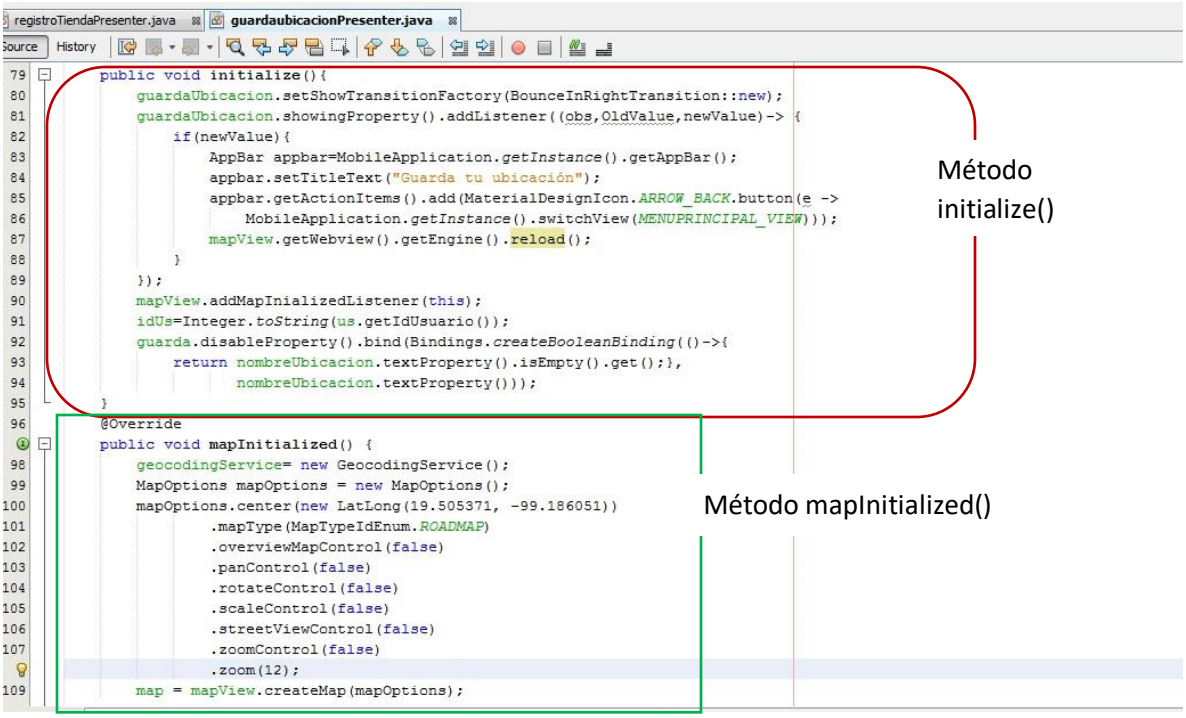
En la clase guardaubicacionPresenter, ver Figura 45, se colocaron los elementos mencionados previamente. Para hacer uso de las funciones de GoogleMap se implementó *MapComponentInitializedListener* y la interfaz del servicio que se usará en este caso *GeocodingServiceCallback*.




```
44  public class guardaubicacionPresenter implements MapComponentInitializedListener,GeocodingServiceCallback{
45
47      @FXML
48      private View guardaUbicacion;
49      @FXML
50      private GoogleMapView mapView;
51      @FXML
52      private TextField nombreUbicacion;
53      @FXML
54      private TextField direccionUbi;
55      @FXML
56      private Button guarda;
57      @Inject
59      private Usuario us;
60      private StringProperty direccion = new SimpleStringProperty();
61      private GeocodingService geocodingService;
62      private GoogleMap map;
63      private double lat;
64      private double lng;
65      private String idUs;
66      private MarkerOptions markerOptions;
67      private Marker m;
```

Figura 395 Atributos de la clase guardaubicacionPresenter

En el método initialize() mostrado en la Figura 46, se colocó una transición y un escuchador a la vista para mostrar cada que se cargue esta, un AppBar con el título “Guarda tu Ubicación” junto con un botón que permite volver a la vista de MenúPrincipal, posteriormente fuera de este escuchador se agregó otro a GoogleMapView “mapView” para inicializar la vista del mapa haciendo uso del método *mapInitialized(),* en el String idus se asignó el valor de id de usuario proporcionado por la variable us, para ello se hizo un cast de Double a String y por último se enlazó el botón Guarda con los dos TextField esto para corroborar que ninguno de los dos se encuentre vacío, o de lo contrario no se activa la función del botón.




```
79  public void initialize(){
80      guardaUbicacion.setShowTransitionFactory(BounceInRightTransition::new);
81      guardaUbicacion.showingProperty().addListener((obs,OldValue,newValue)-> {
82          if(newValue){
83              AppBar appbar=MobileApplication.getInstance().getAppBar();
84              appbar.setTitleText("Guarda tu ubicación");
85              appbar.getActionItems().add(MaterialDesignIcon.ARROW_BACK.button(e ->
86                  MobileApplication.getInstance().switchView(MENUPRINCIPAL_VIEW)));
87              mapView.getWebview().getEngine().reload();
88          }
89      });
90      mapView.addMapInializedListener(this);
91      idUs=Integer.toString(us.getIdUsuario());
92      guarda.disableProperty().bind(Bindings.createBooleanBinding(()->{
93          return nombreUbicacion.textProperty().isEmpty().get();},
94                  nombreUbicacion.textProperty()));
95  }
96  @Override
    public void mapInitialized() {
98      geocodingService= new GeocodingService();
99      MapOptions mapOptions = new MapOptions();
100     mapOptions.center(new LatLong(19.505371, -99.186051))
101             .mapType(MapTypeIdEnum.ROADMAP)
102             .overviewMapControl(false)
103             .panControl(false)
104             .rotateControl(false)
105             .scaleControl(false)
106             .streetViewControl(false)
107             .zoomControl(false)
                .zoom(12);
109     map = mapView.createMap(mapOptions);
```

Figura 406 Métodos initialize() y mapInitialized() de la clase guardaubicacionPresenter

A continuación se explica el código de MapInitialized():

Para hacer uso del servicio de geocodificación se creó una instancia de GeocodingService(), posteriormente se instanció el objeto mapOptions() para definir lo que se mostrará en el *mapView*, para ello se colocaron sus respectivas funciones y se añadieron los parámetros que se requieren; center hace referencia a cómo se mostrará el mapa en el GoogleMapView en este caso en la latitud y longitud (19.505371, -99.186051), mapType se refiere al tipo de mapa a mostrar, se eligió ROADMAP puesto que solo se pretende mostrar el mapa en 2D, overviewMapControl, panControl, rotateControl, scaleControl, streetViewControl y zoomControl en false ya que solo se pretende mostrar la vista con el mapa sin tener todas las características habilitadas que muestra normalmente GoogleMaps y por último el zoom de 12.

La función que se encarga de convertir la dirección de la ubicación del usuario en coordenadas(latitud,longitud) es actionGeocode(), ver Figura 47, para ello se invocó el método geocode del objeto GeocodingService, este recibe como parámetro la dirección de la que se obtendrán las coordenadas y la invocación de las funciones GeocodingResult y GeocodingStatus enviando los resultados y el estado obtenidos, se creó un objeto de tipo LatLong para hacer uso de los resultados obtenidos el cual se inicializó en null. Posteriormente se verificaron los resultados, en el primer bloque, se verifica que no se hayan encontrado resultados, por lo que se muestra una

alerta indicando que no se encontraron sugerencias y termina la función con return, los otros dos bloques indican que, si se hallaron resultados, por lo cual se coloca a latlong el valor obtenido, el cual es dividido en latitud y longitud para pasar este a su respectiva variable y se centra el mapa ahora en estas coordenadas.

Posteriormente se realizó la instanciación del objeto markerOptions, colocándole por medio de su función position las coordenadas obtenidas y en visible el valor true, para asegurar que será mostrado, una vez hecho esto se instanció el objeto Marker pasándole como parámetro el objeto markerOptions y por último se agregó este marcador al mapa por medio del método addMarker() de GoogleMap.

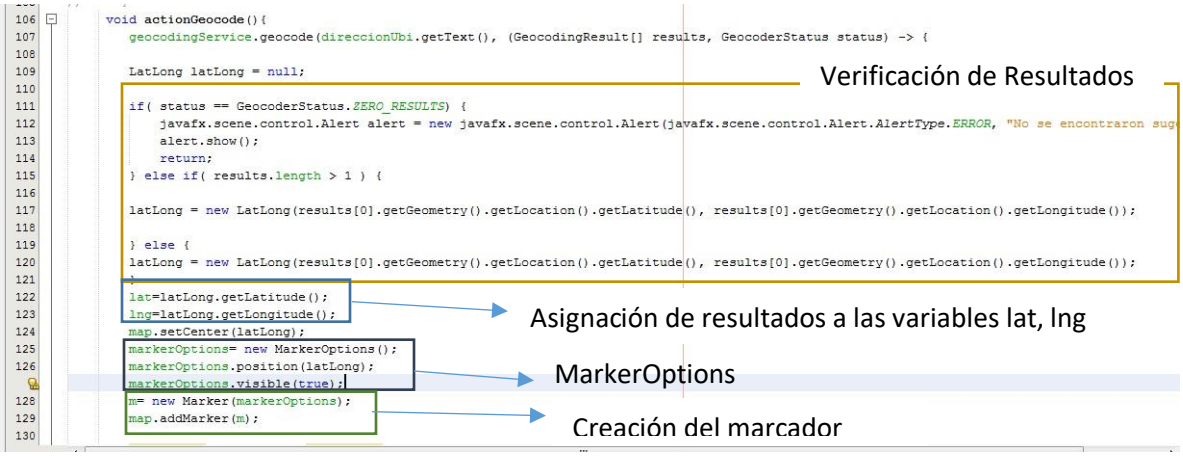


Figura 417 Código de la función actionGeocode de la clase guardaubicacionPresenter

Para el envío de esta ubicación al servidor se instanció un objeto de este tipo con la dirección y nombre proporcionados por el usuario y la latitud y longitud recuperadas, se creó un objeto de tipo RestClient para enviar este objeto al servidor de manera similar a como se envió el objeto usuario en el apartado de Registrar/Login haciendo uso de DataProvider.storeObject().

En la Figura 48 se muestra el código del botón Guardar que invoca la función actionGeocode().

```
    @FXML
    void guardar(){

//        Services.get(PositionService.class).ifPresent(serivce -> {
//            serivce.positionProperty().addListener((obs, oldPos, newPos) -> {
//                outputPos(newPos);
//            });
//        });

        actionGeocode();
        Alert alert= new Alert(javafx.scene.control.Alert.AlertType.INFORMATION,"Registro Exitoso!.");
        alert.showAndWait();
        MobileApplication.getInstance().switchToPreviousView();
    }
}
```

Figura 428 Funcionalidad del botón Guardar de la vista guardaubicacion

- *Funcionalidad del botón Guardar:* Llama al método actionGeocode() para hacer el envío de la ubicación al servidor, posteriormente muestra una alerta indicando que el registro de esta fue exitoso y por último hace el cambio a la vista de Menú Principal haciendo uso de MobileApplication.getInstance().switchToPreviousView().

2.5.6.- Registro de Productos

Para el correcto registro de productos es necesario registrar la tienda en que se encuentra dicho producto, proporcionar su precio, su categoría y subcategoría, por lo que la aplicación hace uso de tres vistas diferentes para permitir al usuario realizar cada una de estas acciones de manera sencilla.

A continuación, se explican las tres vistas utilizadas para el registro de productos:

Registro de Producto

Esta vista permite registrar un nuevo producto en base a su categoría y subcategoría perteneciente, para llevar a cabo estas tareas esta vista hace uso de un Label para colocar el código del producto a registrar, un TextField para el nombre del producto y dos ComboBox para la elección de categoría y subcategoría, el diseño de esta vista se muestra en la Figura 49.

Para la manipulación de los elementos Label, TextField y los dos ComboBox se le colocó a cada uno un identificador, el cual será usado en la clase registroproductoPresenter.

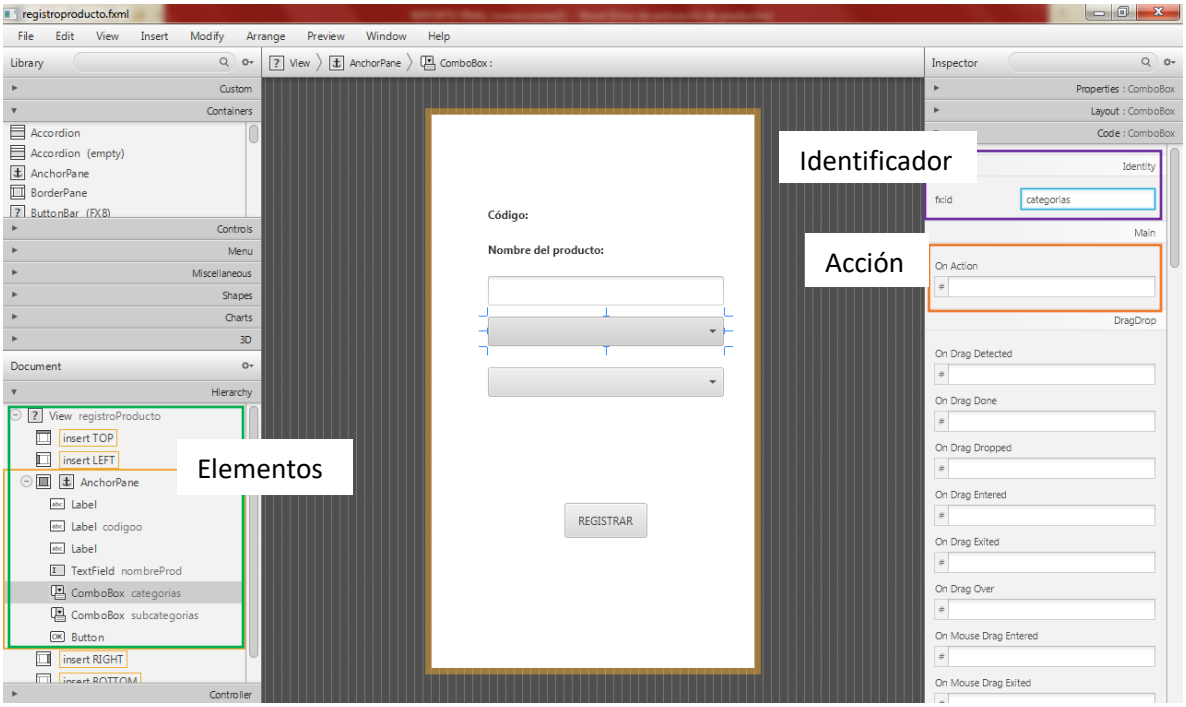


Figura 49 Diseño de la vista registroproducto en SceneBuilder

Para la manipulación de los elementos agregados en SceneBuilder[22], se añadieron estos con la notación @FXML y su respectivo identificador a la clase RegistroProductoPresenter mostrada en la figura 51 y se agregó además un usuario y un producto con notación @Inject, esto para recuperar los datos del usuario que está usando la aplicación y el producto que ha sido generado previamente en la clase de MenuPrincipalPresenter (ver sección 2.5.3).

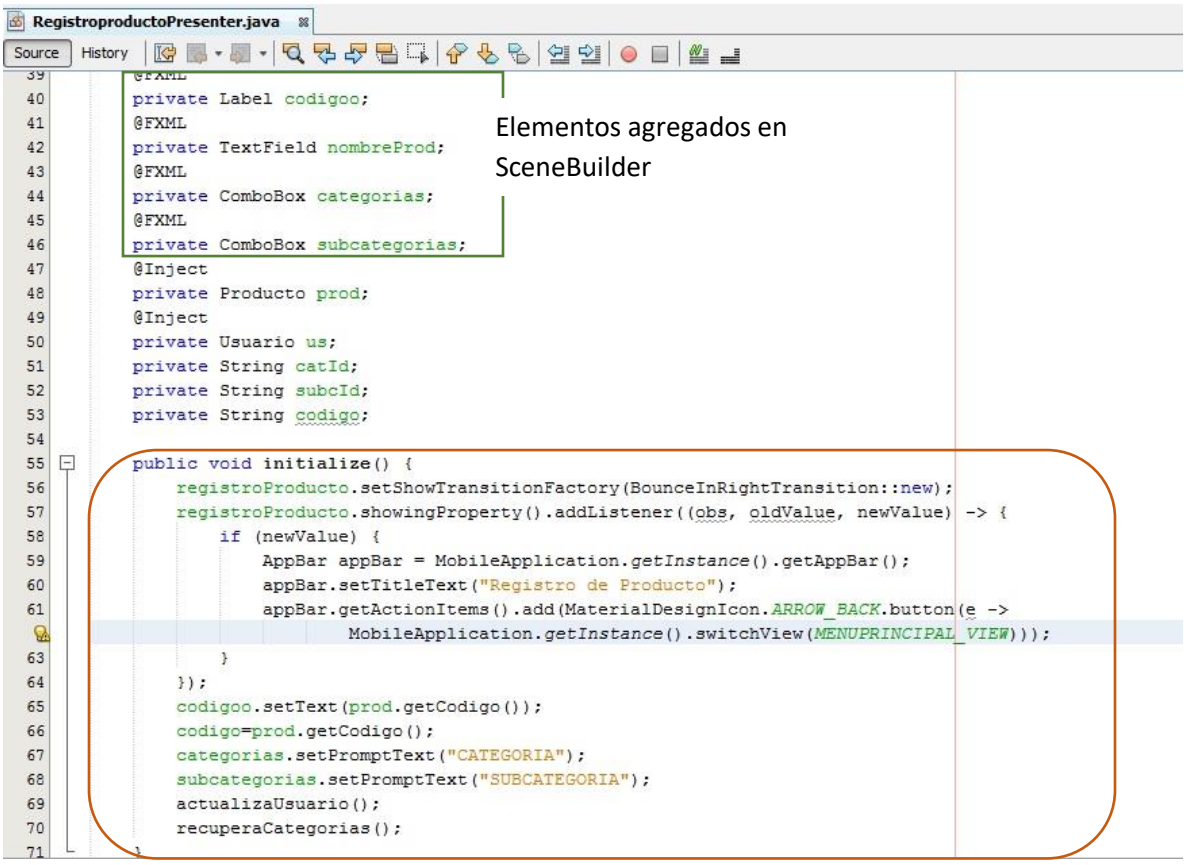
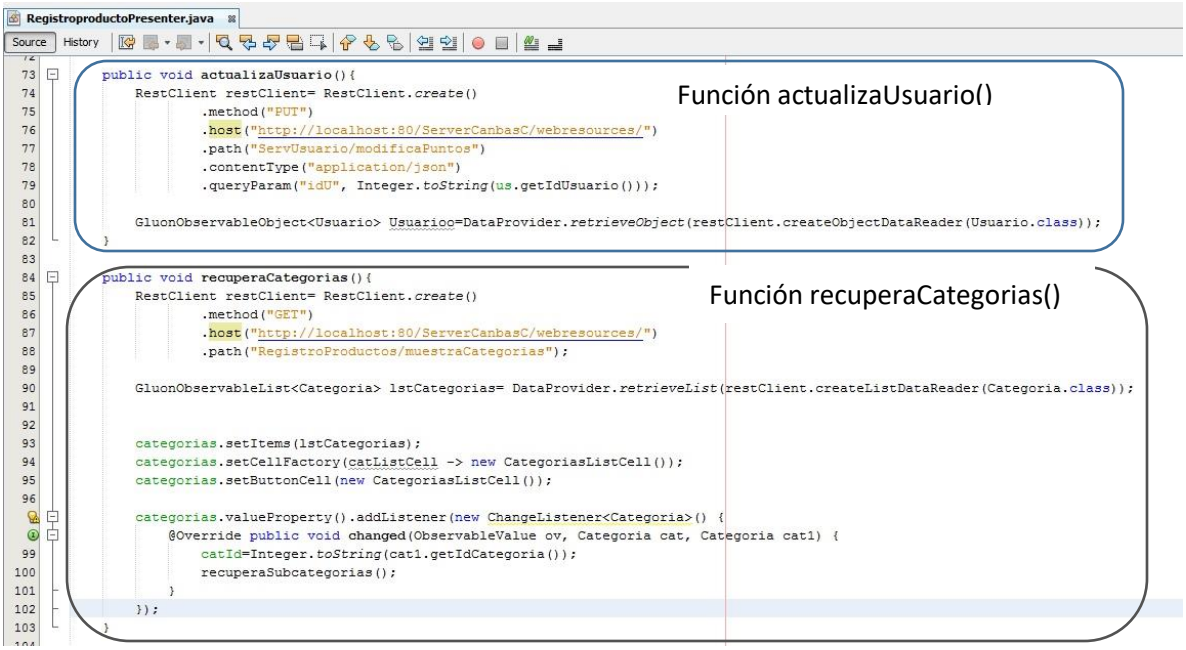


Figura 430 Código de la clase RegistroproductoPresenter

Se añadió un escuchador al método initialize() de la clase RegistroProductoPresenter para hacer más notorio el cambio entre vistas, se agregó también un escuchador a la vista para colocar un AppBar con el título "Registro de Producto" y un botón que permite volver a la vista de MenuPrincipal, ya fuera de este escuchador se colocaron los valores iniciales a ser mostrados en la vista, como al Label codigoo al que se le colocó el valor del código del producto a registrar, a cada ComboBox se le asignó un valor a mostrar para definir a que hace referencia, en este caso a las Categorías y Subcategorías, posteriormente se invocan las funciones actualizaUsuario() y recuperaCategorias().

A continuación se muestra el código de las funciones actualizaUsuario() y recuperaCategorias(), ver Figura 51.




```
public void actualizaUsuario(){
    RestClient restClient= RestClient.create()
            .method("PUT")
            .host("http://localhost:80/ServerCanbasC/webresources/")
            .path("ServUsuario/modificaPuntos")
            .contentType("application/json")
            .queryParam("idU", Integer.toString(us.getIdUsuario()));

    GluonObservableObject<Usuario> Usuarioo=DataProvider.retrieveObject(restClient.createObjectDataReader(Usuario.class));
}

public void recuperaCategorias(){
    RestClient restClient= RestClient.create()
            .method("GET")
            .host("http://localhost:80/ServerCanbasC/webresources/")
            .path("RegistroProductos/muestraCategorias");

    GluonObservableList<Categoria> lstCategorias= DataProvider.retrieveList(restClient.createListDataReader(Categoria.class));


    categorias.setItems(lstCategorias);
    categorias.setCellFactory(catListCell -> new CategoriasListCell());
    categorias.setButtonCell(new CategoriasListCell());

    categorias.valueProperty().addListener(new ChangeListener<Categoria>() {
        @Override public void changed(ObservableValue ov, Categoria cat, Categoria cat1) {
            catId=Integer.toString(cat1.getIdCategoria());
            recuperaSubcategorias();
        }
    });
}
```

Figura 441 Función actualizaUsuario() y recuperaCategorias() de la clase RegistroproductoPresenter

La función actualizaUsuario() es utilizada para agregar más puntos al usuario por el registro del producto y con ello aumentar su credibilidad, por lo que esta función hace uso de una petición al Servidor vía REST haciendo uso de RestClient y su función create(), method en este caso PUT, host con la dirección del servidor (http://localhost:80/ServerCanbasC/webresources/), path la dirección relativa al servicio y función a invocar ("servUsuario/modificaPuntos"), contentType para especificar el tipo de contenido a enviar en este caso el objeto usuario en formato JSON ("application/Json") y por último el queryParam el identificador de usuario en este caso el cast de Integer a String con el id de usuario. Para el envío correcto del objeto se hizo uso de GluonObservableObject<usuario> y de DataProvider y su función retrieveObject.

Por su parte la función recuperaCategorias() hace una petición al Servidor para obtener las Categorías registradas, para ello hace uso de RestClient y GluonObservableList<Categoria> y de DataProvider.retrieveList() para obtener la lista de categorías, una vez obtenidas son colocadas en el ComboBox; para mostrar solamente el nombre de las categorías obtenidas se creó una clase CategoriasListCell la cual es similar a UbicacionesListCell mostrada previamente en el apartado 4.4.5 de la vista Agregar Ubicación, posteriormente se agregó un escuchador al ComboBox para permitir la obtención de subcategorías a partir de la categoría mostrada, esto

empleando el segundo ComboBox y haciendo un llamado a la función recuperaSubcategorias(), ver Figura 52.

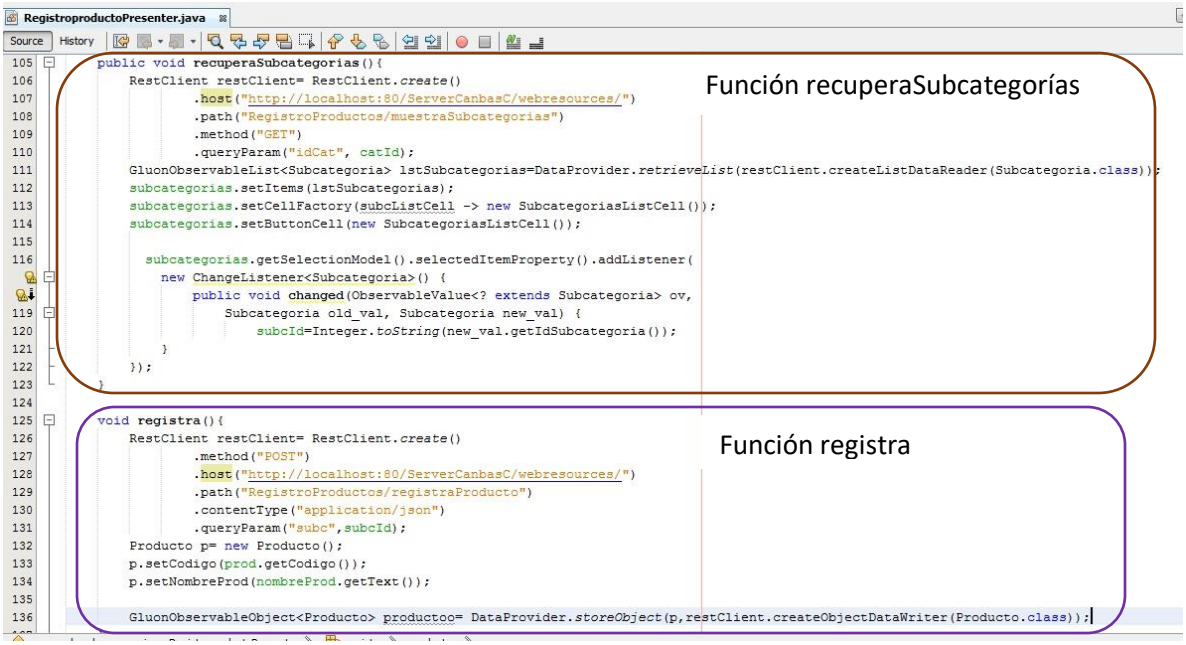


Figura 52 Función recuperaSubcategorias() y registra() de la clase RegistroproductoPresenter

La función encargada del registro de producto es registra(), ver figura 52, la cual hace uso de RestClient colocando como parámetro la subcategoría a la que pertenece el producto y enviando a tráves de GluonObservableObject<producto> y DataProvider.storeObject() el objeto creado con los valores proporcionados por el usuario, esta función es invocada por la acción "RegistrarProducto" del botón REGISTRAR, la cual una vez hecho el registro hace el cambio a la vista OpcionesTienda.

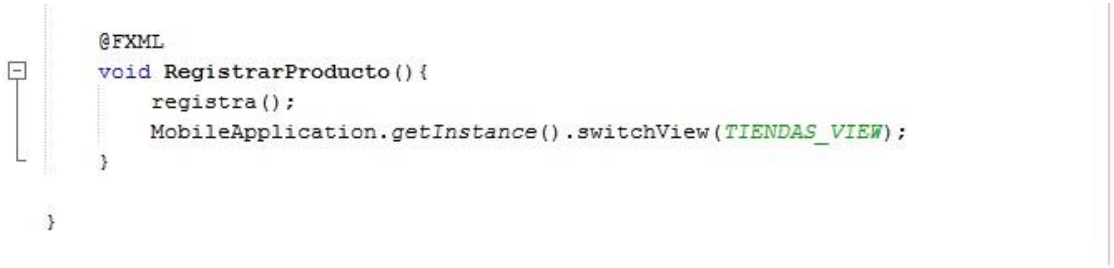


Figura 453 Funcionalidad del botón Registra de la vista regristroproducto

Opciones Tienda

Esta vista permite el registro de un producto en determinada tienda, introduciendo el precio que tiene en esta, para ello hace uso de un ComboBox que muestra las tiendas que ha registrado el usuario en la

ubicación previamente seleccionada en la vista Perfil, un TextField que permite introducir el precio y dos botones uno para registrar la Tienda si es que no se tiene registro de esta y otro para hacer el registro del producto en la tienda seleccionada, el diseño de esta vista se muestra en la Figura 54.

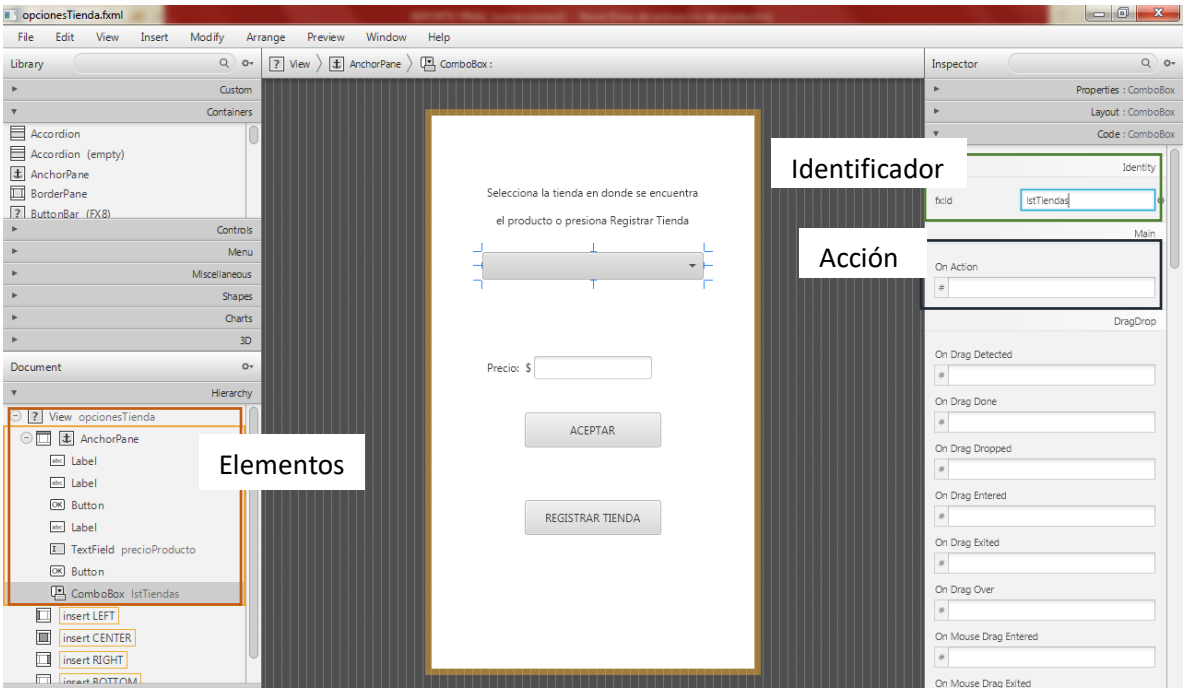
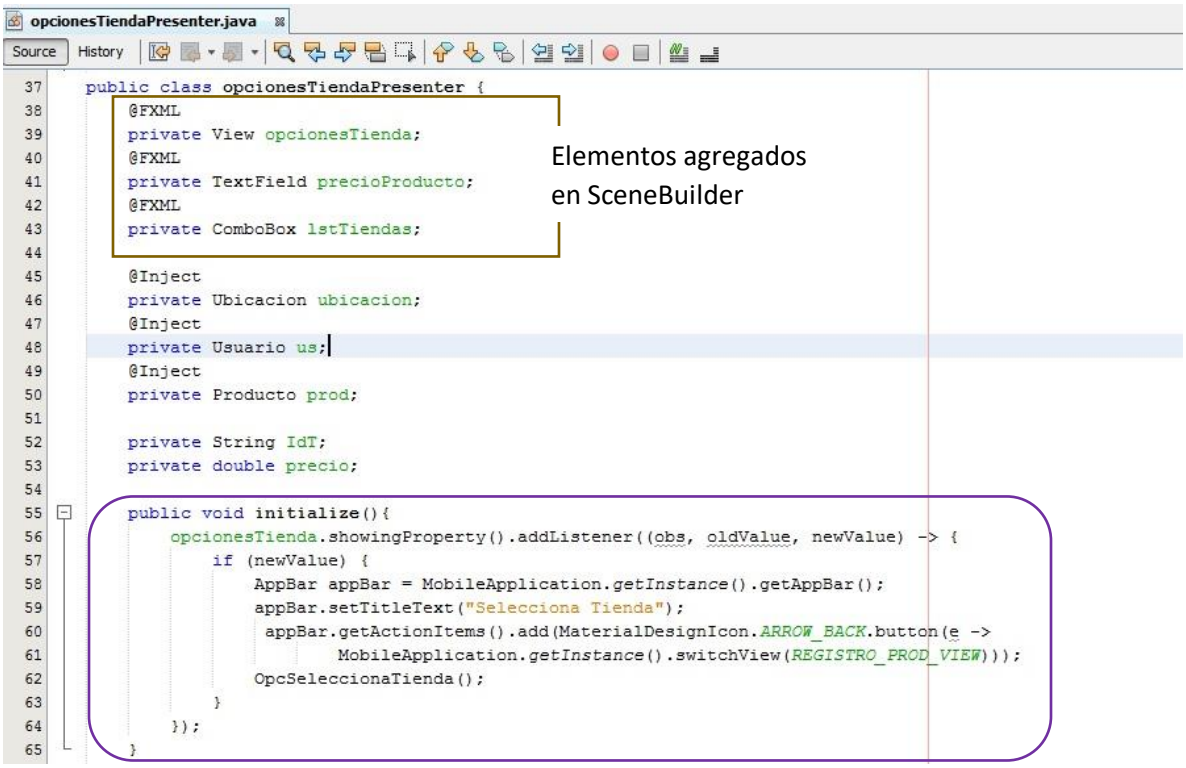


Figura 464 Diseño de la vista opcionesTienda en SceneBuilder

Para manipular correctamente algunos de los elementos añadidos se les colocó un identificador.

En la clase opcionesTiendaPresenter, ver Figura 55, se agregaron los elementos previamente mencionados, además se agregaron un objeto ubicación, uno de tipo usuario y uno de tipo producto, con notación @Inject, puesto que estos objetos son provenientes de otras vistas, la ubicación es la seleccionada desde la vista Perfil, el usuario es el que ha ingresado a la aplicación en la vista de Login y el producto es el que se registró previamente en la vista registro producto.

En el método initialize() se colocó un escuchador a la vista para mostrar cada que se cargue un AppBar con el título “Selecciona Tienda”, un botón que permite regresar a la vista anterior y por último la invocación a la función opcSeleccionaTienda().




```
37  public class opcionesTiendaPresenter {
38      @FXML
39      private View opcionesTienda;
40      @FXML
41      private TextField precioProducto;
42      @FXML
43      private ComboBox lstTiendas;
44
45      @Inject
46      private Ubicacion ubicacion;
47      @Inject
48      private Usuario us;
49      @Inject
50      private Producto prod;
51
52      private String IdT;
53      private double precio;
54
55      public void initialize(){
56          opcionesTienda.showingProperty().addListener((obs, oldValue, newValue) -> {
57              if (newValue) {
58                  AppBar appBar = MobileApplication.getInstance().getAppBar();
59                  appBar.setTitleText("Selecciona Tienda");
60                  appBar.getActionItems().add(MaterialDesignIcon.ARROW_BACK.button(e ->
61                          MobileApplication.getInstance().switchView(REGISTRO_PROD_VIEW)));
62                  OpcSeleccionaTienda();
63              }
64          });
65      }
```

Figura 475 Código inicial de la clase opcionesTiendaPresenter

La clase opcionesTiendaPresenter hace uso de dos funciones, la primera OpcSeleccionaTienda(), ver Figura 56, hace una petición al servidor para obtener la lista de tiendas registradas en la ubicación actual del usuario, para ello hace uso de un objeto RestClient y sus métodos como create() que permite crear una instancia de este, host, la dirección del Servidor (“http://localhost:80/ServerCanbasC/webresources/”), path, la dirección del Servicio a invocar y su respectiva función (“RegistroProductos/muestraTiendas”), method “GET” y queryParam en este caso el id de la ubicación, obteniendo este del objeto ubicación con notación @Inject, una vez recuperadas las tiendas son colocadas en el ComboBox con identificador lstTiendas, para mostrar correctamente el nombre de cada tienda en cada opción del ComboBox se creó una clase TiendasListCell, posteriormente se agregó un escuchador a este para obtener el id de tienda seleccionada el cual será utilizado por la función regProd() para el correcto registro del producto.

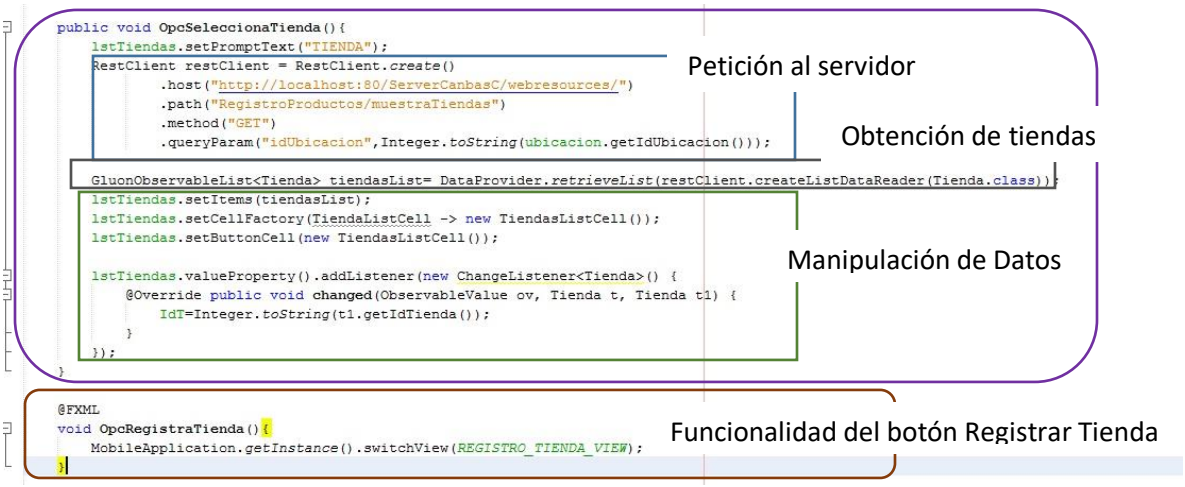


Figura 486 Función OpcSeleccionaTienda() y funcionalidad del botón Registrar Tienda de la clase RegistroproductoPresenter

Cabe señalar que la función OpcSeleccionaTienda() es llamada desde el método initialize() para permitir con ello la actualización del ComboBox cada que se cargue la vista.

Por su parte el botón de Registrar Tienda solo tiene la funcionalidad de hacer el cambio a la vista registroTienda, haciendo uso de MobileApplication.getInstance().switchView(REGISTRO_TIENDA_VIEW).

La segunda función de la clase opcionesTiendaPresenter es regProd(), ver Figura 57, la cual permite el envío de un objeto de tipo ProductoRegistrado al servidor para crear un registro de este en la tabla correspondiente añadiendo la tienda en que se encuentra y su precio, para ello se hizo uso de un objeto RestClient y sus respectivos métodos como create() el cual permite crear una instancia de este, host la dirección del servidor (“http://localhost:80/ServerCanbasC/webresources/”), path la dirección del Servicio a invocar con su respectiva función (“RegistraProductos/almacenaProductoenTienda”), method (“POST”), contenType el tipo de dato a enviar (“application/JSON”) y tres queryParam, el primero es el id del usuario que realiza el registro, el segundo el código del producto a registrar y el tercero la tienda en que se encuentra, posteriormente para enviar el ProductoRegistrado se crea una instancia de este colocando a través del método set el valor obtenido del TextField precioProducto, posteriormente se hizo uso de GluonObservableObject<ProductoRegistrado> y de DataProvider con su método storeObject proporcionando a este como parámetros el objeto a enviar y el tipo de este.

Figura 497 Función regProd() y funcionalidad del botón Registrar Producto de la clase RegistroproductoPresenter

La funcionalidad del botón Registrar Producto es llamar a la función regProd() y con ello asegurar el registro del producto en tienda, posteriormente se encarga de realizar el cambio a la vista registroexitoso.

Registro Tienda

Esta vista permite hacer el registro de una tienda en la ubicación previamente seleccionada por el usuario en la vista Perfil, para ello hace uso de GoogleMapView para mostrar el mapa, un Label para mostrar el nombre de la ubicación y dos TextField uno para el nombre de la tienda y el otro para la dirección de esta, el diseño de esta vista se muestra en la Figura 58. Para el manejo correcto de los elementos utilizados en la clase registroTiendaPresenter les fue colocado a cada uno un identificador.

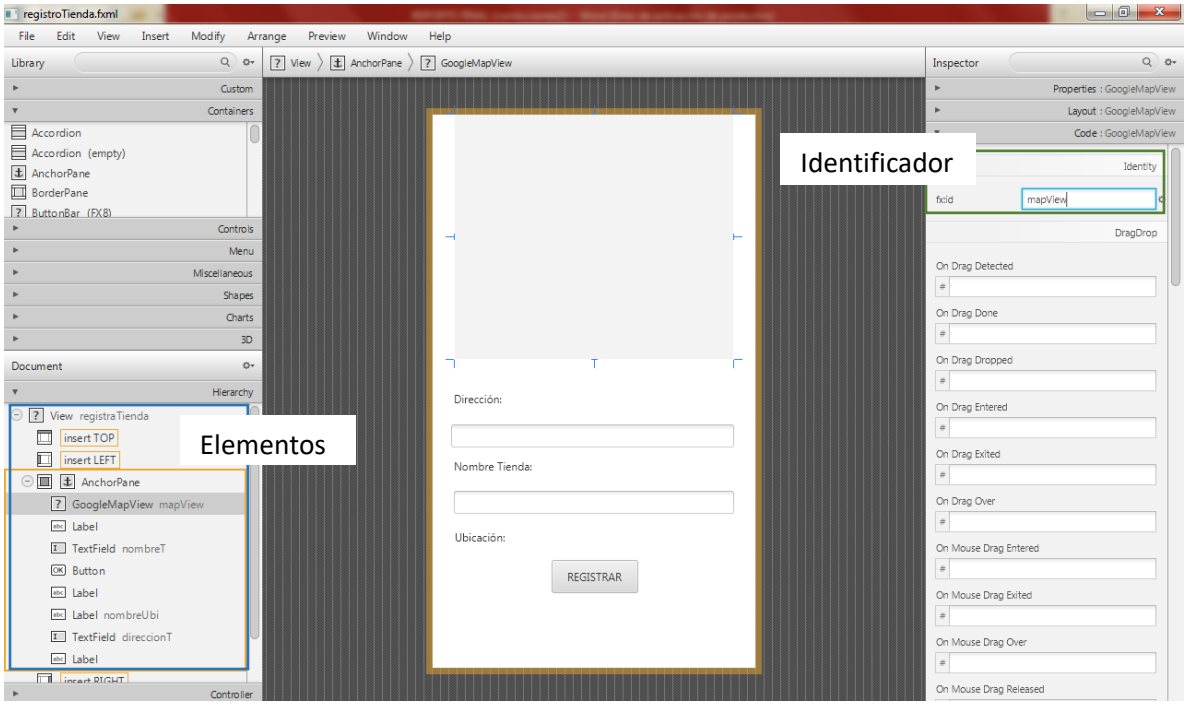


Figura 508 Diseño de la vista registroTienda en SceneBuilder

Para implementar cada una de las funciones de los elementos agregados en SceneBuilder [22] se hizo uso de la clase registroTiendaPresenter, mostrada en la Figura 59, esta clase hace uso del servicio de geocodificación proporcionado por el API GMapsFX[21], para ello el usuario debe ingresar correctamente la dirección de la tienda a registrar para que este pueda convertirla en coordenadas (latitud y longitud) y poder hacer así el correcto registro de la tienda.

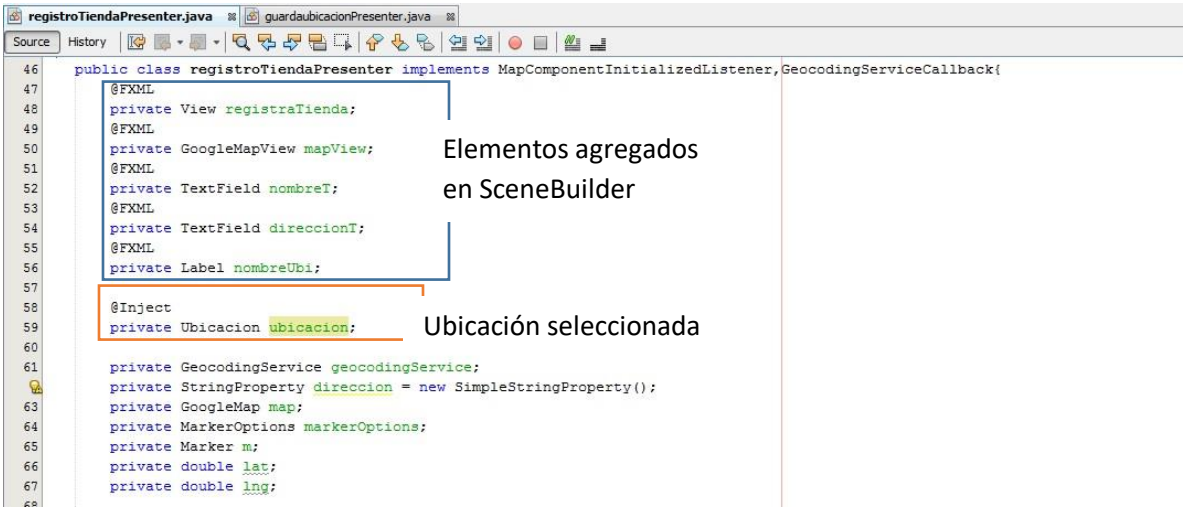


Figura 59 Atributos de la clase registroTiendaPresenter

La clase registroTiendaPresenter tiene dos métodos importantes: el método initialize(), dentro del cual se agregó una transición a la vista para hacer más notorio el cambio entre estas, posteriormente se agregó un escuchador haciendo uso del condificional if(newvalue) para mostrar cada vez que esta se cargue, un AppBar con el título "Registra Tienda" y con un botón que permite volver a la vista opcionesTienda, dentro de este escuchador se fija el valor del Label nombreUbi, al cual se le asignó el valor del nombre del objeto ubicación con la notación @Inject, además se agregó código perteneciente al GoogleMapView para asegurar que se reinicie la vista del mapa, fuera de este condicional se agregó un escuchador para la vista del mapa haciendo uso de mapView.addMapInitializedListener(this), además de enlazar el String dirección con él TextField que hace referencia a esta. Por otra parte el segundo método importante es mapInitialized() el cual es utilizado para inicializar la vista del mapa con ciertos parámetros tales como la posición en que se mostrará el mapa haciendo uso del objeto LatLong, el tipo de mapa en este caso ROADMAP que muestra la vista de este en 2D, el zoom colocado en 12 y valores como overviewMapControl, panControl, rotateControl, scaleControl, steetViewControl, los cuales fueron desactivados a través del parámetro false, ya que solo se pretende mostrar una vista simple del mapa sin tanta funcionalidad extra.

Cabe mencionar que en algunas pruebas se mostraba la vista del mapa por unos segundos y posteriormente mostraba un mensaje de error, este problema se resolvió agregando el código mostrado en la Figura 60, el cual hace que cada que se mande llamar esta vista cargue de nuevo el mapa.

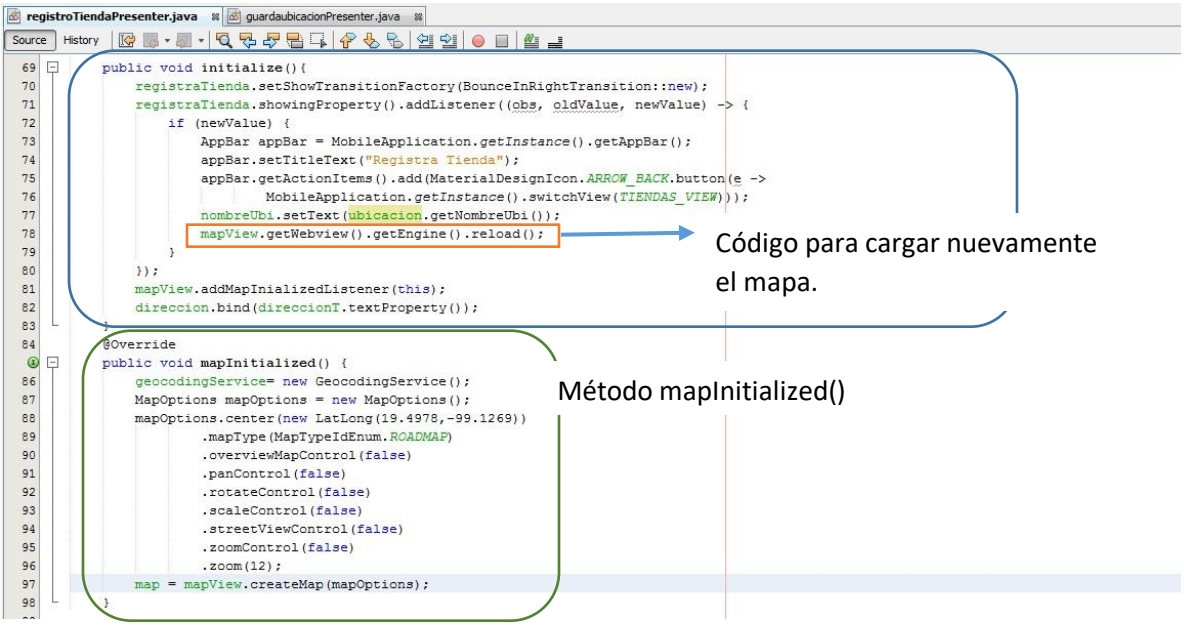


Figura 510 Métodos initialize() y mapInitialized() de la clase registroTiendaPresenter

La función que se encarga de realizar la geocodificación, es actionGeocode(), ver Figura 61, la funcionalidad de esta es similar al del código actionGeocode() en la clase guardaUbicacionPresenter, ver apartado 2.5.5 de esta sección, con la diferencia de que en este caso el objeto a enviar al servidor es una Tienda y no una ubicación y se envía además como queryParam el id de la ubicación en que se encuentra el usuario registrando esta.

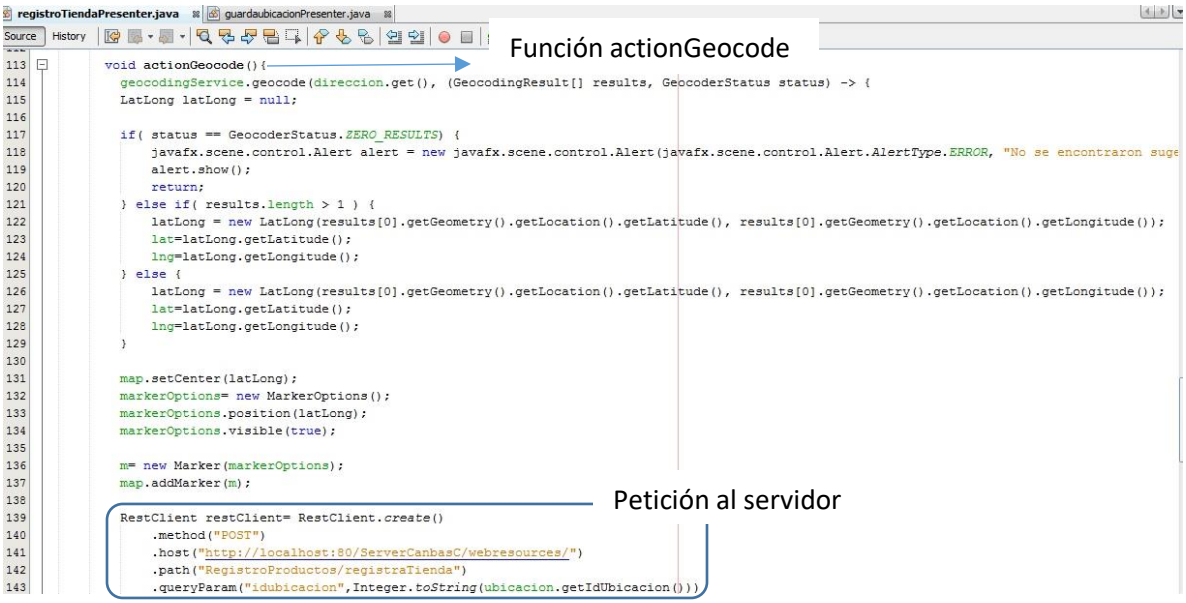


Figura 521 Código de la función actionGeocode de la clase registroTiendaPresenter

El botón que se encarga de invocar la función actionGeocode es Registrar y para ello se hace uso del código mostrado en la Figura 62, en el cual lo primero es invocar esta función para enviar los datos de la tienda al servidor, posteriormente muestra una alerta indicando que el registro fue exitoso, se limpian los campos de dirección y nombre colocándolos en blanco y por último hace el cambio a la vista opcionesTienda, en dónde esta tienda ya aparecerá dentro del ComboBox lstTiendas para que el usuario pueda seleccionarla.

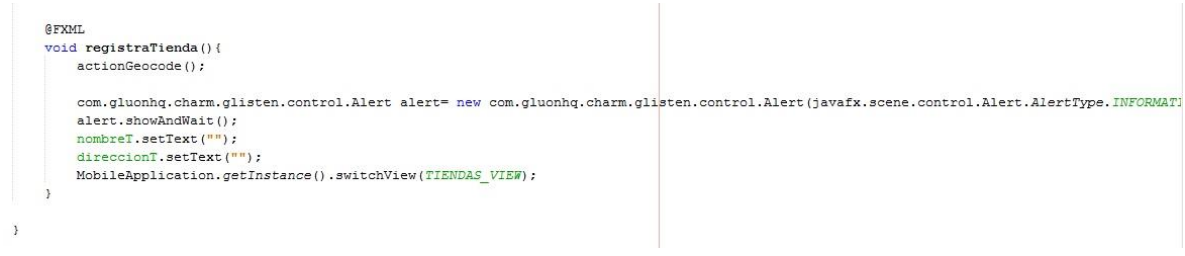


Figura 532 Funcionalidad del botón Registrar de la vista registroTienda

Registro Exitoso

Esta es la vista final del proceso de registrar un producto, ver Figura 63, por lo que permite al usuario seleccionar alguna acción entre volver al menú o registrar otro producto.

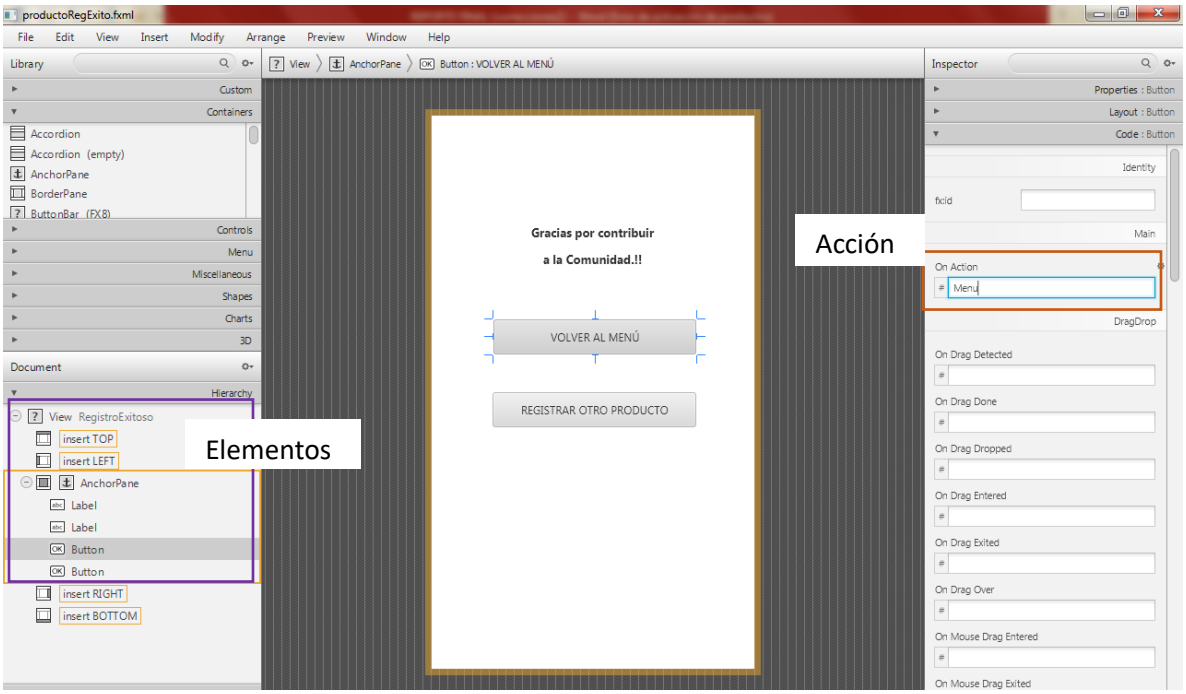


Figura 543 Diseño de la vista productoRegExito en SceneBuilder

Dado que esta vista solo tiene dos botones para llevar a cabo sus tareas no fue necesario añadirle un identificador a cada uno, pero si una acción, estas son implementadas en la clase productoRegExitoPresenter, como se muestra en la Figura 64.

```
public class productoRegExitoPresenter {

    @FXML
    private View RegistroExitoso;

    public void initialize(){
        RegistroExitoso.setShowTransitionFactory(BounceInRightTransition::new);
    }

    @FXML
    void Menu(){
        MobileApplication.getInstance().switchView(MENUPRINCIPAL_VIEW);
    }

    @FXML
    void RegistrarProducto(){
        MobileApplication.getInstance().switchView(REGISTRO_PROD_VIEW);
    }
}
```

Figura 554 Funcionalidad del botón Volver al Menú y de Registrar otro Producto de la vista productoRegExito

Funcionalidad del botón Volver al Menú: Cambia a la vista AgregarUbicación haciendo uso de MobileApplication.getInstance().switchView(MENUPRINCIPAL_VIEW).

Funcionalidad del botón Registrar otro Producto: Cambia a la vista AgregarUbicación haciendo uso de MobileApplication.getInstance().switchView(REGISTRO_PROD_VIEW).

2.5.7.- Búsqueda de Canasta

Para llevar a cabo cada una de las funciones de búsqueda de canasta se hizo uso de cinco vistas, las cuáles se explican a continuación:

Productos Disponibles

Esta vista muestra las subcategorías de las que el usuario puede seleccionar alguna para visualizar los productos dentro de esta y elegir aquellos que desea agregar a su canasta, para ello se hizo uso de un SplitPane vertical, el cual permite agrandar el tamaño de uno de sus lados para poder visualizar bien el nombre de las subcategorías y de los productos, este cuenta con un AnchorPane izquierdo y un derecho, el primero muestra las subcategorías y el segundo los productos encontrados en la subcategoría seleccionada, el diseño de esta vista se muestra en la Figura 65.

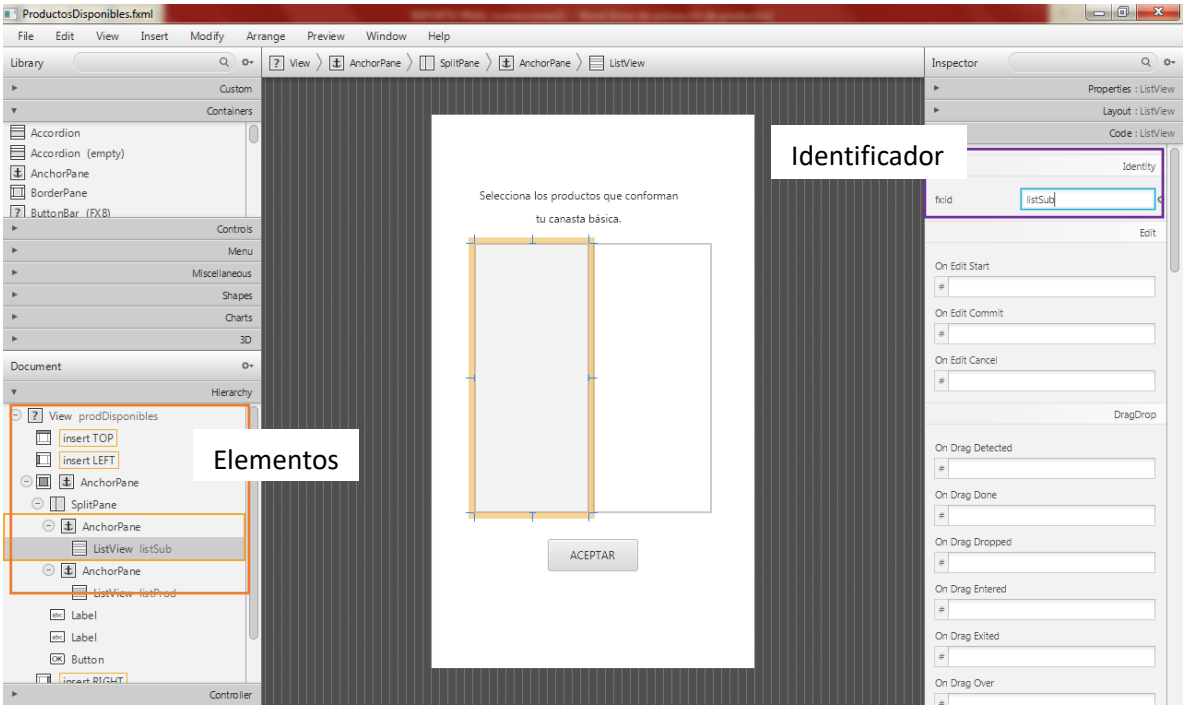
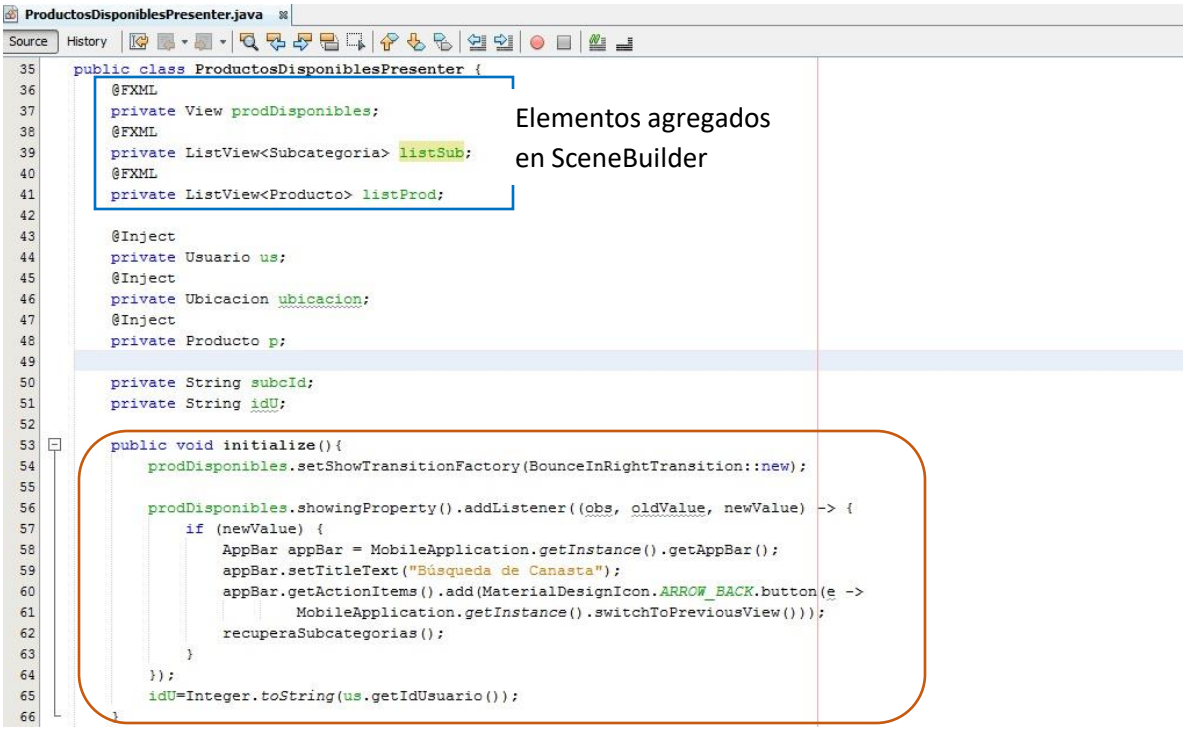


Figura 565 Diseño de la vista ProductosDisponibles en SceneBuilder

Para la correcta manipulación de los elementos se colocó a cada uno un identificador y al botón una acción.

La clase ProductosDisponiblesPresenter, ver Figura 66, hace uso de los elementos mencionados previamente para su correcta manipulación, para ello estos llevan la notación @FXML y su identificador, también se agregaron los objetos de tipo Usuario, Ubicación y Producto con la notación @Inject, puesto que estos pueden ser utilizados por otras vistas. Esta clase posee un método initialize() el cual se invoca cada que se carga la vista, dentro de este se agregó una transición a la vista y un escuchador el cual muestra un AppBar con el título "Búsqueda de Canasta" y un botón que permite volver a la vista de menú principal, además de llamar a la función recuperaSubcategorias() cada que se cargue la vista, ya fuera del escuchador se le coloca el valor de id de Usuario del objeto Usuario al String definido como idU.




```
35  public class ProductosDisponiblesPresenter {
36      @FXML
37      private View prodDisponibles;
38      @FXML
39      private ListView<Subcategoria> listSub;
40      @FXML
41      private ListView<Producto> listProd;
42
43      @Inject
44      private Usuario us;
45      @Inject
46      private Ubicacion ubicacion;
47      @Inject
48      private Producto p;
49
50      private String subcId;
51      private String idU;
52
53      public void initialize(){
54          prodDisponibles.setShowTransitionFactory(BounceInRightTransition::new);
55
56          prodDisponibles.showingProperty().addListener((obs, oldValue, newValue) -> {
57              if (newValue) {
58                  AppBar appBar = MobileApplication.getInstance().getAppBar();
59                  appBar.setTitleText("Búsqueda de Canasta");
60                  appBar.getActionItems().add(MaterialDesignIcon.ARROW_BACK.button(e ->
61                          MobileApplication.getInstance().switchToPreviousView()));
62                  recuperaSubcategorias();
63              }
64          });
65          idU=Integer.toString(us.getIdUsuario());
66      }
```

Figura 576 Código inicial de la clase ProductosDisponiblesPresenter

A continuación se explica la función recuperaSubcategorias(), ver figura 67:

La función recuperaSubcategorias() hace una petición al servidor para obtener la lista de subcategorías registradas en la base de datos, para ello hace uso del objeto RestClient y sus métodos, create() para crear una instancia de RestClient, host la dirección del servidor (http://localhost:80/ServerCanbasC/webresources), path la dirección del Servicio y su función a invocar (“BusquedaCanasta/muestraSubcategorias”) y por último method en este caso “GET”, posteriormente obtiene los resultados por medio de GluonObservableList<subcategorías> y de DataProvider.retrieveList(), colocándole como parámetro el tipo de objeto a recuperar, para ello se creó previamente el objeto subcategorías dentro del paquete CanbasCheapApp.POJOS. Para mostrar la lista al usuario se hizo uso de un ListView, al cual se le colocó la lista recuperada previamente y se creó una clase SubcategoriasListCell para mostrar en cada elemento de la lista solo el nombre de la subcategoría, posteriormente se le agregó un escuchador el cual permite recuperar los productos registrados en esta subcategoría invocando la función de recuperaProductos() y colocando al String subid el valor de la subcategoría seleccionada.

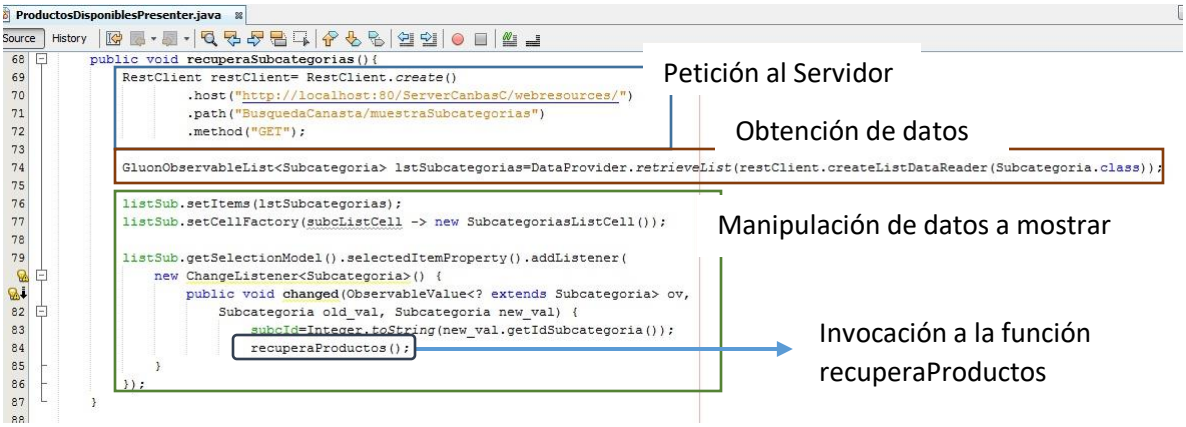


Figura 587 Código de la función recuperaSubcategorias() de la clase ProductosDisponiblesPresenter

La función recuperaProductos(), ver Figura 68, se encarga de obtener la lista de productos relacionados con la subcategoría seleccionada permitiendo al usuario seleccionar aquel que desee agregar a su lista de canasta básica, para llevar a cabo estas tareas hace una petición al servidor a través del objeto RestClient, similar a la de la función recuperaSubcategorías, solo que esta vez colocando como path (“BusquedaCanasta/muestraProductos”) y como queryParam el id de la subcategoría, posteriormente hace uso de GluonObservableList<Producto> y de DataProvider.retrieveList() para recuperar la lista de productos, una vez hecho esto se le asigna está a la ListView listProd, para ser mostrada al usuario se creó una clase ProductosListCell la cual permite mostrar en cada elemento de la lista el nombre del Producto, posteriormente se agrega un escuchador al ListView el cual cada que el usuario seleccione un producto realiza una instancia del objeto Producto y hace el cambio a la vista Cantidad Producto.

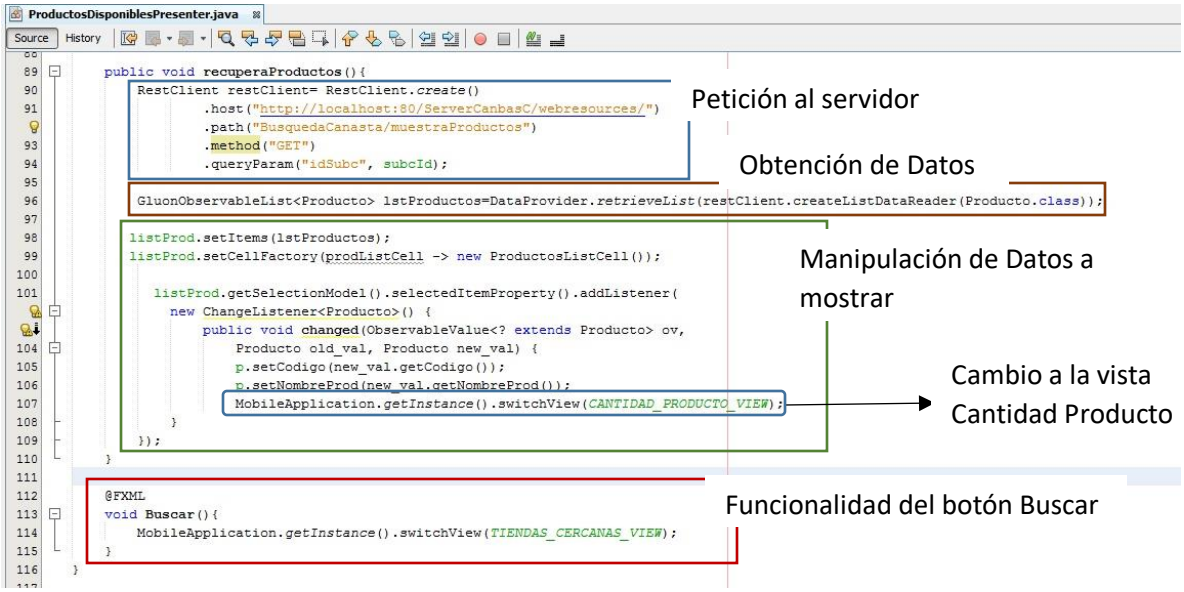


Figura 598 Código de la función recuperaProductos() y funcionalidad del botón Buscar de la clase ProductosDisponiblesPresenter

Por otra parte el botón de Buscar solo tiene como funcionalidad hacer el cambio a la vista Tiendas Cercanas, este botón debe ser presionado cuando el usuario ya haya terminado de elegir los productos que desea agregar a su lista de canasta básica.

Cantidad Producto

Esta vista permite al usuario ingresar la cantidad del producto seleccionado a agregar en su lista de canasta básica, para ello hace uso de un Label que muestra el nombre del producto seleccionado, un TextField para ingresar la cantidad y por último un botón que invoca la función para enviar estos datos al servidor, el diseño de esta vista en SceneBuilder [22] se muestra en la Figura 69.

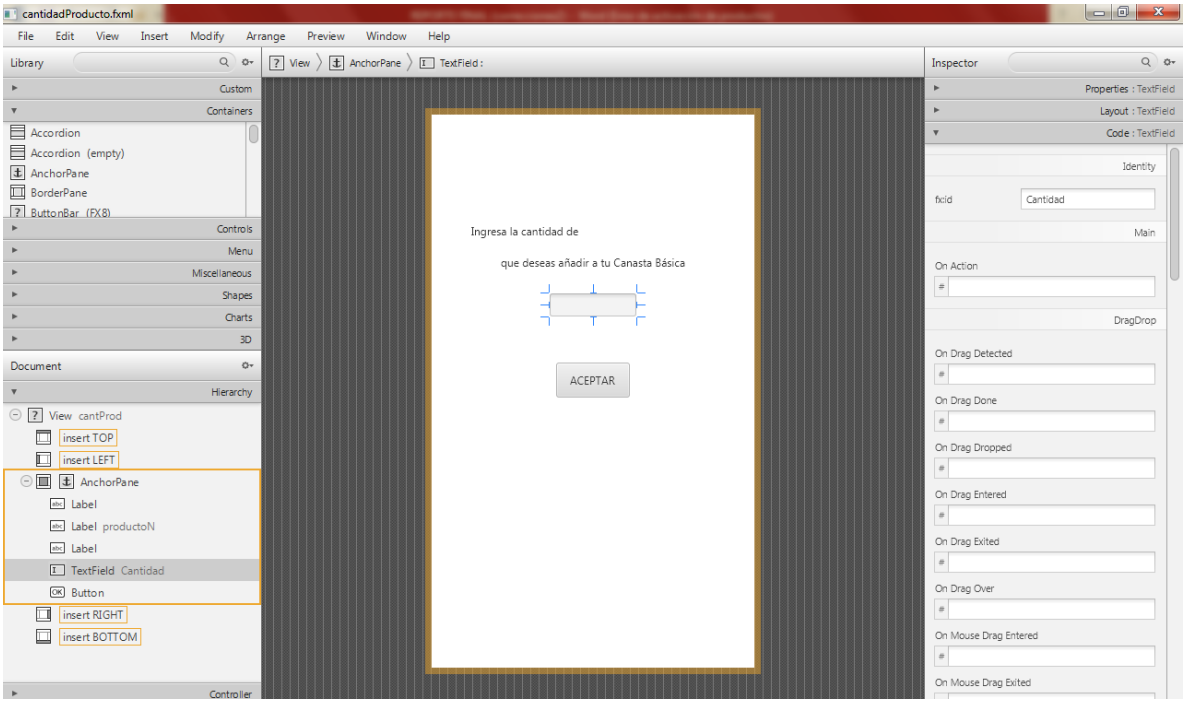


Figura 69 Diseño de la vista cantidadProducto en SceneBuilder

Para la correcta manipulación de los elementos agregados, le fue colocado a cada uno un identificador.

Para llevar a cabo las tareas que realiza cada elemento de la vista se creó la clase cantidadProductoPresenter, ver Figura 70, en esta los elementos son añadidos usando la notación @FXML, además de que se agregaron dos con notación @Inject, el usuario que es el que accedió a la aplicación desde la vista de Login y el producto, este se creó en la vista Productos Disponibles, por otra parte el método initialize() solo consta de una transición para hacer más notorio el cambio a esta vista, un escuchador que muestra cada que se carga la vista, un AppBar que contiene un título (“Cantidad de Productos”) y un botón que permite volver a la vista anterior (Productos Disponibles), dentro de este escuchador se fija también el valor a la etiqueta productoN, asignándole el nombre del producto proveniente de la vista anterior.


```
public class cantidadProductoPresenter {

    @FXML
    private View cantProd;

    @FXML
    private Label productoN;

    @FXML
    private TextField Cantidad;

    @Inject
    private Usuario us;

    @Inject
    private Producto p;

    public void initialize(){
        cantProd.setShowTransitionFactory(BounceInRightTransition::new);

        cantProd.showingProperty().addListener((obs, oldValue, newValue) -> {
            if (newValue) {
                AppBar appBar = MobileApplication.getInstance().getAppBar();
                appBar.setTitleText("Cantidad de Productos");
                appBar.getActionItems().add(MaterialDesignIcon.ARROW_BACK.button(e ->
                        MobileApplication.getInstance().switchToPreviousView()));
                productoN.setText(p.getNombreProd());
            }
        });
    }
```

Figura 600 Código inicial de la clase cantidadProductoPresenter

Esta vista hace el envío de cada producto seleccionado con la cantidad ingresada por el usuario para agregar este a una lista, para ello hace uso de la función productoSeleccionado, ver Figura 71, la cual recibe como parámetro el objeto producto generado desde la vista anterior, ProductosDisponibles, esta función es invocada por la acción del botón agregarProducto la cual una vez que ha hecho el envío del objeto Producto hace el cambio a la vista ProductosDisponibles permitiendo al usuario elegir más productos; a continuación se explica la función productoSeleccionado:

Esta función hace uso de RestClient para hacer la petición al servidor, para ello se agregaron las funciones de este, tales como create() para la creación de este, host la dirección del servidor (http://localhost:80/ServerCanbasC/webresources), path la dirección del servicio y su función a invocar (“BusquedaCanasta/agregaProductosLista”), method en este caso “POST”, contentType indicando el tipo de dato a enviar (“application/json”) y dos queryParam el primero hace referencia al id del usuario y el segundo a la cantidad ingresada por el usuario, para ello se recupera el valor ingresado por el TextField, posteriormente se hace uso de GluonObservableObject<Producto> y de DataProvider.storeObject() colocándole a este como parámetros el objeto a enviar (en este caso el objeto Producto creado desde la vista anterior (ProductosDisponibles)) y el tipo de este objeto.

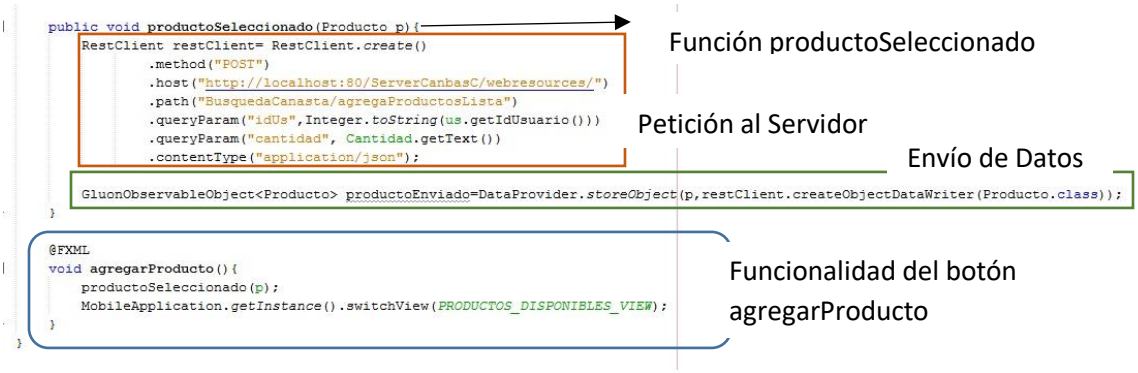


Figura 611 Código de la función productoSeleccionado y funcionalidad del botón agregarProducto de la clase cantidadProductoPresenter

Tiendas Cercanas

Esta vista permite hacer la búsqueda de tiendas que poseen todos los productos de la lista del usuario y que además se encuentran dentro del rango de distancia proporcionado por este, el diseño de esta vista en SceneBuilder [22] se muestra en la Figura 72.

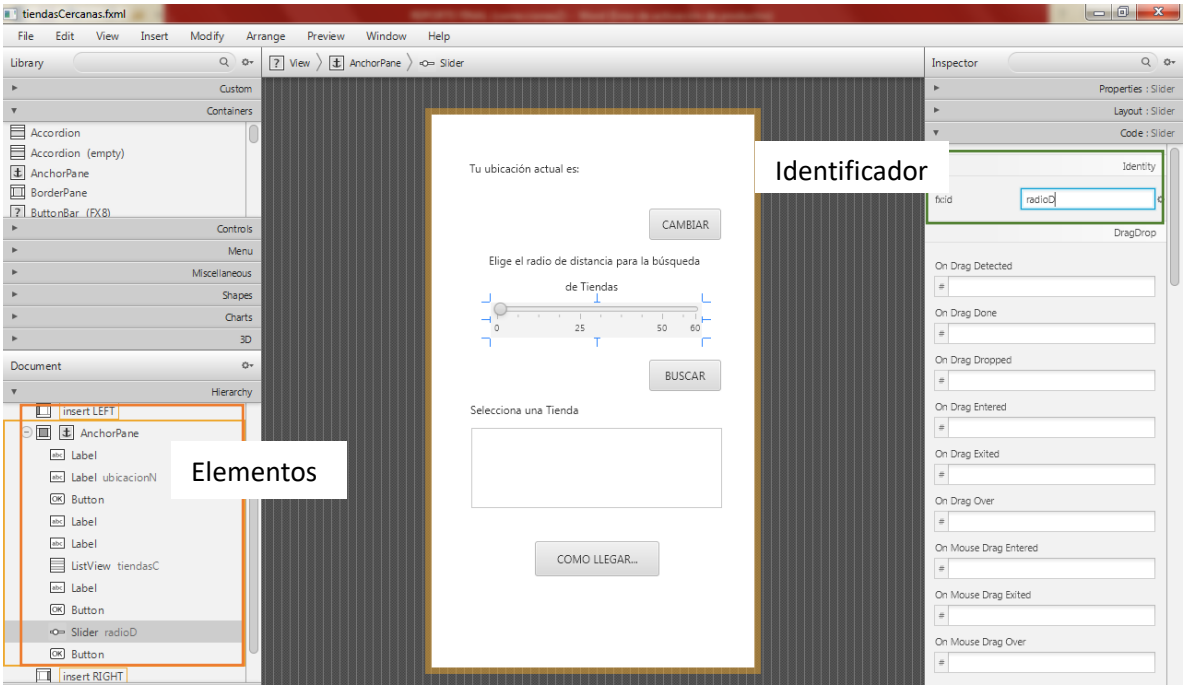


Figura 622 Diseño de la vista tiendasCercanas en SceneBuilder

A continuación se explica el propósito de cada uno de sus elementos:

Esta vista se puede dividir en tres funciones, la primera consiste en permitir al usuario visualizar la ubicación seleccionada en la vista Perfil y si lo desea, cambiar esta por alguna otra, para ello hace uso de un Label que muestra la ubicación y un botón con la acción para cambiarla. La segunda es que

permite al usuario seleccionar el radio de distancia en el que desea realizar la búsqueda de tiendas que contengan todos los productos que ha agregado a su lista, para ello se utilizó un Slider y un botón de Buscar. La tercera función muestra las tiendas encontradas, permitiéndole al usuario elegir alguna, para ello hace uso de un ListView y un botón de Como Llegar, la clase que se encarga de implementar cada una de estas funcionalidades y proveer alguna acción a los elementos es tiendasCercanasPresenter mostrada en la Figura 73.


```
public class tiendasCercanasPresenter {
    @FXML
    private View tiendasCercanas;
    @FXML
    private Label ubicacionN;
    @FXML
    private Slider radioD;
    @FXML
    private ListView tiendasC;

    @Inject
    private Ubicacion ubicacion;
    @Inject
    private Tienda tiendaa;
    @Inject
    private Usuario us;

    private int radio;

    public void initialize(){
        tiendasCercanas.setShowTransitionFactory(BounceInRightTransition::new);
        tiendasCercanas.showingProperty().addListener((obs, oldValue, newValue) -> {
            if (newValue) {
                radioD.setValue(0.0);
            }
        });
        ubicacionN.setText(ubicacion.getNombreUbi());
    }

    @FXML
    void Perfil(){
        MobileApplication.getInstance().switchView(PERFIL_USUARIO_VIEW);
```

Figura 633 Código inicial de la clase tiendasCercanasPresenter

La clase tiendasCercanasPresenter hace uso de los elementos previamente mencionados, además de un objeto de tipo ubicación, uno de tipo tienda y otro de usuario con la notación @Inject, esto porque son objetos que pueden ser usados en otras vistas. En su método initialize() la vista invoca una transición para hacer más notorio el cambio a esta, además de agregar un escuchador a la vista, colocando el valor inicial del Slider en 0.0, cada vez que se cargue esta, ya fuera de este escuchador se coloca en el Label ubicacionN, el valor de esta proveniente del objeto ubicación seleccionado en la vista Perfi.

Para permitir al usuario hacer el cambio entre vistas el botón Cambiar tiene la acción “Perfil()” asociada a él en la que se hace el cambio a la vista de Perfil de usuario para que este pueda seleccionar alguna otra ubicación.

El código de la función que permite realizar la búsqueda de tiendas cercanas se muestra en la Figura 74.

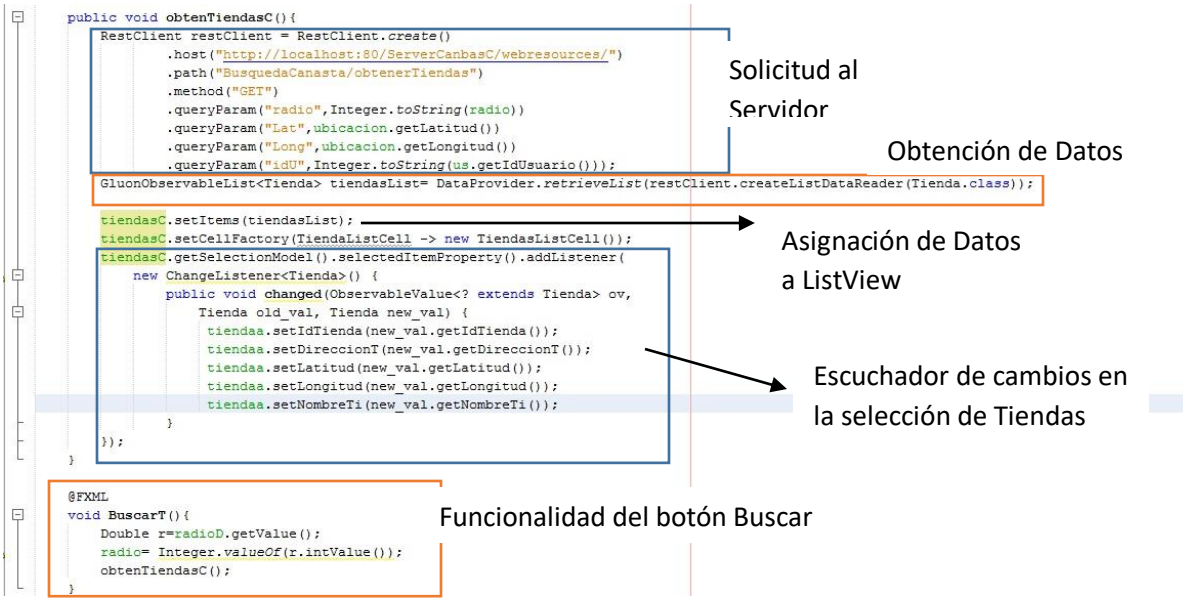


Figura 644 Código de la función obtenTiendasC y funcionalidad del botón Buscar de la clase tiendasCercanasPresenter

La función obtenTiendasC utiliza el objeto RestClient para hacer la petición al servidor, colocándole para ello ciertos parámetros en sus funciones, host la dirección del servidor (http://localhost:80/ServerCanbasC/webresources/), path la dirección del servicio y su función a invocar ("BusquedaCanasta/obtenTiendas"), method en este caso "GET" y cuatro queryParam, el primero indicando el radio elegido por el usuario, el segundo y tercer o hacen referencia la latitud y la longitud de la ubicación seleccionada y por último el cuarto es el id de usuario, posteriormente hace uso de GluonObservableList<Tienda> para recuperar la lista de tiendas que poseen todos los productos seleccionados por el usuario y que además están dentro del rango de distancia seleccionado, esta lista es asignada al ListView tiendasC y para controlar lo que será mostrado en cada elemento de la lista se creó una clase TiendasListCell la cual permite mostrar solo el nombre, por otra parte se añadió también un escuchador a esta, permitiendo con ello conocer la tienda seleccionada por el usuario y asignarla al objeto Tienda con notación @Inject ya que este será utilizado en la vista Como Llegar. La acción BuscarT del botón Buscar es la que hace la llamda a esta función, para ello primero recupera el valor del radio seleccionado haciendo el cast de Double a Integer.

Por último, la acción del botón Como Llegar se muestra en la Figura 75, la cual solamente hace el cambio a la vista Como llegar y reinicia el valor del Slider.

```
@FXML
void BuscarT(){
    Double r=radioD.getValue();
    radio= Integer.valueOf(r.intValue());
    obtenTiendasC();
}
```

Funcionalidad del botón Buscar

```
@FXML
void comoLlegar(){
    MobileApplication.getInstance().switchView(COMO_LLEGAR_VIEW);
    radioD.setValue(0.0);
}
}
```

Funcionalidad del botón Como Llegar

Figura 655 Funcionalidad del botón Como Llegar de la clase tiendasCercanasPresenter

Como Llegar

Esta vista permite al usuario visualizar la ruta a seguir para llegar a la tienda previamente seleccionada en la vista tiendas cercanas, permite también volver al menú o calcular el precio total de su lista de canasta básica. El diseño de esta vista en SceneBuilder se muestra en la Figura 76.

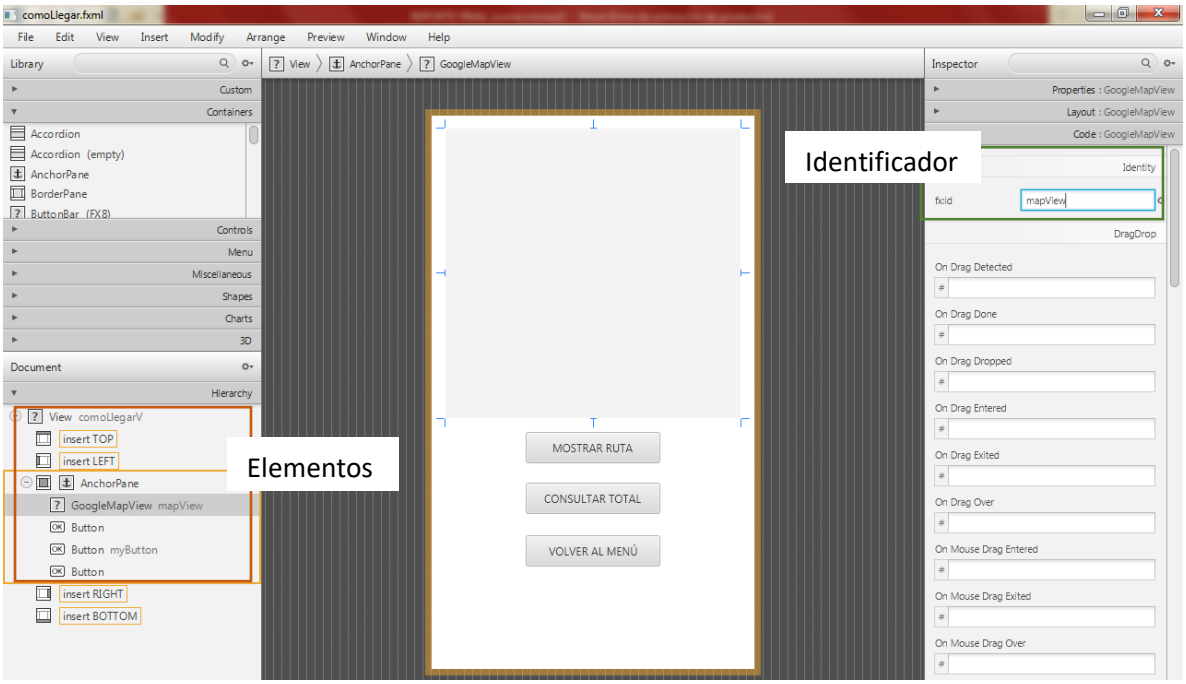


Figura 666 Diseño de la vista comoLlegar en SceneBuilder

Para la correcta funcionalidad de esta vista se hizo uso de un GoogleMapView para mostrar el mapa con la ruta que debe seguir el usuario, para invocar esta función se le colocó una acción al botón Mostrar Ruta, se utilizaron además otros dos botones con acciones como calcular total y volver al menú.

La clase que se encarga de implementar cada acción que realizan los elementos mencionados previamente es comoLlegarPresenter, ver Figura 77, a continuación se explica cómo está desarrollada cada acción de esta.

```
public class comoLlegarPresenter implements MapComponentInitializedListener, DirectionsServiceCallback{
    @FXML
    private View comoLlegarV;
    @FXML
    private GoogleMapView mapView;
    @FXML
    private Label nombreUbi;
    @FXML
    private Button myButton;

    @Inject
    private Ubicacion ubicacion;
    @Inject
    private Tienda tiendaa;

    protected DirectionsService directionsService;
    protected DirectionsPane directionsPane;
    private String origen;
    private String destino;

    public void initialize(){
        comoLlegarV.setShowTransitionFactory(BounceInRightTransition::new);
        origen=ubicacion.getDireccion();
        destino=tiendaa.getDireccionT();
        mapView.addMapInializedListener(this);
    }
```

Figura 677 Código inicial de la clase comoLlegarPresenter

Esta clase posee un método initialize() el cual carga los valores iniciales de los elementos de esta vista, en este caso este contiene: una transición para hacer más notorio el cambio a esta, posteriormente se inicializan los valores de los dos puntos en que se desea calcular la ruta, origen, recuperando el valor de la dirección de la ubicación previamente seleccionada en la vista Perfil, destino, el valor de la dirección de la tienda elegida en la vista anterior (tiendasCercanas) y por último se agregó un escuchador a la vista del mapa para colocar sus parámetros iniciales, por lo cual la clase comoLlegarPresenter implementa a MapComponentInitializedListener y a DirectionsServiceCallBack para hacer uso de DirectionsService de GMapsFX [21]. A continuación, se describe la funcionalidad de los tres botones de esta vista, ver Figuras 78 y 79:

- Mostrar Ruta: Se encarga de invocar la función de DirectionsRequest para mostrar la ruta desde la ubicación a la tienda seleccionada, para ello se crea una instancia del objeto DirectionsService y una de DirectionsPane, esto ocurre en el método mapInitialized() el cual permite colocar los valores iniciales a mostrar en el mapa, por medio de MapOptions, indicándole la posición del mapa con center, el zoom, el tipo de mapa y desactivando algunas funciones que no son requeridas para esta vista, Directions Request se instancia en el código de acción de este botón y recibe como parámetros la dirección de origen (ubicación), la dirección de destino (tienda) y por último el modo de viaje en este caso DRIVING, lo que indica que será en auto,

una vez hecho esto invoca al método getRoute() de DirectionsService, colocándole como parámetros el objeto request, previamente creado, DirectionsServiceCallBack y se crea una instancia de DirectionsRenderer el cual permite mostrar la ruta, para ello se hace visible colocando su primer paámetro con true y agregándolo a la vista de GoogleMapView y por último se invoca el objeto directionsPane creado en mapInitialized().

- Volver al Menú: Realiza el cambio a la vista de menú principal.
- Consultar Total: Realiza el cambio a la vista de Total.

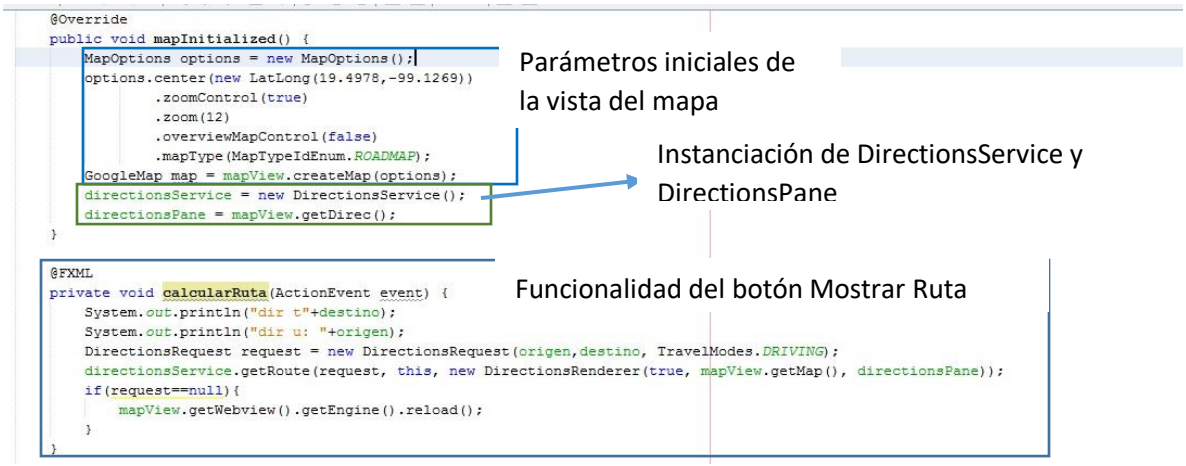


Figura 68 Método mapInitialized() y funcionalidad del botón Mostrar Ruta en la clase comoLlegarPresenter

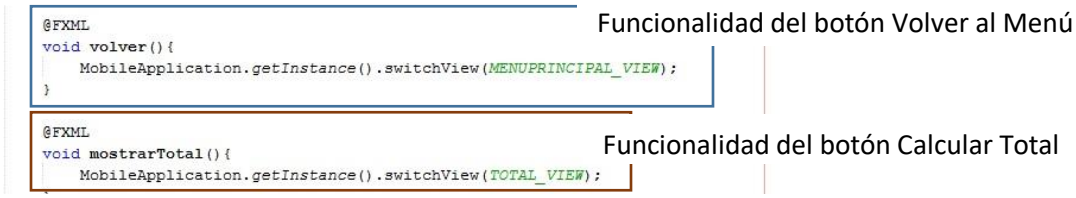


Figura 79 Funcionalidad del botón Volver al Menú y Calcular Total en la clase comoLlegarPresenter

Total

Esta vista muestra la lista de productos seleccionados y su precio unitario, además de mostrar el precio total de esta lista, para ello hace uso de dos ListView, una para mostrar los nombres de los productos y otro para mostrar los precios, además de un botón y un Label en que se muestra el total. El diseño de esta vista en SceneBuilder [22] se muestra en la Figura 80.

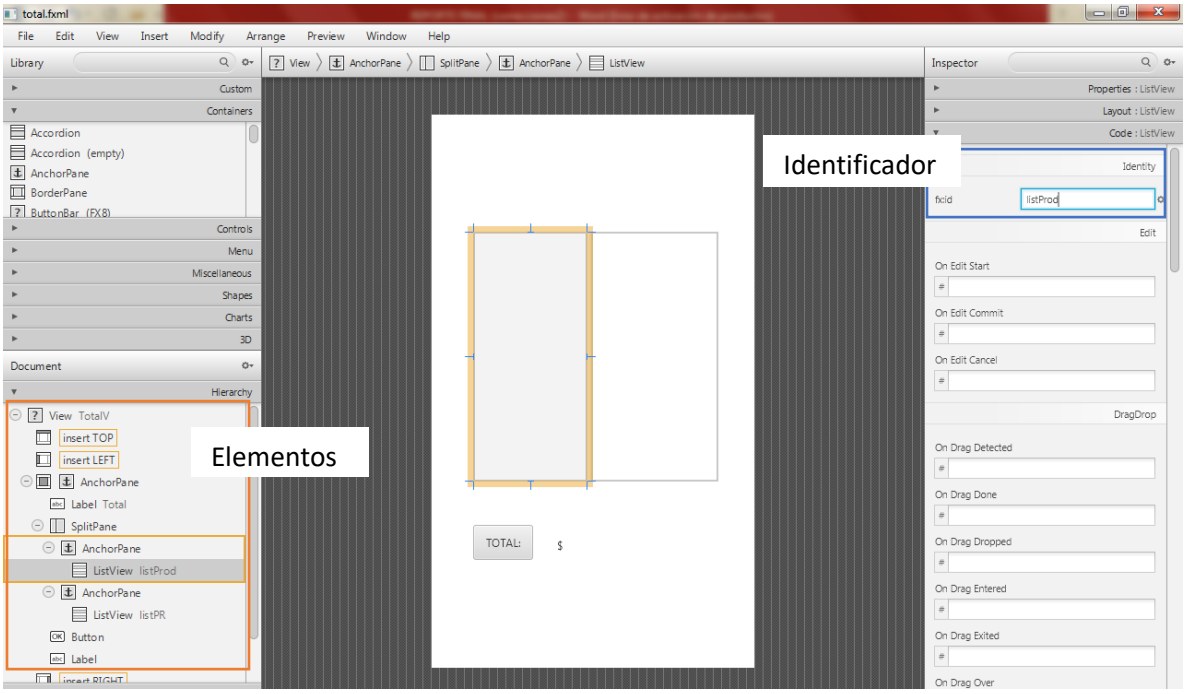


Figura 690 Diseño de la vista total en SceneBuilder

La clase que implementa cada una de las acciones de la vista es totalPresenter, ver Figura 81, esta clase hace uso de los elementos mencionados previamente, para ello se colocaron estos con notación @FXML, por otra parte hace uso de dos objetos con notación @Inject, usuario y tienda, en su método initialize() define una transición para esta vista, para hacer más notorio el cambio a esta, además de añadir un escuchador que permite mostrar un AppBar con un título ("Calculo Total") y un botón para volver a la vista anterior, posteriormente fuera de este escuchador se coloca el valor inicial de la variable total en 0 y se invocan las funciones vistaProd() y vistaProdReg().


```
public class totalPresenter {
    @FXML
    private View TotalV;
    @FXML
    private ListView<ProductoRegistrado> listPR;
    @FXML
    private ListView<Producto> listProd;
    @FXML
    private Label Total;

    @Inject
    private Usuario us;
    @Inject
    private Tienda tiendaa;

    private ObservableList<ProductoRegistrado> pr;
    private double total;
    private double sub=0.0;

    public void initialize(){
        TotalV.setShowTransitionFactory(BounceInRightTransition::new);
        TotalV.showingProperty().addListener((obs, oldValue, newValue) -> {
            if (newValue) {
                AppBar appBar = MobileApplication.getInstance().getAppBar();
                appBar.setTitleText("Calculo Total");
                appBar.getActionItems().add(MaterialDesignIcon.ARROW_BACK.button(e ->
                        MobileApplication.getInstance().switchToPreviousView()));
            }
        });
        total=0.0;
        vistaProd();
        vistaProdReg();
```

Figura 701 Código inicial de la clase totalPresenter

A continuación, se explican las funciones vistaProd y vistaProdReg, las cuales se muestran en la Figura 82:

- vistaProd(): Recupera de la lista del usuario los productos seleccionados, para ello hace uso de RestClient, colocando los parámetros necesarios para la correcta petición al servidor tales como host, la dirección del servidor (http://localhost:80/ServerCanbasC/webresources), path la dirección del servicio a invocar y su función (“BusquedaCanasta/mostrarProductosLista”), method en este caso “GET” y como queryParam el id del usuario, posteriormente hace uso de GluonObservableList<Producto> y de DataProvider.retrieveList() colocándole como parámetro el tipo de la lista de objetos a obtener, una vez hecho esto se asigna al ListView listProd y se hace uso de clase ProductosListCell para permitir mostrar en cada elemento de la lista solo el nombre del producto.
- VistaProdReg(): Recupera la lista de productos registrados en la tienda seleccionada por el usuario en la vista tiendas cercanas, para ello hace uso de RestClient de manera similar que en la función VistaProd(), con la diferencia de que esta vez el path es (“BusquedaCanasta/obtenProductosReg”) y hace uso de dos queryParam, el primero con el id del usuario y el segundo con el id de la tienda, por su parte el código que recupera esta lista cambia indicando esta vez como tipo de objetos de lista ProductoRegistrado, posteriormente la lista obtenida se asigna al ListView listpR y se hace

uso de la clase prodSelListCell para mostrar en cada elemento del ListView solo el valor del precio de cada producto, cabe señalar que es el servidor el que se encarga de obtener los precios de los productos tomando en cuenta la tienda seleccionada.

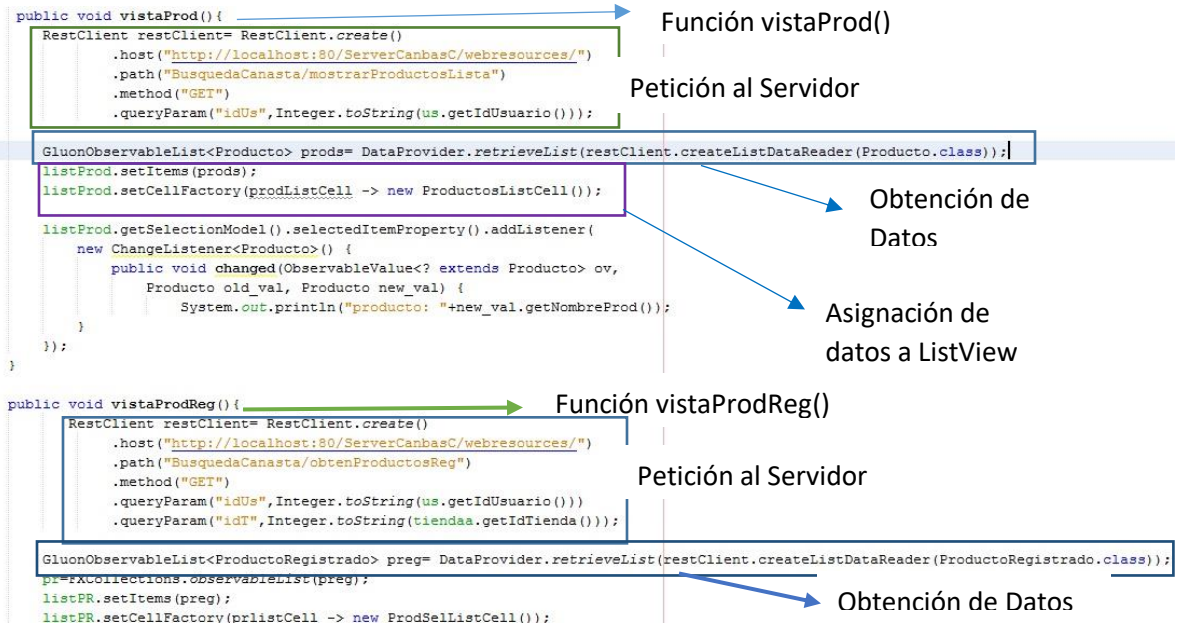


Figura 712 Función vistaProd() y vistaProdReg() de la clase totalPresenter

Funcionalidad del botón Total, ver Figura 83: Muestra el precio total de la lista de productos seleccionados por el usuario colocando en el Label el total obtenido del objeto Lista, para llevar a cabo esta acción hace el llamado a la función mostrarPrecios, la cual uso del objeto RestClient, de manera análoga a la función vistaProd(), indicando esta vez como path ("BusquedaCanasta/obtenTotal") y dos queryParam el primero indicando el id del usuario y el segundo indicando el id de la tienda seleccionada en la vista anterior, posteriormente hace uso de GluonObservableObject<Lista> y de DataProvider.retrieveObject(), colocándole como parámetro el objeto a recibir de tipo Lista, para obtener una instancia de este, una vez recuperado se le coloca un escuchador para corroborar que el estado de la petición fue exitoso y obtener el valor del total.

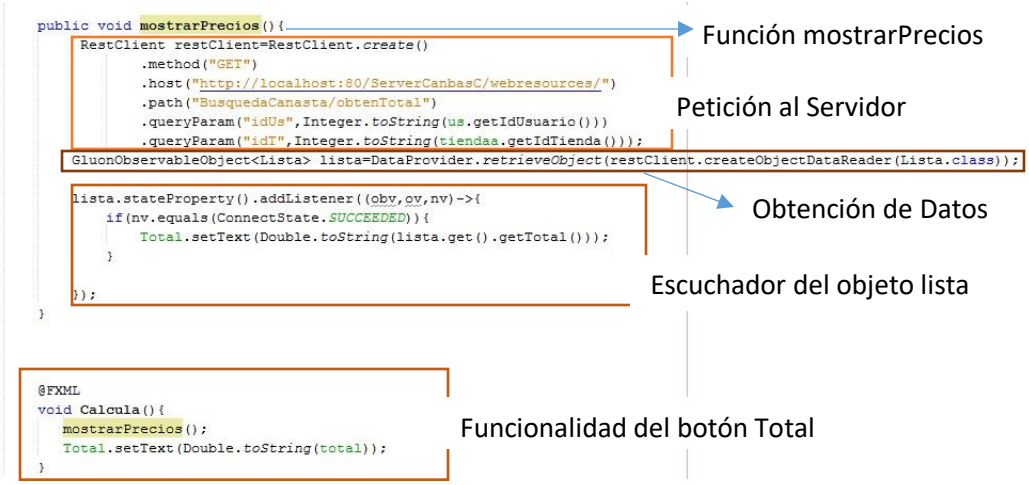


Figura 723 Función mostrarPrecios() y funcionalidad del botón Total en la clase totalPresenter

3. RESULTADOS

En el presente proyecto se desarrolló la aplicación Canbas-Cheap la cuál fue probada en una computadora de escritorio con sistema operativo Windows 8, cabe señalar que las pruebas iniciales se hicieron en un dispositivo móvil con sistema operativo Android 4.4.2 (Kitkat), pero dado que no se lograron mostrar las vistas que hacen uso de Google Maps, se manejó como versión final la versión web. Los resultados obtenidos se muestran a continuación:

Se desarrolló e implementó un servidor web con las funciones necesarias para el correcto intercambio de información con la aplicación cliente.

Para la persistencia de la información se diseñó e implementó una base de datos que contiene todas las tablas necesarias para que el servidor pueda realizar las consultas solicitadas por la aplicación cliente.

El resultado del módulo **Registrar/Login** fue satisfactorio ya que se logró la comunicación entre cliente y servidor para el registro de usuarios y así mismo el Login a través de los formularios que se muestran en las Figuras 84 y 85.

Figura 734 Pantalla de Login

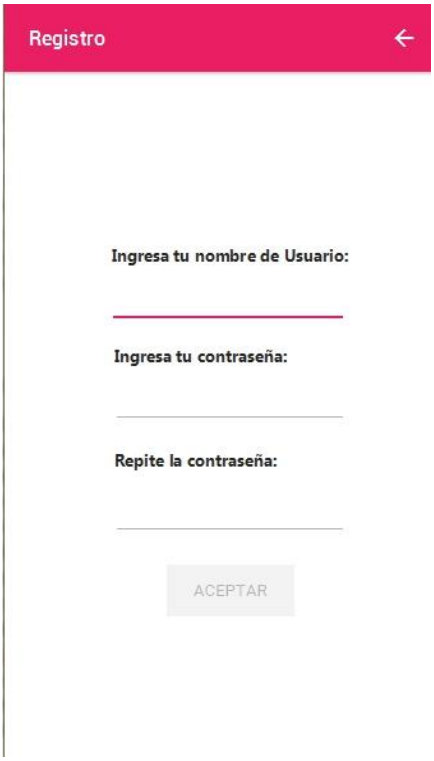


Figura 85 Pantalla de Registro

Una vez que el usuario ha ingresado correctamente sus datos podrá acceder al menú principal de la aplicación que se muestra en la Figura 86.

Ya en el menú principal el usuario ve como primera opción **Ver Mi Perfil**, el resultado de este módulo se muestra en la figura 87. Este módulo permite al usuario consultar sus datos y elegir la ubicación en que se encuentra actualmente o en la que le gustaría realizar la búsqueda de tiendas.

Figura 86 Pantalla de Menú Principal

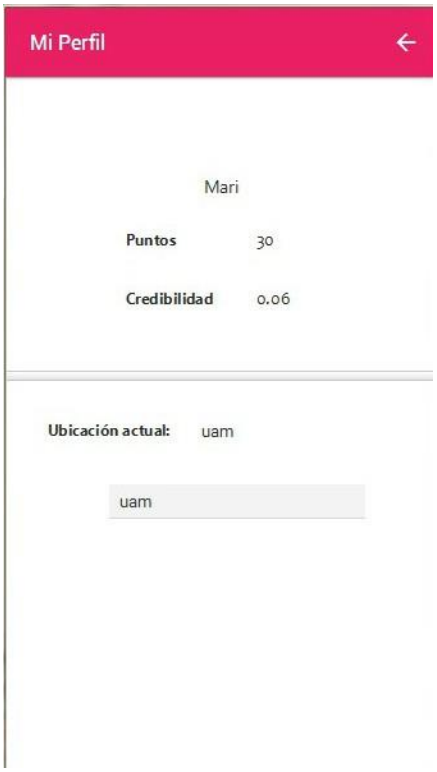


Figura 87 Pantalla de Perfil

Agregar Ubicación, el resultado de este módulo se muestra en la Figura 88, en este módulo el usuario puede registrar varias ubicaciones introduciendo el nombre de estás y su correspondiente dirección, la geocodificación de estas direcciones se hizo de manera exitosa haciendo uso del API GMapsFX, a su vez estas ubicaciones son utilizadas para el correcto registro de tiendas.

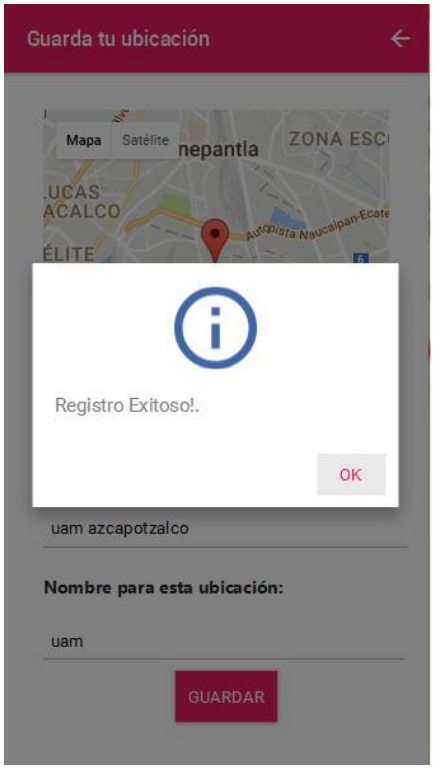


Figura 88. Pantalla Agregar Ubicación

Para explicar el resultado de las pruebas al módulo de **Registrar Producto** se hace uso de sub-módulos tales como el registro de producto, el resultado de este se muestra en la Figura 89, en el cual el usuario selecciona la categoría y la subcategoría a la que corresponde dicho producto e introduce su nombre para posteriormente ser enviados estos datos al servidor mediante una solicitud POST, hecho esto se muestra al usuario una vista (opciones tienda), en la que debe seleccionar la tienda en que se encuentra el producto a registrar, si es que tiene registrada la tienda en la ubicación en que se encuentra actualmente, e introducir el precio del producto y presionar Registrar para enviar estos datos al servidor, en caso de que no tenga registrada está tienda debe presionar registrar tienda; para el correcto funcionamiento de esta vista se hace uso de los sub-módulos recuperar tienda por ubicación y registrar producto en tienda, ver Figura 90. Si el usuario no tiene tiendas registradas se hace uso del sub-módulo registrar tienda en el cual el usuario debe introducir el nombre de la tienda que desea registrar y su dirección para que posteriormente sea convertida en coordenadas haciendo uso del API de GMapsFX para llevar a cabo esta función y enviar los datos de la tienda al servidor mediante una solicitud POST, el resultado de este sub-módulo se muestra en la Figura 91.

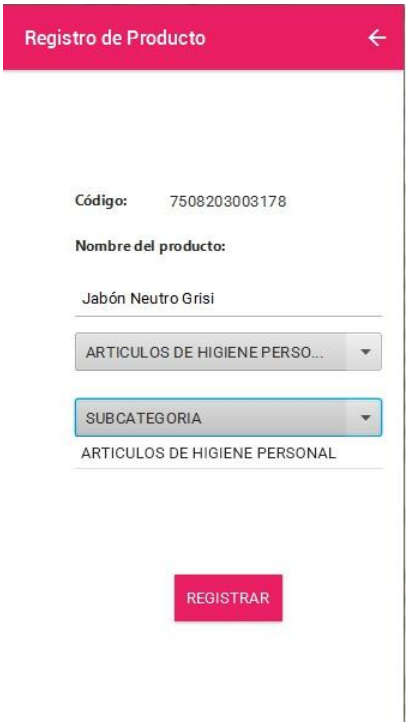


Figura 89. Pantalla de Registro de Producto

Figura 90. Pantalla de Opciones Tienda

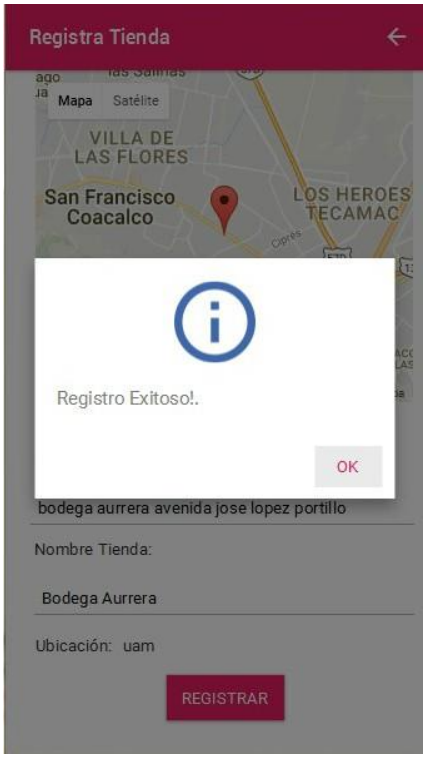


Figura 91. Pantalla de Registro de Tienda

Se obtienen resultados satisfactorios en cada uno de estos sub-módulos puesto que cada uno cumple bien con su tarea específica haciendo uso de las funciones implementadas en el servicio web, por lo que el resultado del módulo como tal es satisfactorio.

Para explicar el resultado de las pruebas al módulo **Búsqueda Canasta** se hace uso de sub-módulos tales como obtener producto por subcategoría, el resultado de este se muestra en la Figura 92 (vista productos disponibles), en la que se hace la petición al servidor para crear una nueva lista para el usuario, en esta vista se muestran al usuario las subcategorías en las que él puede buscar el producto que le interesa agregar a su canasta, para ello debe seleccionar la subcategoría y elegir de los productos mostrados aquél de su interés, si no se tienen aún productos registrados en esa subcategoría no se mostrará nada en el espacio destinado a mostrar la lista de productos, si el usuario selecciona algún producto se hace uso del sub-módulo agregar productos lista, en esta vista el usuario debe introducir la cantidad del producto en cuestión que desea añadir a su lista y presionar Aceptar, el resultado de este sub-módulo se muestra en la Figura 93, estos datos son enviados al servidor y se muestra al usuario nuevamente la vista de productos disponibles, cuando el usuario haya terminado de añadir los productos de su interés presiona el botón Aceptar, la aplicación invoca al sub-módulo tiendas cercanas mostrado en la Figura 94, en esta vista se muestra al usuario su ubicación actual dándole la posibilidad de cambiarla seleccionando el botón Cambiar, se muestra también un slider en el que el usuario puede seleccionar el radio de distancia y presiona Buscar para obtener las tiendas que se encontraron en ese rango de distancia y que además cuentan con todos los productos seleccionados por el usuario.

Figura 92. Pantalla de Productos Disponibles

Figura 93. Pantalla de Cantidad de Producto

Figura 94 Tiendas Cercanas

El usuario selecciona una tienda, y hecho esto se muestra la vista Como Llegar, ver Figura 95, en la que se hace uso de Google Directions API proporcionada por GMapsFX para trazar la ruta que el usuario debe seguir para llegar a la tienda seleccionada desde su ubicación actual, para ello el usuario debe presionar Como Llegar, esta vista también cuenta con las opciones de Consultar Total y Volver al Menú.

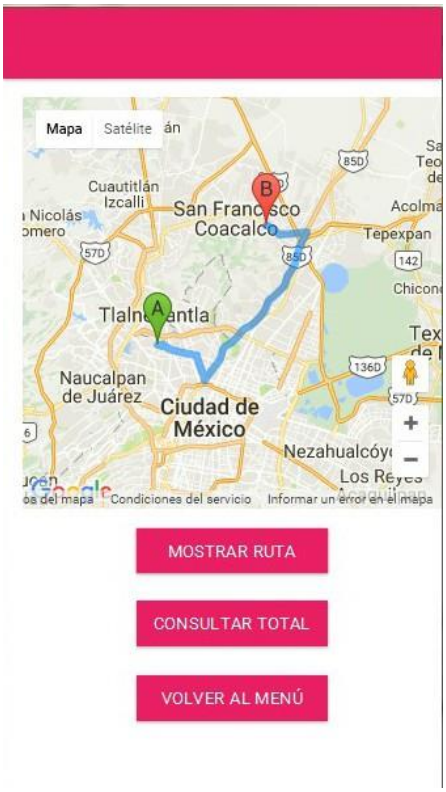


Figura 95. Pantalla de Como Llegar

Si el usuario desea conocer el total por la lista de productos que seleccionó presiona Consultar Total y se muestra la vista de la Figura 96, en dónde se muestran los productos seleccionados en una lista y su correspondiente precio, con un botón de Total que se encarga de obtener el total enviando una petición get al servidor y mostrarlo.

Figura 96. Pantalla de Total

Como podemos ver en las Figuras la aplicación funciona de la manera esperada, esto gracias a que las funciones implementadas en el servidor realizan bien su trabajo y a su vez la aplicación cliente obtiene los objetos esperados en cada petición al servidor en formato JSON.

Se realizaron los registros de veinte productos comprometidos con la propuesta, a su vez que más de cinco usuarios hicieron uso de la aplicación, obteniendo opiniones satisfactorias, tales como que el diseño de las vistas es bueno por lo que cumple con los criterios de usabilidad propuestos, además de mencionar una buena funcionalidad por parte de la aplicación, la Figura 97 muestra los registros de los productos en las distintas tiendas.

id_producto_reg	usuarioo_id	codigo_producto	tienda_id	precio	rango
1	1	7508203003178	1	20	MEDIO
2	1	7508203003178	2	15	MEDIO
3	1	653569833956	1	350	MEDIO
4	2	7501142811118	3	50	MEDIO
5	2	7501142811118	4	45	MEDIO
6	2	7501791644273	3	18	MEDIO
7	1	7501003337825	5	30	MEDIO
8	1	7501003337825	1	30	MEDIO
9	1	7506205807596	5	50	MEDIO
10	2	7501112000078	3	50	MEDIO
11	2	086141002608	3	50	MEDIO
12	2	7501791638623	3	25	MEDIO
13	2	7501008497555	3	120	MEDIO
14	2	7501008497555	4	125	MEDIO
15	2	7501025423872	3	16	MEDIO
16	2	7501025423872	4	18	MEDIO
17	2	500435014076	3	35	MEDIO
18	1	500435014076	5	40	MEDIO
19	1	500435014076	1	45	MEDIO
20	2	7501052472423	3	35	MEDIO

Figura 97.-Productos Registrados en distintas tiendas

4. CONCLUSIONES

Para el correcto funcionamiento de Canbas-Cheap se implementó una arquitectura Cliente-Servidor.

El desarrollo inicial del Servidor constituyó un gran reto, puesto que se pretendía hacer uso de Hibernate-JPA con un conocimiento básico de esta herramienta, por lo que aprender a hacer el mapeo de la base de datos a clases entidad y posteriormente hacer las consultas de manera correcta consumió algo de tiempo, ya que en las primeras pruebas se obtenían errores en cuanto a la declaración de sentencias de SQL nativo.

Por otra parte, se logró hacer el despliegue del servidor utilizando Apache Tomcat como publicador, por lo que la aplicación cliente podía tener acceso al servidor y ejecutar correctamente sus tareas, para probar la funcionalidad del servidor y de los servicios que este proporciona, se hizo uso de POSTMAN, logrando con ello encontrar algunos errores en cuanto a las funciones que debía llevar a cabo el servidor y corregirlas de manera

satisfactoria, por lo que se puede concluir que el desempeño del servidor fue satisfactorio, puesto que todas las funciones que se implementaron en este llevaban correctamente a cabo sus tareas, se logró el correcto acceso a la base de datos a través de Hibernate con JPA-2.1 obteniendo así un conocimiento más a fondo de esta herramienta tan útil en estos días.

Para el desarrollo de la aplicación cliente se tuvieron más limitaciones, tomando en cuenta que se propuso PhoneGap para su desarrollo, se comenzaron a hacer algunas vistas lo que consumió tiempo, posteriormente se optó por cambiar este software a Gluon Mobile puesto que es un software más robusto y además permite el desarrollo de aplicaciones de escritorio, para dispositivos Android e iOs escribiendo un sólo código, cumpliendo los requisitos de la presente aplicación. Para el uso de Gluon Mobile fue necesario aprender las nociones básicas de JavaFX, ya que Gluon trabaja con este lenguaje y con Java para el desarrollo de sus aplicaciones.

Gluon Mobile trabaja con la arquitectura Modelo Vista Presentador, por lo que se requirió hacer varias pruebas para aprender su funcionamiento y en qué difiere del Modelo Vista Controlador, obteniendo con ello más conocimiento de esta arquitectura.

Para el desarrollo y diseño de las vistas de la aplicación se consideraron criterios de usabilidad, tratando de hacer lo más predictivo posible el funcionamiento de cada componente, se hizo uso de SceneBuilder como herramienta de diseño ya que permite agregar elementos a la vista sin necesidad de codificarlos, además de permitir añadirle a los elementos un identificador y alguna acción a realizar. Estos elementos son invocados dentro de la clase Presentador de cada una de las vistas proporcionándoles con ello una funcionalidad específica.

Gluon Mobile proporciona muchas herramientas para el desarrollo de aplicaciones móviles, por lo que se requirió aprender el funcionamiento de estas, tales como Gluon Charm Down, esta herramienta en su versión más reciente 4.3, puede inducir en tiempo de ejecución en qué plataforma se está probando la aplicación y con base en ello ejecutar diferentes funciones de visibilidad de los componentes, por lo que fue utilizada a lo largo de todo el desarrollo de la aplicación cliente, por otra parte Gluon Connect fue utilizada para el acceso a los servicios Rest.

La dificultad quizá más significativa fue el uso de Google Maps, ya que se tuvo que hacer una búsqueda de distintas API´s que pudieran ser funcionales para la aplicación cliente, se optó por GMapsFx puesto que hace uso de Java y no de JavaScript como el API original, además de proporcionar los servicios requeridos por la aplicación tales como Geocoding y Directions, las vistas que hacen uso de estos servicios funcionan de manera correcta en

las pruebas de escritorio, pero al momento de probarlas en un dispositivo móvil (Android 4.4.2 KitKat) estas mostraban un error, se consumió tiempo buscando una solución a esto y dado que aún es un API en desarrollo aún no cuenta con la funcionalidad en dispositivos móviles, por lo que se tomó la decisión de dejar la aplicación en su versión web.

Aunque parcialmente se logró el objetivo de hacer una aplicación funcional en móvil, considero que el funcionamiento de las aplicaciones cliente y servidor son satisfactorios puesto que ambos cumplen con los objetivos especificados, además de que se logró un gran aprendizaje en el desarrollo de este tipo de aplicaciones que consideran criterios de usabilidad para el diseño de las interfaces de usuario, consumen servicios Rest, hacen uso de nuevas API's como GMapsFX., son funcionales en distintas plataformas, esto en cuanto a la aplicación cliente y en cuanto a la aplicación del servidor se logró un conocimiento más avanzando de Hibernate y de servicios Res*t.*

Para futuros trabajos podría considerarse buscar otra herramienta para mostrar las vistas que hacen uso de Google Maps o talvez incluir código nativo dependiendo de la plataforma y usar como tal el API de Google Maps y con ello lograr la funcionalidad de la aplicación en dispositivos móviles.

5.- BIBLIOGRAFÍA

[1] Elinpc.com.mx, "Canasta básica mexicana 2016 | elinpc.com.mx", 2016. [En línea]. Disponible: http://elinpc.com.mx/canasta -basica-mexicana/. [Visitado: 29-Jan-2016].

[2] Productos en las canastas básicas de distintas instituciones públicas, 1st ed. México, D.F.: PROFECO, 2016. http://www.profeco.gob.mx/transparencia/transfocaliza/Nota_Productos_canastas_basicas.pdf.

[3] GUÍA CÓDIGO DE PRODUCTO, 1ed. México D.F.: GS1 México, 2012, pp.4-6.

[4] K. Guzmán Villanueva, "Aplicación móvil para la recomendación de productos en el comercio electrónico", proyecto de integración, División de Ciencias Básicas e Ingeniería, Universidad Autónoma Metropolitana Azcapotzalco, México, 2015.

[5] "Profeco", *Profeco.gob.mx*, 2016. [En línea]. Disponible: http://www.profeco.gob.mx/precios/canasta/default.aspx [Visitado: 10-Feb-2016].

[6] Radarprice.com, “Radarprice – Tu app comparador de precios IOS/Android”,2016. [En línea]. Disponible: http://www.radarprice.com/es/. [Visitado:11-Feb-2016].

[7] idealo.es, “Comparador de Precios y Ofertas Online”, 2016. [En línea]. Disponible: http://www.idealo.es/. [Visitado: 12-Feb-2016]

[8] M. Álvarez Durán, “Análisis, diseño e implementación de un sistema de control de ingreso de vehículos basado en visión artificial y reconocimiento de placas en el parqueadero de la Universidad Politécnica Salesiana-Sede Cuenca”, Licenciatura, Universidad Politécnica Salesiana-Sede Cuenca, 2014.

[9] I. Hidalgo Bejarano y R. Sánchez García de Blas, “RECONOCIMIENTO DE CARACTERES MEDIANTE IMÁGENES EN CONTADORES DE GAS EN ENTORNOS REALES”, Licenciatura, Universidad Complutense de Madrid, 2015.

[10] ÍNDICE NACIONAL DE PRECIOS AL CONSUMIDOR ENERO DE 2016, 1st ed. AGUASCALIENTES, AGS.: INEGI, 2016.pdf.

[11] "Google Maps - Glosario Mercadotecnia", *Glosario Mercadotecnia*, 2016. [En línea]. Disponible: https://www.headways.com.mx/glosario-mercadotecnia/palabra/google-maps/. [Visitado: 16- Nov- 2016].

[12] "Guía del desarrollador | Google Maps Geocoding API | Google Developers", *Google Developers*, 2016. [En línea]. Disponible: https://developers.google.com/maps/documentation/geocoding/intro?hl=es-419. [Visitado: 20- Nov- 2016].

[13] "Guía del desarrollador | Google Maps Directions API | Google Developers", *Google Developers*, 2016. [En línea]. Disponible: https://developers.google.com/maps/documentation/directions/intro?hl=es-419. [Visitado: 15- Dec- 2016].

[14] "Fórmula del haversine", *Es.wikipedia.org*, 2016. [En línea]. Disponible: https://es.wikipedia.org/wiki/F%C3%B3rmula_del_haversine. [Visitado: 10- Jun- 2016].

[15] "What Are RESTful Web Services? - The Java EE 6 Tutorial", *Docs.oracle.com*, 2016. [En línea]. Disponible: http://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html. [Visitado: 18- Jul- 2016].

[16] J. Rodriguez and J. Rodriguez, "¿Qué es JPA (Java Persistence API)?", *Oraclejuniors.blogspot.mx*, 2016. [En línea]. Disponible: http://oraclejuniors.blogspot.mx/2014/11/que-es-jpa-java-persistence-api.html. [Visitado: 19- Feb- 2016].

[17]A. Project, "Apache Tomcat® - Welcome!", *Tomcat.apache.org*, 2016. [En línea]. Disponible: https://tomcat.apache.org/index.html. [Visitado: 17- Jul- 2016].

[18]"NetBeans IDE 8.2 Release Information", *Netbeans.org*. [En línea]. Disponible: https://netbeans.org/community/releases/82/. [Visitado: 12- Feb- 2016].

[19]F. 4.3.0, "Gluon Mobile Documentation", *Docs.gluonhq.com*, 2016. [En línea]. Disponible: http://docs.gluonhq.com/charm/4.3.0/. [Visitado: 02- Jan- 2017].

[20]A. PhoneGap, "PhoneGap", *Phonegap.com*, 2016. [En línea]. Disponible: http://phonegap.com/. [Visitado: 27- Feb- 2016].

[21]"GMapsFX", *Rterp.github.io*. [En línea]. Disponible: http://rterp.github.io/GMapsFX/. [Visitado: 18- Dec- 2016].

[22]"Scene Builder - Gluon", *Gluon*. [En línea]. Disponible: http://gluonhq.com/products/scene-builder/. [Visitado: 21- Aug- 2016].

[23]A. Kinga, "Afterburner.fx The Opinionated JavaFX Productivity Igniter", *Afterburner.adam-bien.com*. [En línea]. Disponible: http://afterburner.adam-bien.com/. [Visitado: 20- Nov- 2016].

[24]F. 1.2.0, "Gluon Connect Documentation", *Docs.gluonhq.com*, 2016. [En línea]. Disponible: http://docs.gluonhq.com/connect/1.2.0/. [Visitado: 22- Jan- 2017].

[25]"Gluon Mobile 4.3.0 API", *Docs.gluonhq.com*. [En línea]. Disponible: http://docs.gluonhq.com/charm/javadoc/4.3.0/index.html?com/gluonhq/charm/down/plugins/PositionService.html. [Visitado: 15- Dec- 2016].

[26]"Gluon Mobile 4.3.0 API", *Docs.gluonhq.com*, 2016. [En línea]. Disponible: http://docs.gluonhq.com/charm/javadoc/4.3.0/index.html?com/gluonhq/charm/down/plugins/BarcodeScanService.html. [Visitado: 16- Dec- 2016].

6. ENTREGABLES

6.1 Entregable A.

Código Fuente de la aplicación, ubicado en el cd del presente proyecto en la ubicación:

CodigoFuente/CanbasCheap.zip

6.2 Entregable B.

Manual de usuario, se muestra un vídeo explicando el uso de la aplicación, ubicado en el cd del presente proyecto en la ubicación:

ManualDeUsuario/CanbasCheapUso.zip