UNIVERSIDAD AUTONOMA METROPOLITANA
UNIDAD AZCAPOTZALCO

División de Ciencias Básicas e Ingeniería
Licenciatura en Ingeniería en Computación

Proyecto Tecnológico

Plataforma Digital para la asignación óptima de transportes turísticos a clientes utilizando geolocalización

Proyecto de Integración de Ingeniería en Computación

Karla Angélica Nieto García

Asesora
Dra. Silvia Beatriz González Brambila
Profesora Titular
Departamento de Sistemas

Coasesor
M. en C. Josué Figueroa González
Profesor Asociado
Departamento de Sistemas

17-Primavera

31 de agosto de 2017

DECLARATORIA

Yo, **SILVIA BEATRIZ GONZÁLEZ BRAMBILA**, declaro que aprobé el contenido del presente Reporte de Proyecto Integración y doy mi autorización para su publicación en la Biblioteca Digital, así como el Repositorio Institucional de UAM Azcapotzalco.

González Brambila

Silvia Beatriz González Brambila

Yo, **JOSUÉ FIGUEROA GONZÁLEZ**, declaro que aprobé el contenido del presente Reporte de Proyecto Integración y doy mi autorización para su publicación en la Biblioteca Digital, así como el Repositorio Institucional de UAM Azcapotzalco.

Josué F.G.

Josué Figueroa González

Yo, **KARLA ANGÉLICA NIETO GARCÍA**, 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 el Repositorio Institucional de UAM Azcapotzalco.

Karla Angélica Nieto García

RESUMEN

En los últimos años, el avance de la tecnología en aplicativos Web ha permitido a los usuarios poder acceder con mayor facilidad a la localización de sus sitios de interés, la disponibilidad de la tecnología en dispositivos como celulares, computadoras y tablets abre un mundo de oportunidades a aplicaciones que pueden ser usadas de manera cotidiana, facilitando el arribo a sitios específicos proporcionando múltiples rutas para llegar a ellos, además de un tiempo estimado de traslado.

Sumando herramientas como el API de Google Maps y Geolocalización se tiene la posibilidad de desarrollar una herramienta capaz de analizar, seleccionar, asignar, monitorear y almacenar cierta información que ayuda a mejorar el control de todo aquel sitio que tenga una ubicación física.

Con la finalidad de proporcionar una herramienta informática que permita conocer la posición geográfica (latitud y longitud) de una o más sucursales de transporte turístico, en el presente trabajo se describe el diseño e implementación de un Plataforma Web que permite asignar un vehículo de transporte turístico mediante la generación de un servicio que recibe como parámetros de entrada la posición del usuario y requerimientos específicos del vehículo para posteriormente evaluar la distancia entre ese punto y las sucursales registradas en el sistema.

El sistema es capaz de asignar un vehículo registrado y ligado a una sucursal de trasporte turísticos evaluando la posición del usuario, la posición de las sucursales y validando que los vehículos cumplan con las características especificadas.

Como resultado se obtiene una Plataforma Web desarrollada es una herramienta eficaz para la asignación de transportes turísticos, el despliegue de las sucursales dentro de un mapa de Google Maps y el monitoreo de los servicios asignados.

TABLA DE CONTENIDO

INTRODUCCIÓN **7**

ANTECEDENTES **8**

JUSTIFICACIÓN **10**

OBJETIVOS **11**

Objetivo General 11
Objetivos Específicos 11

MARCO TEÓRICO **12**

Plataforma de Desarrollo. 12
Arquitectura del Sistema. 12
Aplicación Web 13
API de Google Maps 13

DESARROLLO DEL PROYECTO **14**

DISEÑO DE LAS BASES DE DATOS 14
Mongo DB 15
DISEÑO DE LA FUNCIONALIDAD 16
DISEÑO DE LAS INTERFACES 20

DISEÑO DE LA ARQUITECTURA **36**

INTERACCION DE TODOS LOS COMPONENTES DE LA PLATAFORMA 36

CONCLUSIONES **37**

BIBLIOGRAFÍA **38**

APÉNDICE A (CÓDIGO FUENTE) **39**

CONFIGURACIÓN DE VARIABLE DE ENTORNO FILE_CONFIG_WS 39
ARCHIVO DE CONFIGURACIÓN 41
CLASES PRINCIPALES 42
COLECCIONES DE MONGO DB 54
REPOSITORIOS DE MONGO DB 59

ÍNDICE DE FIGURAS

FIGURA 1. ARQUITECTURA DEL SISTEMA 12
FIGURA 2. DIAGRAMA DE BASE DE DATOS SQL 14
FIGURA 3. COLECCIÓN DE SERVICIOS 15
FIGURA 4. COLECCIÓN DE SPRINGSESSION 15
FIGURA 5. DIAGRAMA DE SECUENCIA ADMINISTRADOR PRINCIPAL 16
FIGURA 6. DIAGRAMA DE SECUENCIA ADMINISTRADOR DE SUCURSAL 17
FIGURA 7. DIAGRAMA DE SECUENCIA AGENTE DE SUCURSAL 18
FIGURA 8. DIAGRAMA DE SECUENCIA AGENTE GENERAL 19
FIGURA 9. INTERFAZ DE LOGIN 20
FIGURA 10. EJEMPLO DE LLENADO DE DATOS PARA ACCESO AL SISTEMA 20
FIGURA 11. ACCIÓN DE CARGA PARA NAVEGACIÓN ENTRE PÁGINAS 21
FIGURA 12. INTERFAZ DE CONSULTA DE SUCURSALES 21
FIGURA 13. INTERFAZ PARA REGISTRO DE SUCURSALES DE TRANSPORTE 22
FIGURA 14. DESPLIEGUE DE SUCURSALES REGISTRADAS EN MAPA DE GOOGLE MAPS 22
FIGURA 15. INTERFAZ DE CONSULTA DE CARACTERÍSTICAS ESPECIALES 23
FIGURA 16. INTERFAZ DE REGISTRO DE CARACTERÍSTICAS ESPECIALES 23
FIGURA 17. INTERFAZ DE CONSULTA DE USUARIOS 24
FIGURA 18. INTERFAZ DE REGISTRO DE USUARIOS PARA ADMINISTRADOR PRINCIPAL 25
FIGURA 19. LISTA DE ROLES PARA REGISTRO DE USUARIOS DEL ADMINISTRADOR PRINCIPAL 25
FIGURA 20. INTERFAZ DE REGISTRO DE SERVICIO 26
FIGURA 21. INTERFAZ DE MONITOREO DE SERVICIOS ACTIVOS 27
FIGURA 22. LLENADO DE INTERFAZ LOGIN PARA ACCEDER COMO USUARIO ADMINISTRADOR DE SUCURSAL 27
FIGURA 23. INTERFAZ DE MONITOREO PARA ADMINISTRADOR DE SUCURSAL 28
FIGURA 24. INTERFAZ DE CONSULTA PARA VEHÍCULOS REGISTRADOS DE UNA SUCURSAL 28
FIGURA 25. INTERFAZ DE ALTA DE VEHÍCULOS PARA ADMINISTRADOR DE SUCURSAL 29
FIGURA 26. INTERFAZ DE CONSULTA DE OPERADORES 29
FIGURA 27. INTERFAZ PARA ALTA DE OPERADORES 30
FIGURA 28. INTERFAZ DE CONSULTA DE USUARIOS REGISTRADOS POR ADMINISTRADOR DE SUCURSAL 30
FIGURA 29. INTERFAZ DE REGISTRO DE USUARIOS PARA EL ADMINISTRADOR DE SUCURSAL 31
FIGURA 30. LISTA DE ROLES PARA ADMINISTRADOR DE SUCURSAL 31
FIGURA 31. INTERFAZ DE MONITOREO DE SERVICIOS PARA AGENTE DE SUCURSAL 32
FIGURA 32: INTERFAZ QUE DESPLIEGA MENSAJE DE ACCIÓN AL AGENTE DE SUCURSAL 32

FIGURA 33. INTERFAZ QUE MUESTRA ÍCONO DE TÉRMINO DE SERVICIO 33
FIGURA 34. INTERFAZ QUE DESPLIEGA MENSAJE DE TERMINO DE SERVICIO..... 33
FIGURA 35. INTERFAZ PARA MOSTRAR EL HISTORIAL DE SERVICIOS 34
FIGURA 36. INTERFAZ DE REGISTRO DE SERVICIO PARA AGENTE GENERAL..... 34
FIGURA 37. INTERFAZ DE MONITOREO DE SERVICIOS PARA AGENTES GENERALES..... 35
FIGURA 38. INTERFAZ QUE DESPLIEGA EL MENSAJE PARA AGENTES GENERALES..... 35
FIGURA 39. INTERACCIÓN DE LOS COMPONENTES Y MÓDULOS DE LA PLATAFORMA DIGITAL 36
FIGURA 40. PROPIEDADES DEL EQUIPO Y VENTANA EMERGENTE DE CONFIGURACIONES AVANZADAS 39
FIGURA 41. VARIABLE DE ENTORNO PARA ACCESO A ARCHIVOS DE CONFIGURACIÓN..... 40

INTRODUCCIÓN

En la actualidad la geolocalización ha revolucionado algunos servicios, principalmente los de ocio, turísticos y de transporte, ya que hace más eficiente la búsqueda al conocer la posición del usuario que la realiza. El servicio de Google Maps presenta una posición y puede proponer una ruta para llegar más rápido a esa posición, proporcionando adicionalmente un tiempo estimado de arribo dependiendo de cómo se desee realizar el traslado, ya se caminando, en automóvil propio, transporte público o usando el servicio de Uber.

En el área turística, las empresas que ofrecen servicios de transporte a los turistas, tienen problemas ya que éstos suelen requerir unidades con ciertas características y muchas veces la comunicación entre sucursales de una empresa no es lo bastante eficiente para enviar una unidad adecuada al cliente, lo que genera molestia y pérdidas de tiempo y de recursos para la empresa, así como la insatisfacción de los clientes y malas críticas respecto al respecto al servicio.

Actualmente no existe ninguna aplicación que ofrezca el servicio de asignación por geolocalización de transporte turístico (autos y camionetas que satisfagan requerimientos de espacio para uno o más usuarios).

En el presente documento se explica el desarrollo de una Plataforma Digital que se encarga de la asignación de vehículos de transporte a sus clientes, tomando en cuenta las características especificadas por ellos, mejorando la logística y administración de los vehículos, proporcionando atención eficaz y reduciendo insatisfacción de los clientes.

ANTECEDENTES

Sistema de información Web para la geolocalización de Unidades Aéreas [1].

En este sistema se implementó un sistema para visualizar en tiempo real los puntos en coordenadas de aeronaves que sobrevolaban el Espacio Aéreo Mexicano.

Dicho sistema implementó:

- La visualización en tiempo real y con márgenes de error mínimo de los puntos en coordenadas de aeronaves que sobrevuelan.
- El procesamiento en alta velocidad de despliegue dentro del Sistema Web de datos en información cartográfica.

Es diferente al presente desarrollo en el tipo de unidades a monitorear, ya que aquí se propone la gestión de unidades ubicadas en sucursales de transporte turístico.

Búsqueda de rutas más cortas entre diferentes sistemas de transporte en la Ciudad de México aplicando el enfoque del Modelo GT [2].

El sistema implementó una aplicación para hallar la ruta más corta entre un punto y una zona geográfica, usando solo medios de transporte públicos.

Las funcionalidades implementadas en este proyecto fueron:

- Solicitud de Origen-Destino
- Generación de Grafos GT
- Implementación del algoritmo
- Despliegue de la ruta encontrada

La diferencia principal con el presente desarrollo es que aquí se trata de unidades de transporte que deben cumplir con ciertas características y no de sistemas de transporte masivo con rutas ya establecidas.

Sistema de monitoreo para vehículos con una ruta predeterminada mediante GPS [3].

Implementó un sistema para la monitorización en una PC de la ubicación de vehículos, obteniendo su ubicación mediante un receptor GPS y transmitiéndola por radiofrecuencia a una estación base en tiempo real para la visualización en mapas.

El principal objetivo de este sistema fue lograr tener un control sobre las rutas y tiempos de desplazamiento de vehículos públicos y privados con una ruta

predeterminada. Se adaptaron e implementaron tecnologías de comunicación de alto nivel como el GPS y transmisiones de RF (aplicaciones comerciales).

Se distingue de este desarrollo, ya que no se utilizará radiofrecuencia para la transmisión de posiciones, se usará el API de GoogleMaps para la ubicación de sucursales y posición del usuario.

Uber [4]

Es una herramienta en la cual un usuario pide un vehículo, la aplicación detecta su localización y a partir de este dato encuentra a un conductor para que lo vaya a recoger. Esta aplicación permite visualizar la posición del conductor en un mapa e incluso poder contactarlo, mediante un mensaje o una llamada. El pago se hace mediante la aplicación, al llegar al destino se carga automáticamente el importe en la cuenta personal, asociada a una tarjeta de débito o crédito.

La principal diferencia este desarrollo es sobre monitorización ya que se propone tener la ubicación de las sucursales y la información de las unidades que tiene cada una para poder proporcionar el servicio, no se enfoca tener la ubicación de manera individual de los vehículos.

Easy Taxi [5]

Es una herramienta que obtendrá automáticamente la dirección del usuario usando GPS, permitiendo la edición o confirmación detectada. Adicionalmente cuenta con la opción para seleccionar el tipo de pago y la personalización de la carrera (ruta para realizar el servicio) y finalmente busca un taxi para atender la petición.

Se distingue de este desarrollo, ya que la ruta y el vehículo para atender un servicio son calculados tomando la posición de la sucursal (indicada en sistema) que tiene la unidad con las características solicitadas, hasta la posición que el usuario proporciona y no con vehículos que estén en movimiento o circulación fuera de ella.

Waze [6]

Es una herramienta que permite mejorar la conducción diaria, proporcionando al usuario rutas para llegar de un punto origen a un punto destino, en las cuales los ayudará a evitar atascos, accidentes o cualquier contratiempo, gracias a la ayuda en tiempo real de otros conductores.

Se relaciona en cuanto al uso de geolocalización, pero no considera la gestión de vehículos y seguimiento a los servicios que están ofreciendo.

JUSTIFICACIÓN

Las agencias de servicios de transporte turístico siempre buscan ofrecer un servicio de calidad a sus clientes, respondiendo de manera rápida a sus peticiones de transporte y que éstos cubran las necesidades requeridas por los mismos. El que un transporte llegue tarde por sus clientes o que se les envíe uno que no satisfaga sus necesidades, puede ocasionar que se considere no usar nuevamente los servicios de esa agencia. Estos problemas suelen presentarse por una mala logística, ya que, al tenerse varias sucursales, no se sabe qué unidades tiene cada una, con qué características cuentan y en dónde están o cómo es el estatus del servicio que están prestando. Lo que genera confusiones y pérdidas de tiempo o envío de unidades que un cliente acaba por rechazar.

El presente desarrollo pretende ayudar a mejorar los tiempos de atención a usuarios que solicitan un servicio de transporte con características específicas (asientos para niños, capacidad para un determinado número de personas, etc.). Esto permitirá mejorar los tiempos en las operaciones de empresas que ofrecen el servicio de transporte a turistas y, por lo tanto, mejorar la calidad en la atención evitando retrasos e incumplimiento a los usuarios.

La aplicación ayudará a una empresa a conocer qué transportes con determinados requerimientos solicitados, se encuentra más cerca de un cliente, reduciendo el tiempo de espera.

Además, le permitirá llevar un mejor control sobre sus unidades de transporte y seguimiento de los servicios que ofrece.

OBJETIVOS

Objetivo General

Construir una Plataforma Digital para la asignación de transportes turísticos utilizando la geolocalización y considerando las necesidades de los usuarios.

Para el cumplimiento del presente objetivo fue necesario subdividirlo en objetivos específicos.

Objetivos Específicos

- Diseñar e implementar un módulo de solicitud de servicios.
- Diseñar e implementar un módulo para visualizar la información de los servicios asignados y las unidades que los atienden.
- Diseñar e implementar un módulo para la gestión de las sucursales de transportes[1] turísticos.
- Diseñar e implementar un módulo para el despliegue de las sucursales de transportes turísticos registrados en el sistema de acuerdo con su geolocalización.
- Diseñar e implementar un módulo para gestión de información de los vehículos de transporte turístico.
- Diseñar e implementar un módulo para gestión de las características de los vehículos de transporte turístico.
- Diseñar e implementar un módulo de Asignación de Unidades para atención de servicios de transportes turísticos.

La Plataforma Digital cubre los objetivos mencionados, ya que cuenta con un módulo para el registro correspondiente de servicios, se tiene un módulo para monitorear el avance y los datos de cada servicio, se crearon módulos para registrar, editar y eliminar sucursales, vehículos, características de vehículos y un módulo para el despliegue de sucursales en un mapa, así como el motor de asignación que es el módulo de asignación de unidades.

[1] Sitio físico (se gestionará su dirección, teléfono, nombre y número de sucursal) en el que se encuentran los vehículos cuando no están en servicio o al que regresan cuando terminan de brindar alguno.

MARCO TEÓRICO

Plataforma de Desarrollo.

El sistema se desarrolló utilizando tecnología Java, a continuación se muestra la arquitectura propuesta y la descripción de los componentes que en ella intervienen.

Arquitectura del Sistema.

Arquitectónicamente la Plataforma Web (ver Figura 1) está compuesta de los siguientes componentes:

- **Conexión segura:** Manejo de usuarios y passwords para garantizar la autenticación de los administradores principales, administradores de sucursales, agentes de sucursales y agentes generales.

- **Apache Tomcat:** Servidor de aplicaciones debido a su bajo costo y soporte para el volumen tareas.

- **SQL:** Se hizo uso de la Base de Datos SQL Server para almacenar los datos relacionales de la Plataforma Web.

- **No SQL:** Se hizo uso de Mongo DB basados en los siguientes factores:
  - No se realizó la conversión entre los objetos del aplicativo y los objetos de la BD, lo cual nos permitió diseñar servicios que, con un mínimo de ajustes permitieran hacer uso de nuevos datos.
  - Hace uso de la memoria interna para almacenar la información de trabajo, lo que permitió un acceso más rápido de los datos
  - Almacenamiento tipo documento: Los datos se almacenan en forma de documentos de estilo JSON.

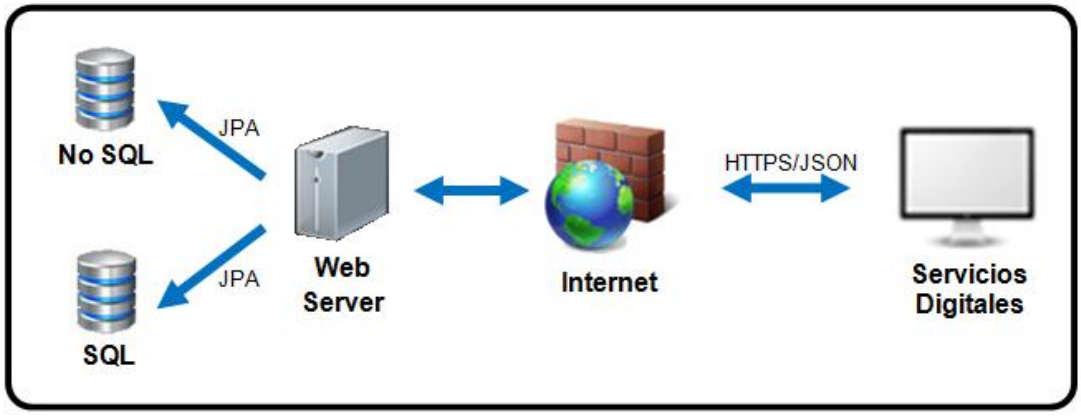

Figura 1. Arquitectura del Sistema

Aplicación Web

La aplicación se basa en una Plataforma para la asignación de vehículos establecidos en Sucursales de Transporte usando el principio de geolocalización, la funcionalidad de esta plataforma se basa en un Motor de Búsqueda y asignación basada en los principios del API de Google Maps evaluando dos puntos de ubicación (origen y destino) para el cálculo en distancia y tiempo del traslado de un punto a otro; para realizar una mejor selección y evaluación en el momento de asignación se evalúan adicionalmente características específicas con las que cuentan los vehículos y disponibilidad de los mismos.

El desarrollo del aplicativo se realizó en tres capas: Front-End usando HTML5, Back-End con servicios en Sprint BOOT (request y response en JSON) y como Middleware AngulasJS, la descripción de la arquitectura correspondiente se puede visualizar en el Apéndice B.

API de Google Maps

Servidor de aplicaciones que permite a los usuarios que permite a los usuarios incorporar un motor de rutas en Google capaz de generar rutas para recorrer en auto, en bicicleta, a pie o en transporte público, permite a los usuarios superponer sus propios datos sobre un Mapa de Google Maps personalizado, ofrece imágenes de mapas desplazables, fotografías por satélite del mundo. Las principales ventajas de esta herramienta son:

- Ofrece 2500 solicitudes diarias de manera gratuita de los Servicios Web de codificación geográfica, cómo llegar, Matriz de Distancia y de Elevación.

- Otorga 2500 solicitudes diarias del API de rutas.

DESARROLLO DEL PROYECTO

DISEÑO DE LAS BASES DE DATOS

SQL Server

Se usó esta herramienta para guardar la información principal para la plataforma como los históricos de los servicios, datos de las sucursales de transportes, vehículos y las características especiales que pueden tener y los operadores, la estructura de la Base de Datos se puede visualizar en la Figura 2:

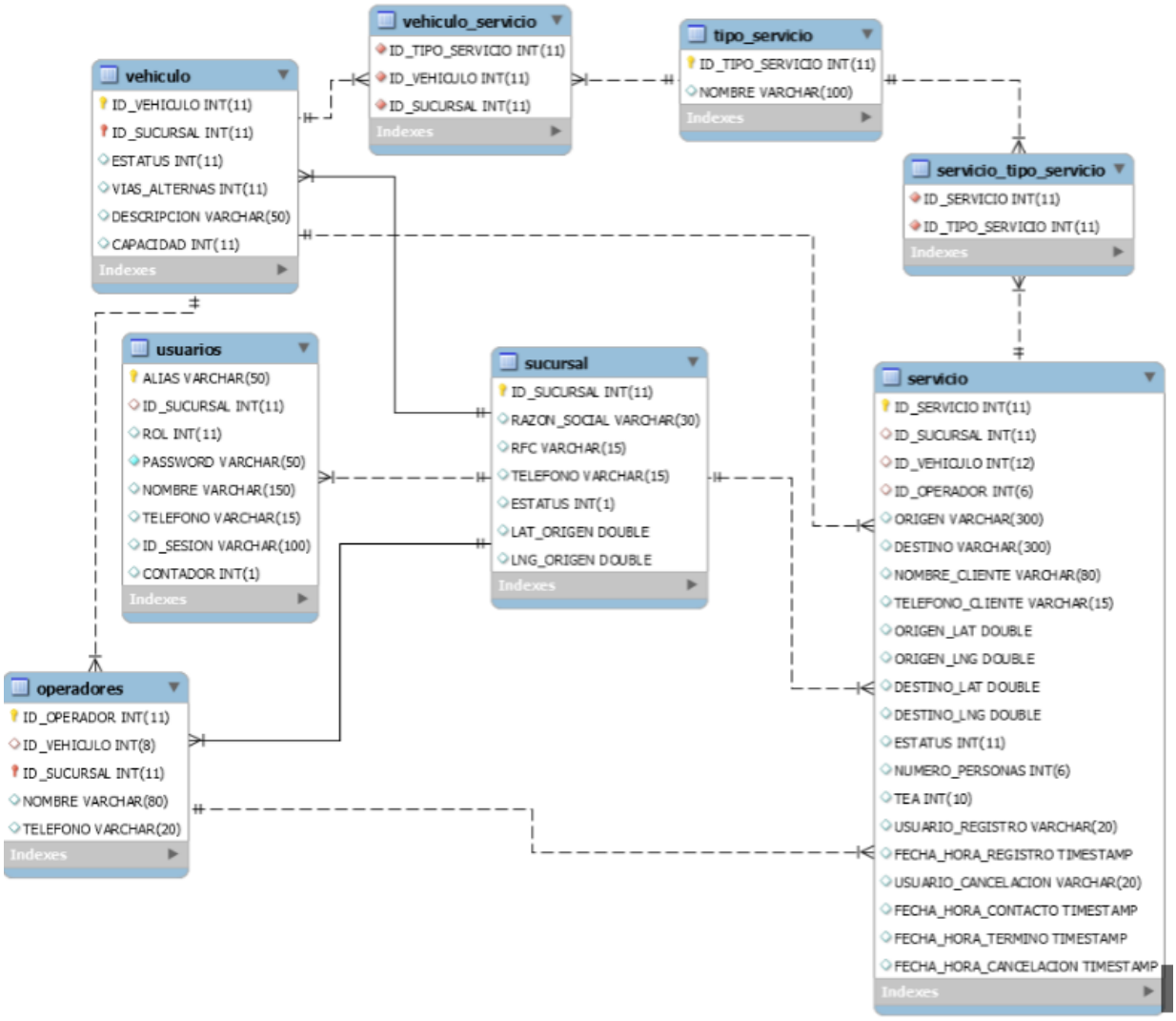

Figura 2. Diagrama de Base de Datos SQL

Mongo DB

Se utilizó este tipo de base de datos no relacional para poder acceder y guardar la información en tiempo real como el registro de los servicios asignados y que están siendo atendidos, así como las sesiones activas. La generación de las tablas correspondientes se realiza tomando los datos desde el paquete de MongoDB que tienen la anotación @Document.

Colección de Servicios

La codificación para la generación del objeto Servicios se puede visualizar en la Figura 3:

```
1  package com.ms.mongo.dto;
2
3  import java.io.Serializable;
8
9  @Document(collection="Servicios")
10 public class Servicios implements Serializable {
11
```

Figura 3. Colección de Servicios

Colección de SessionSpring

La codificación para la generación del objeto SessionSpring se puede visualizar en la Figura 4, que tiene como funcionalidad administrar las sesiones activas dentro de la plataforma:

```
1  package com.ms.mongo.dto;
2
3  import org.springframework.data.mongodb.core.mapping.Document;
5
6  @Document(collection="SessionSpring")
7  public class SessionEntity {
8
```

Figura 4. Colección de SpringSession

DISEÑO DE LA FUNCIONALIDAD

Para conocer la interacción del Administrador Principal con el sistema se diseñó el diagrama de secuencia que se muestra en la Figura 5, en el Módulo de Alta/Editar Usuarios para este rol sólo está permitido el alta de Administradores de Sucursal y Agente General:

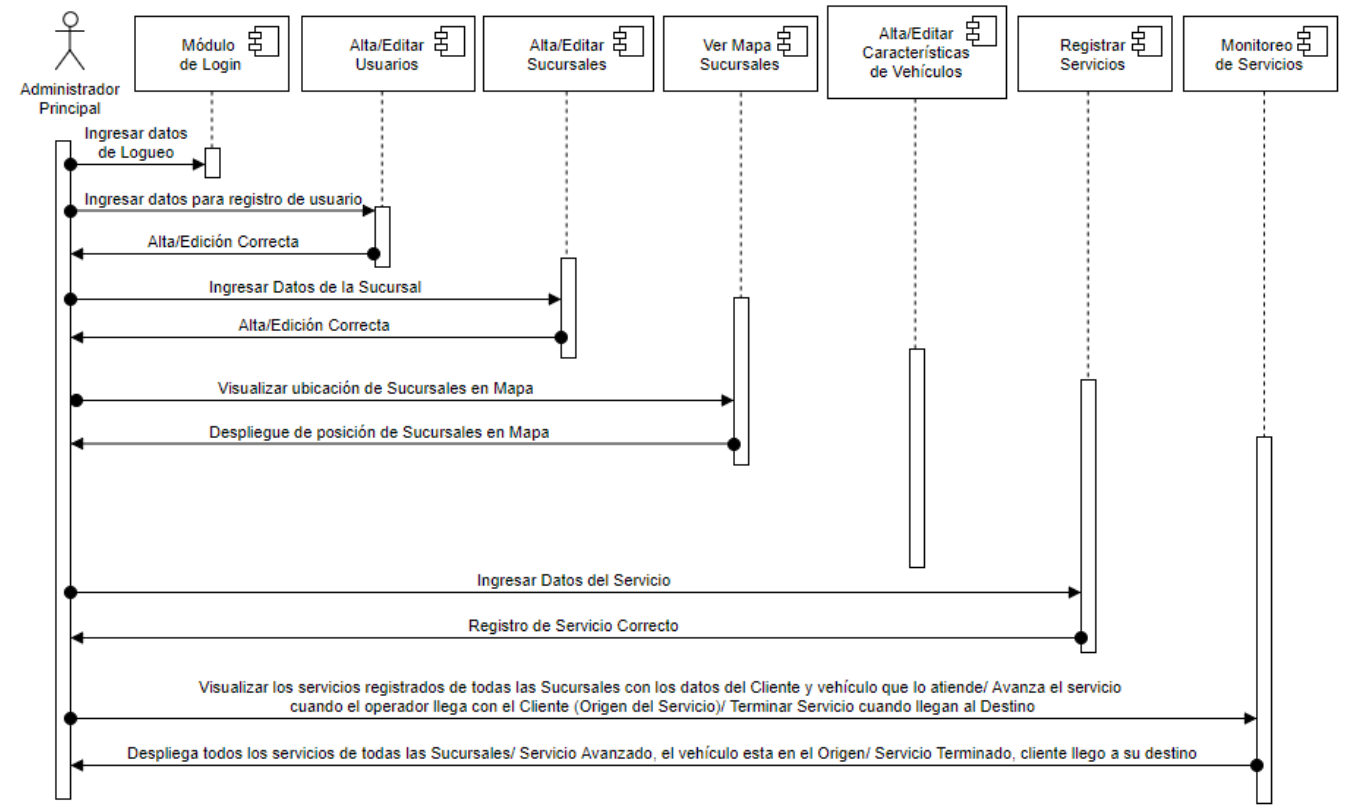

Figura 5. Diagrama de Secuencia Administrador Principal

La interacción del Administrador de Sucursal se visualiza en el diagrama de secuencia mostrado en la Figura 6, para este rol el registro de usuarios sólo incluye a los Agentes de Sucursal, también se permite registrar operadores, pero ellos no tienen acceso a la plataforma:

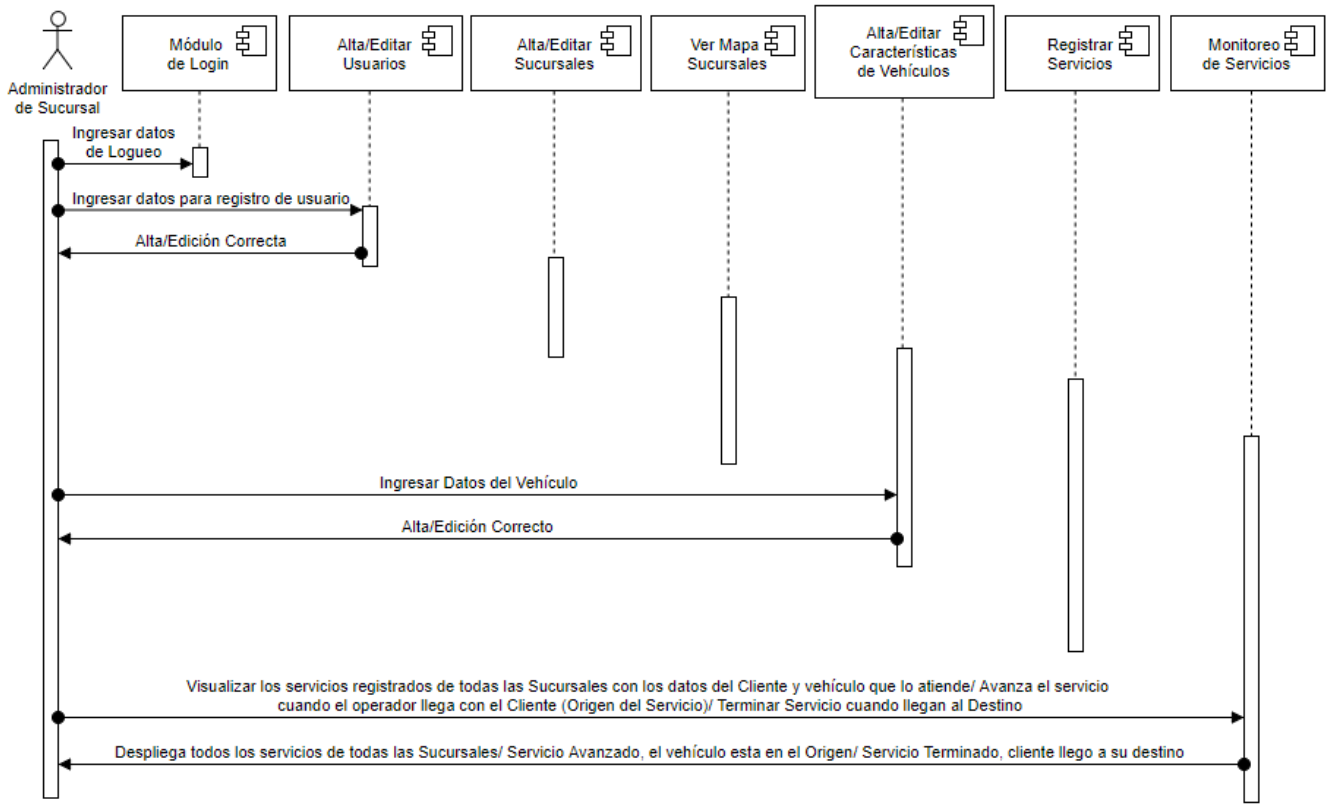

Figura 6. Diagrama de Secuencia Administrador de Sucursal

La forma en la que el Agente de Sucursal interactúa con el sistema se muestra en la Figura 7, este rol, sólo tiene el privilegio de monitoreo de los servicios de la Sucursal a la que pertenece, puede avanzar el servicio cuando el vehículo llega al origen y terminarlo cuando se llega al destino del cliente:

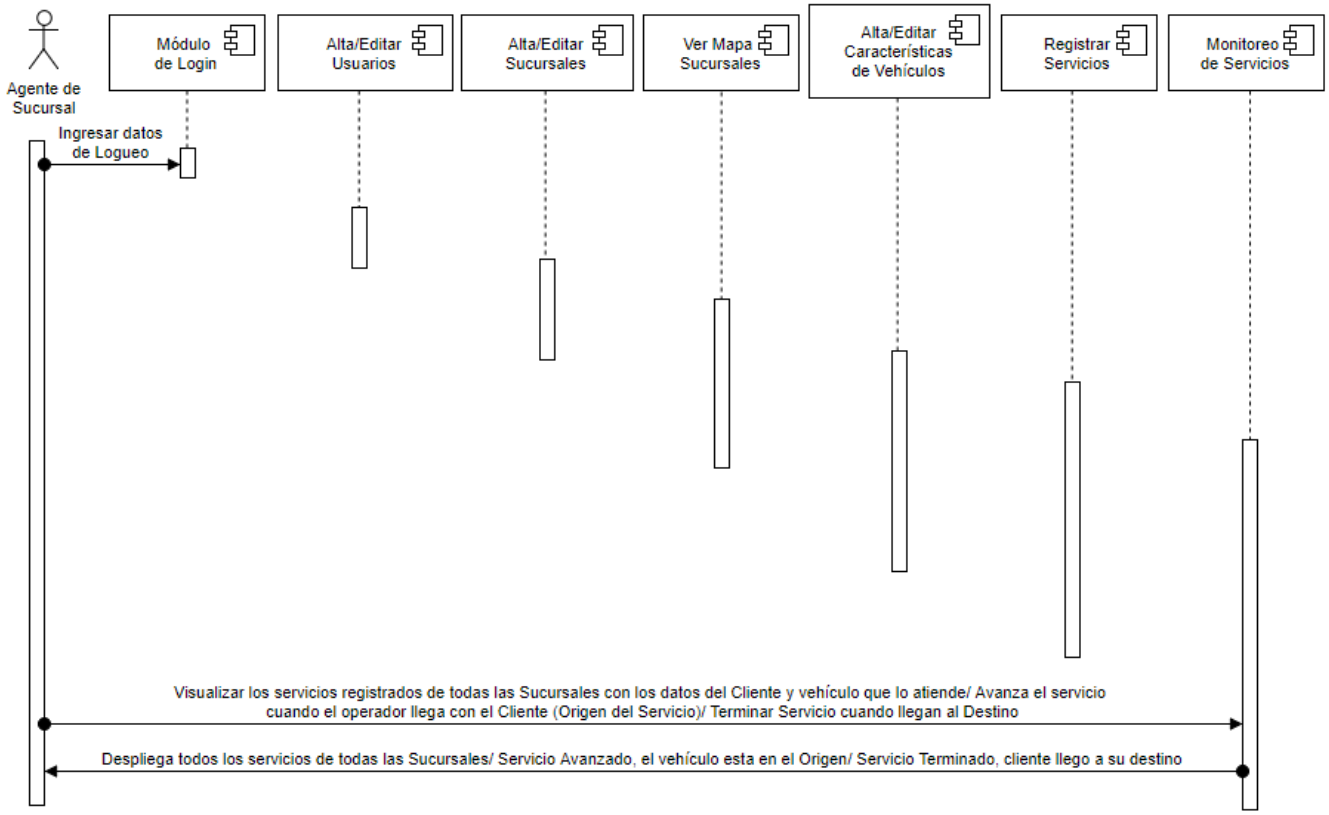

Figura 7. Diagrama de Secuencia Agente de Sucursal

La interacción del Agente General se visualiza en el diagrama de secuencia mostrado en la Figura 8, para este rol sólo se permite el registro de Servicios en el Sistema, sin embargo, puede monitorearlos, pero no tiene la facultad para avanzarlos ni terminarlos:

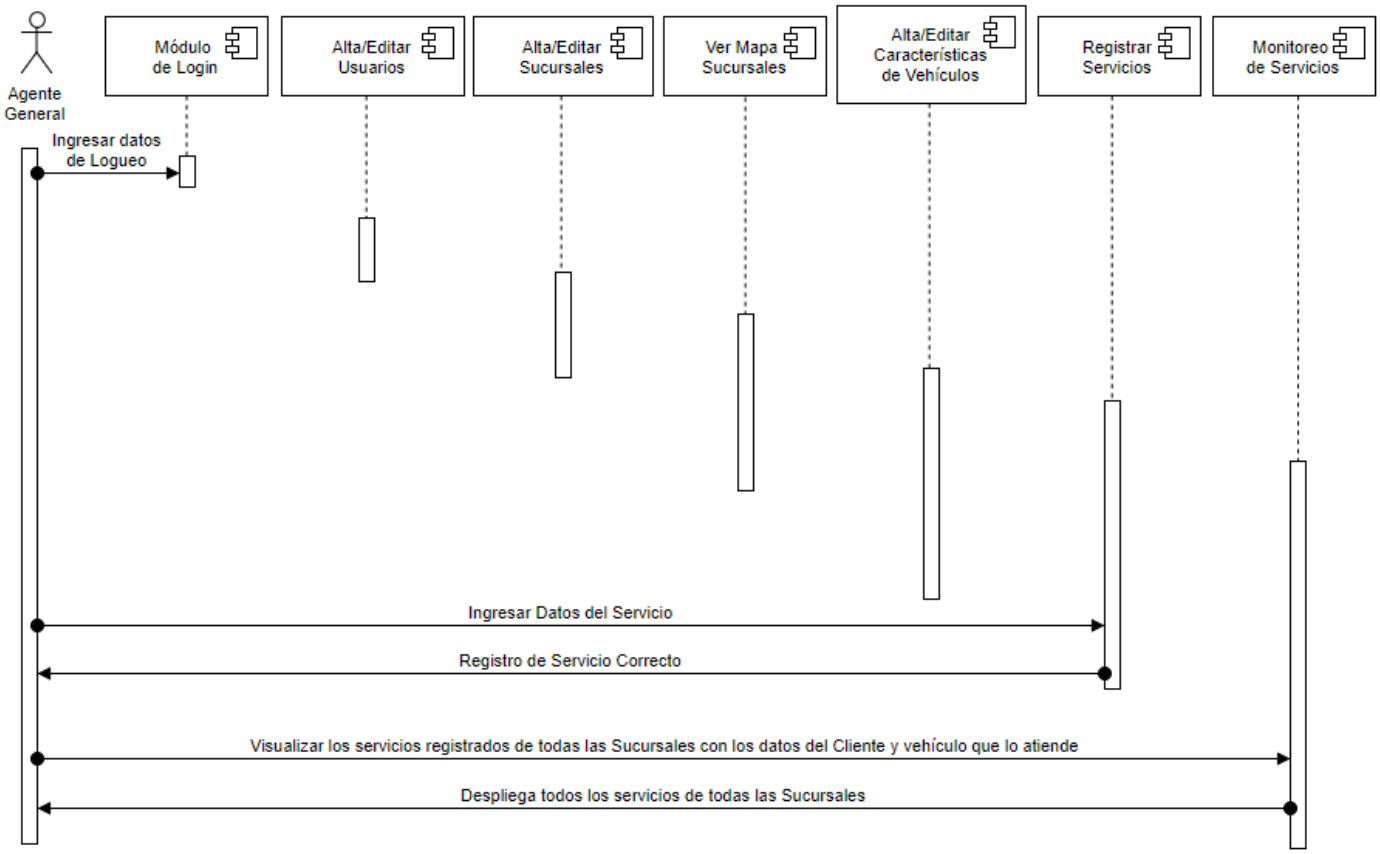

Figura 8. Diagrama de Secuencia Agente General

DISEÑO DE LAS INTERFACES

Se diseñó una página inicial, la cual permite al usuario firmarse para acceder al sistema y poder realizar las operaciones que el desee, por ejemplo el registro de sucursales, vehículos, operadores y servicios, tal como se muestra en la Figura 9:

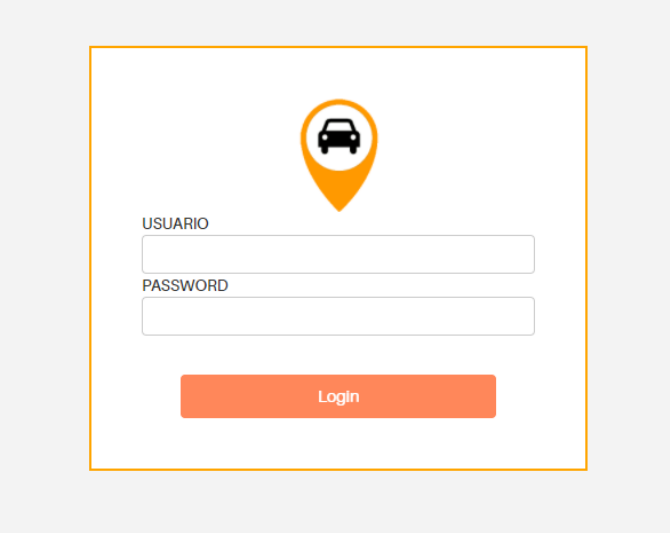

Figura 9. Interfaz de Login

Un ejemplo de colocación de datos para acceder al sistema se muestra en la Figura 10, los datos de acceso son Usuario y Password:

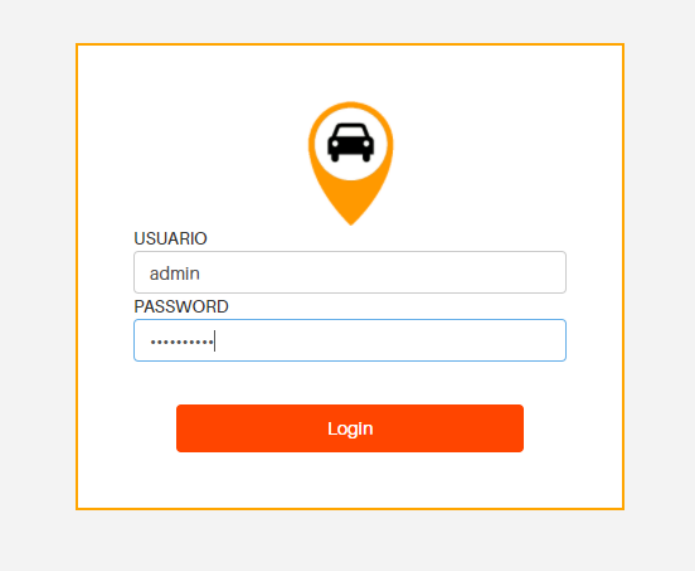

Figura 10. Ejemplo de llenado de datos para acceso al Sistema

Se diseñó una acción para mostrar la carga correspondiente al momento de navegar entre páginas, como se muestra en la Figura 11, se despliega la carga a la siguiente pantalla.

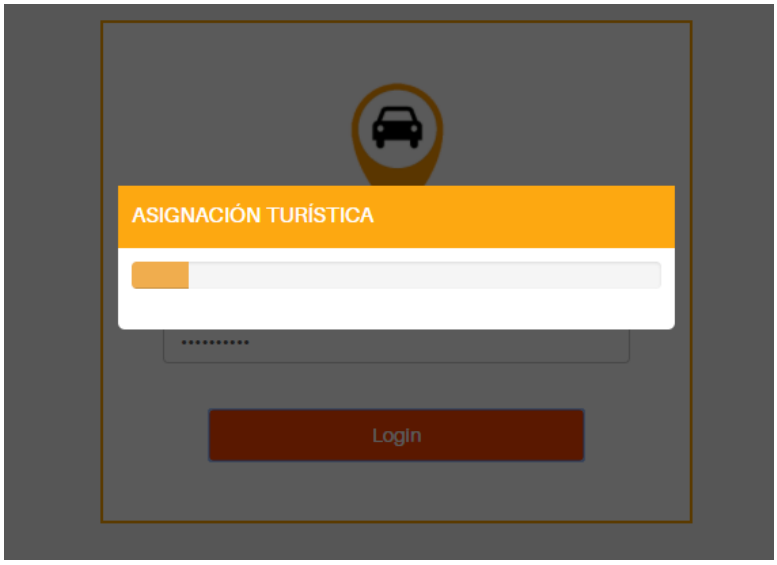

Figura 11. Acción de carga para navegación entre páginas

Se diseñó una Interfaz de Consulta de Sucursales, como se muestra en la Figura 12, se permite visualizar una lista de las que se tienen registradas en el Sistema, así como los íconos de editar y eliminar:

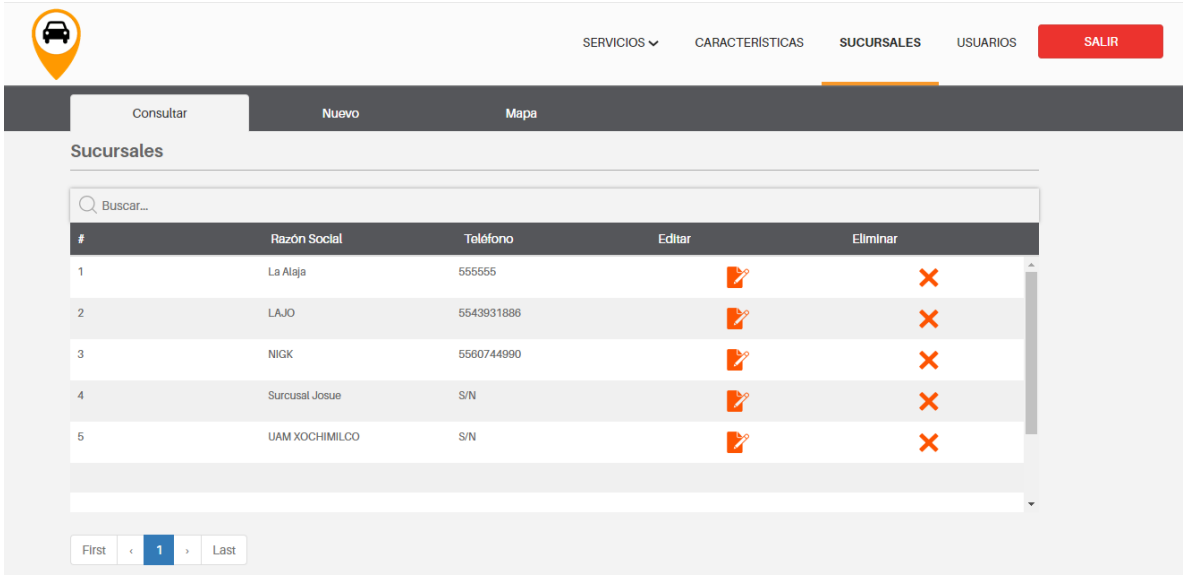

Figura 12. Interfaz de Consulta de Sucursales

Para mejorar la experiencia de navegación dentro del sistema, se diseñó una Interfaz amigable para el registro de Sucursales, en la cual se despliega un mapa, como se muestra en la Figura 13, donde adicionalmente se ingresan la Razón Social, el RFC, Teléfono y Dirección (caja de texto sobre el mapa):

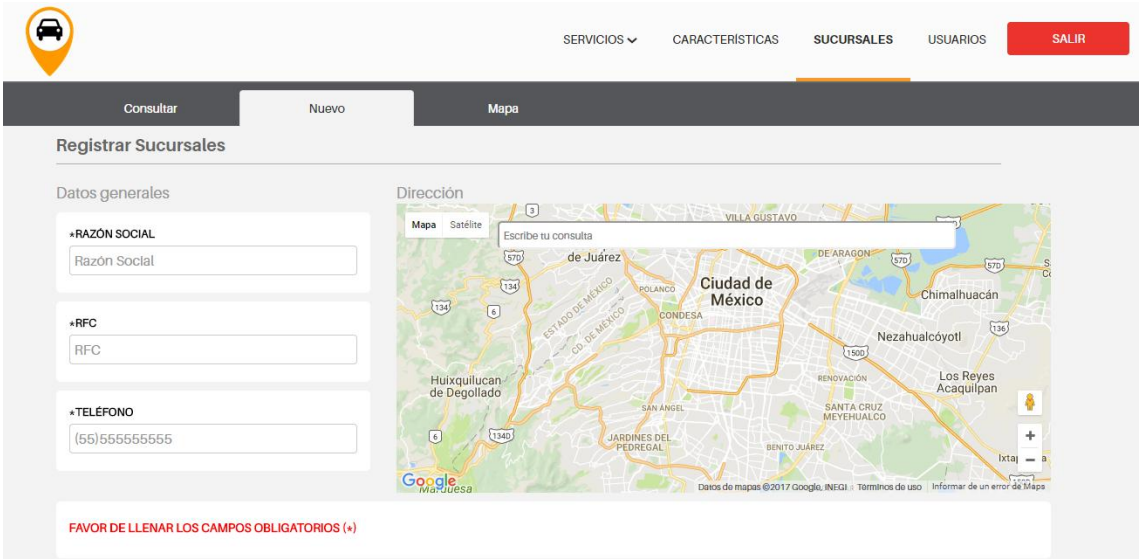

Figura 13. Interfaz para registro de Sucursales de Transporte

Para el despliegue de las Sucursales de Transporte, se diseñó una interfaz más dinámica como se muestra en la Figura 14, la cual permite desplegar las sucursales registradas sobre un mapa, en la cual se hace uso de la Geolocalización (latitud y longitud a partir de la dirección registrada en el sistema):

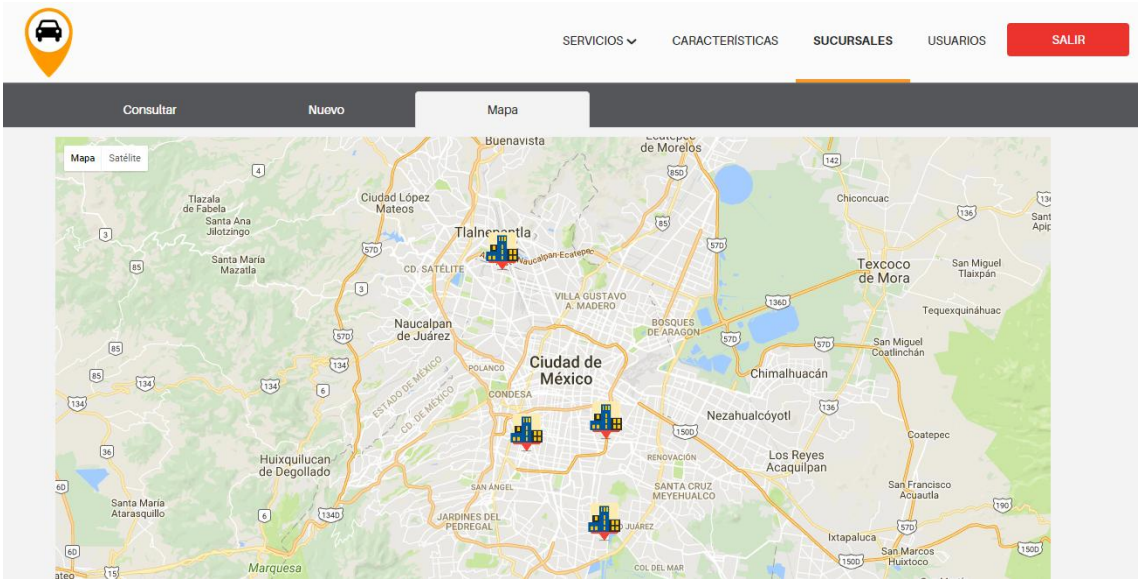

Figura 14. Despliegue de Sucursales registradas en mapa de Google Maps

Para la captura de características especiales, que podrán ser asignadas a los vehículos, se diseñó una interfaz de consulta como se muestra en la Figura 15, la cual permite el despliegue de una lista de características registradas, además de las opciones de Editar y Eliminar:

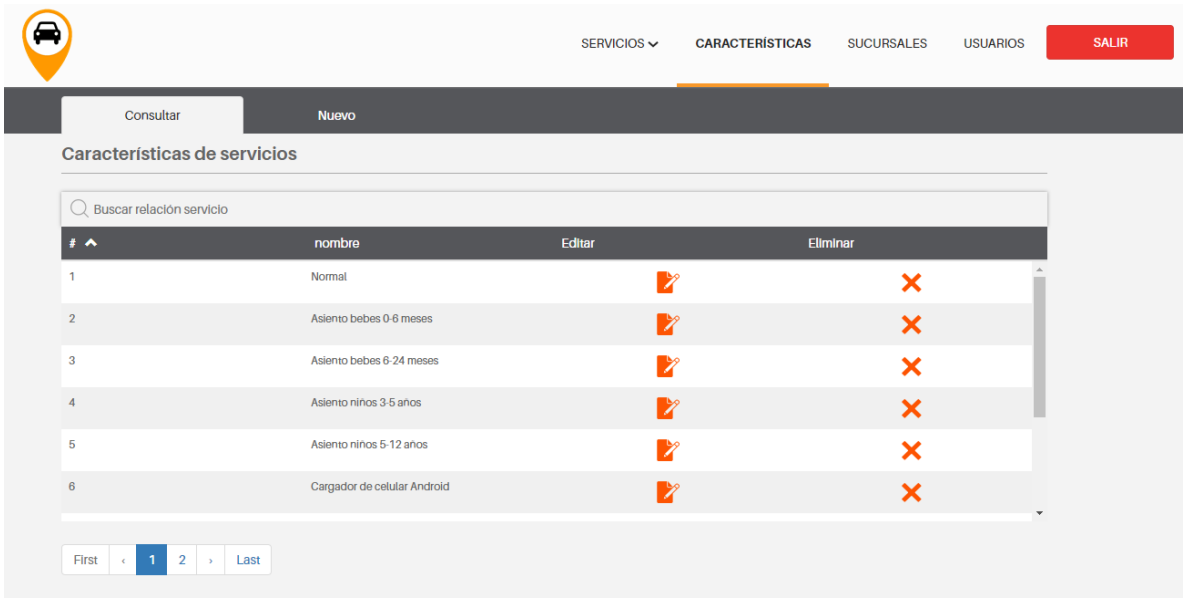

Figura 15. Interfaz de Consulta de características especiales

Se diseñó una interfaz de registro de características de vehículos, como se muestra en la Figura 16, la cual solicita el nombre que esta llevará:

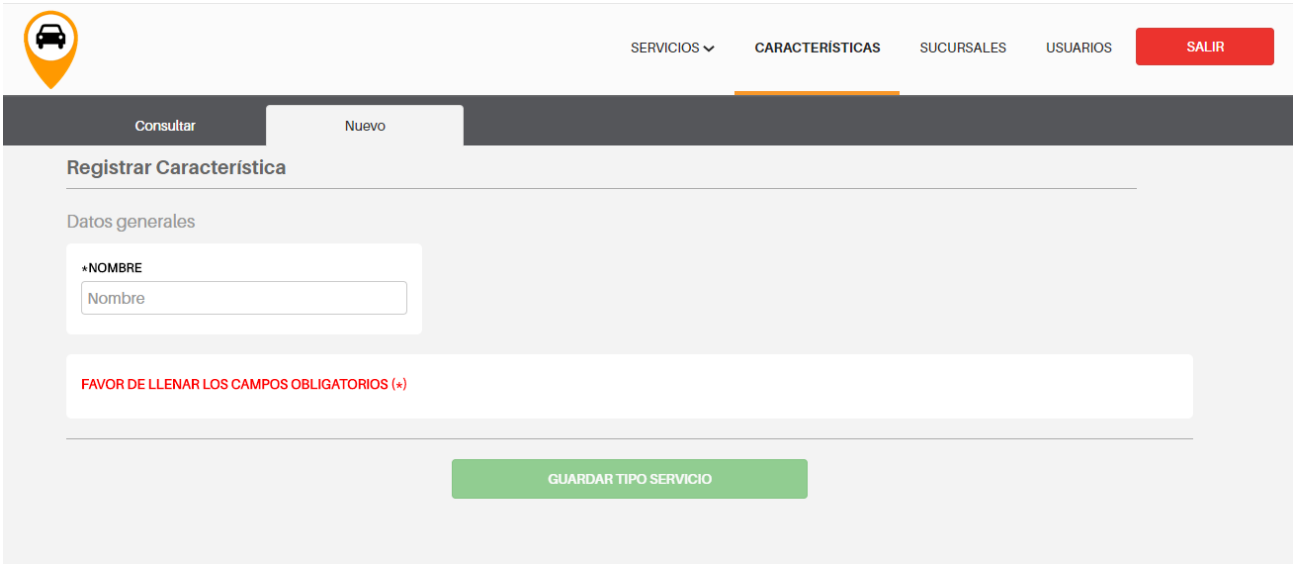

Figura 16. Interfaz de registro de características especiales

Para la consulta de usuarios registrados, se diseñó una interfaz que permita el despliegue de los usuarios registrados por el administrador principal como se muestra en la Figura 17, adicionalmente se colocaron los íconos de Editar y Eliminar:

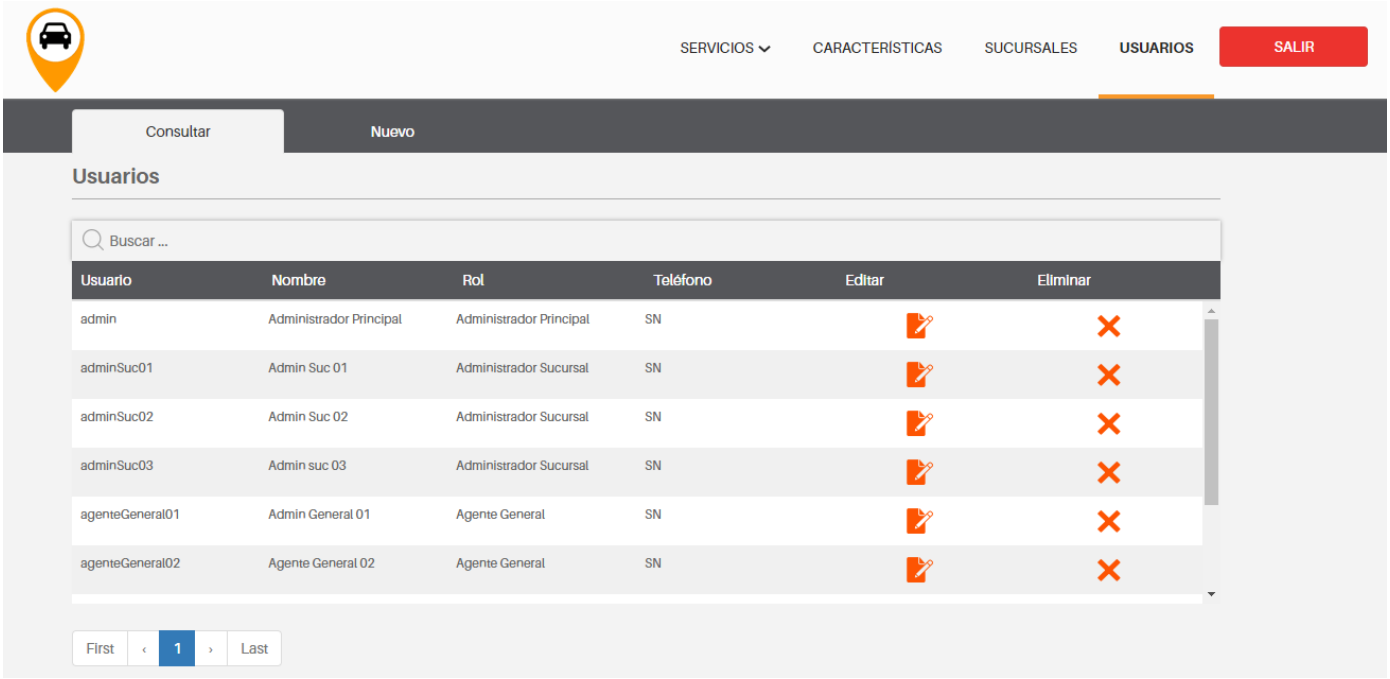

Figura 17. Interfaz de consulta de usuarios

Se diseñó una interfaz de registro de usuarios donde el administrador principal realizará esta acción, tal como se muestra en la Figura 18, en la cual de acuerdo al rol seleccionado serán los privilegios que este usuario tendrá dentro de la plataforma, los datos que deben ingresarse para poder realizar el registros correctamente son usuario, contraseña, confirmación de contraseña, nombre, teléfono y rol:

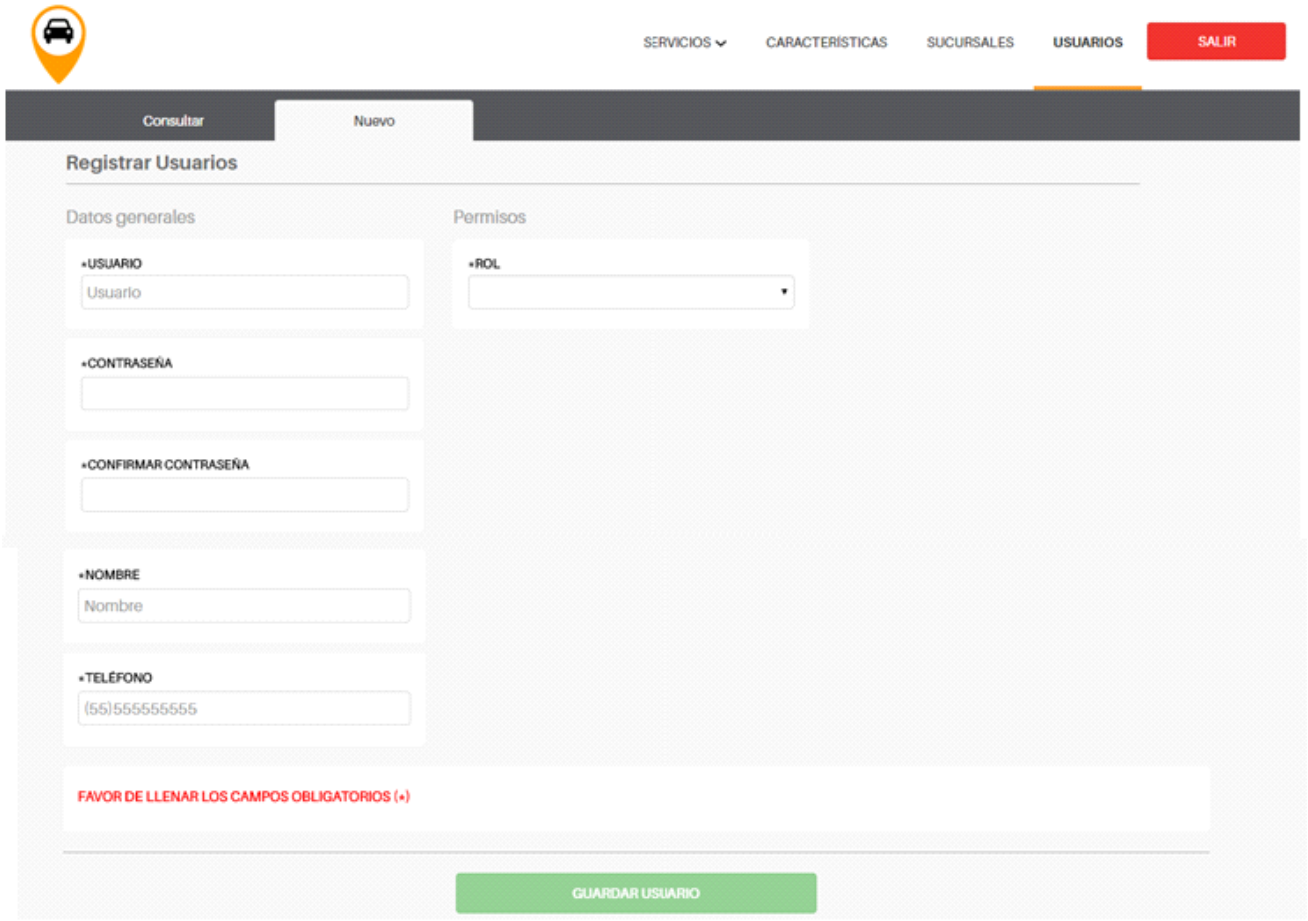

Figura 18. Interfaz de registro de usuarios para administrador principal

Para el registro que se realiza en la Figura 18, la opción de Rol contiene una lista de roles, los cuales podrán ser seleccionados por el administrador principal tal como se muestra en la Figura 19.

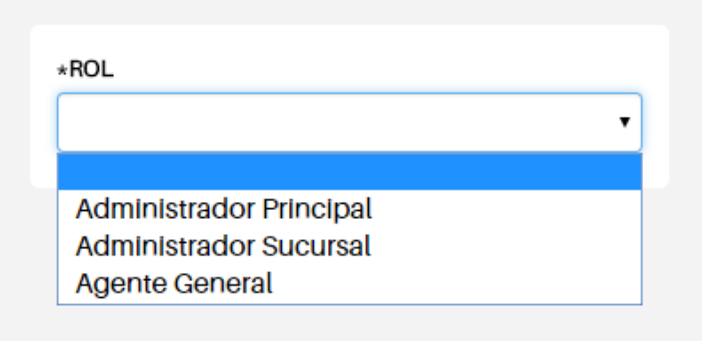

Figura 19. Lista de Roles para registro de usuarios del Administrador Principal

Se diseñó una interfaz para ingresar un servicio, especificando las características que el usuario requiere, además del punto de origen (para seleccionar el vehículo óptimo asignado la sucursal más cercana) y destino para el cálculo de la ruta para realizar el traslado, como se muestra en la Figura 20:

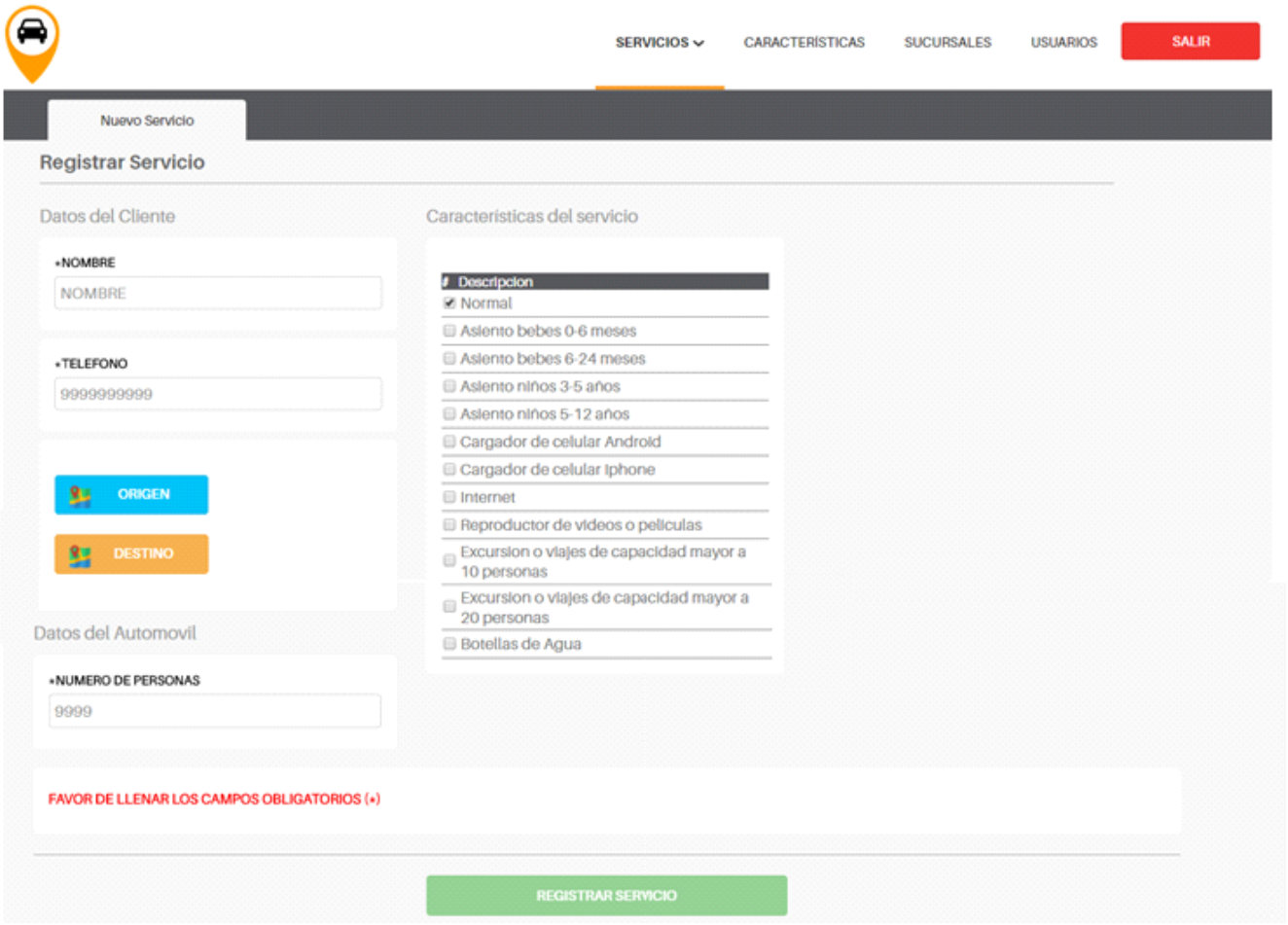

Figura 20. Interfaz de registro de Servicio

Para el monitoreo de los servicios activos de todas las sucursales desde la vista del administrador principal se diseñó una interfaz para poder gestionarlos, es decir, indicar cuándo el vehículo llega al punto de origen (posición donde abordará el usuario) y terminarlo cuando llegan al destino, como se muestra en la Figura 21.

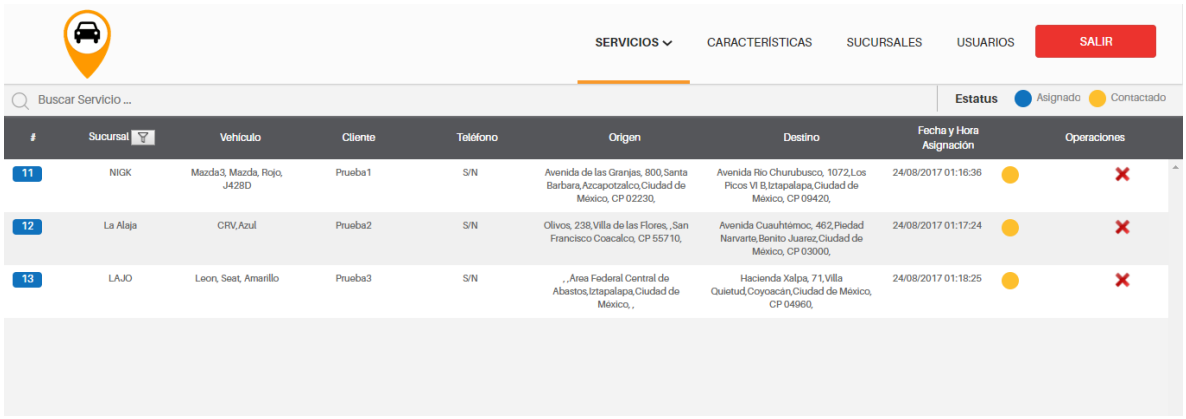

Figura 21. Interfaz de Monitoreo de Servicios activos

En la Figura 22 se muestra como se llenan los datos para acceder desde la interfaz de Login, esto con un usuario Administrador de Sucursal:

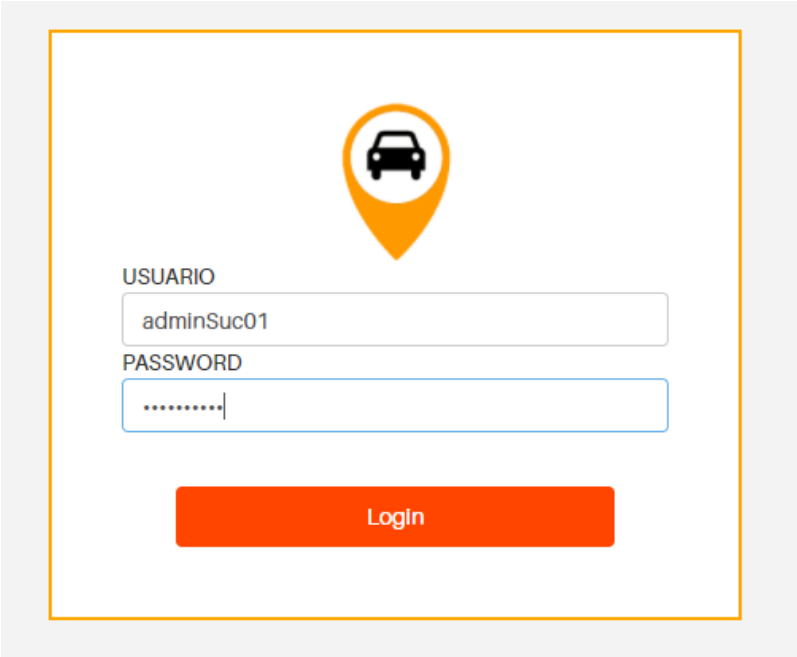

Figura 22. Llenado de Interfaz Login para acceder como usuario Administrador de Sucursal

Se diseñó una interfaz de monitoreo de servicios, para visualizar y avanzar o terminar los que tiene activos la Sucursal a la cual pertenece el administrador, como se muestra en la Figura 23:

Figura 23. Interfaz de monitoreo para Administrador de Sucursal

Se diseñó una interfaz de consulta para visualizar la lista de vehículos registrados en el sistema, pertenecientes a la sucursal que realiza el registro, como se muestra en la Figura 24 se muestran también los íconos de Editar y Eliminar:

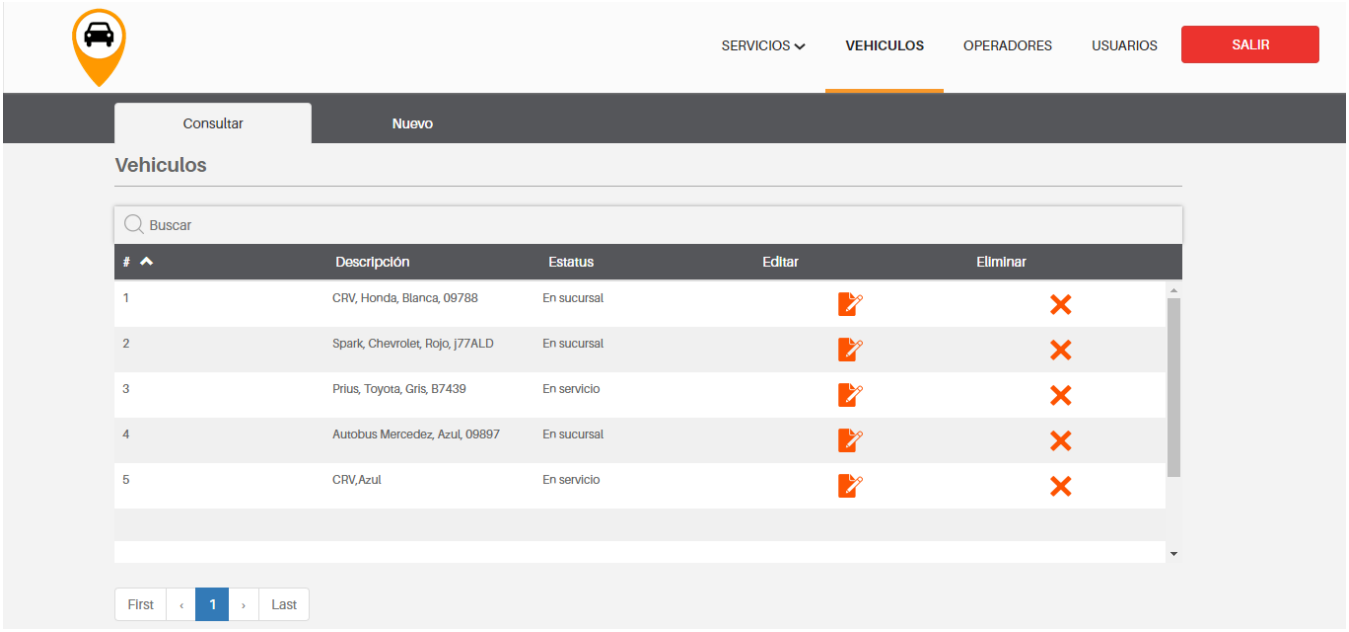

Figura 24. Interfaz de consulta para vehículos registrados de una Sucursal

Para realizar el registro o alta de un vehículo se diseñó una interfaz como se muestra en la Figura 25, en la cual se asignan las características especiales que tiene el vehículo, así como su descripción, capacidad de personas, estatus y si puede acceder a vías alternas:

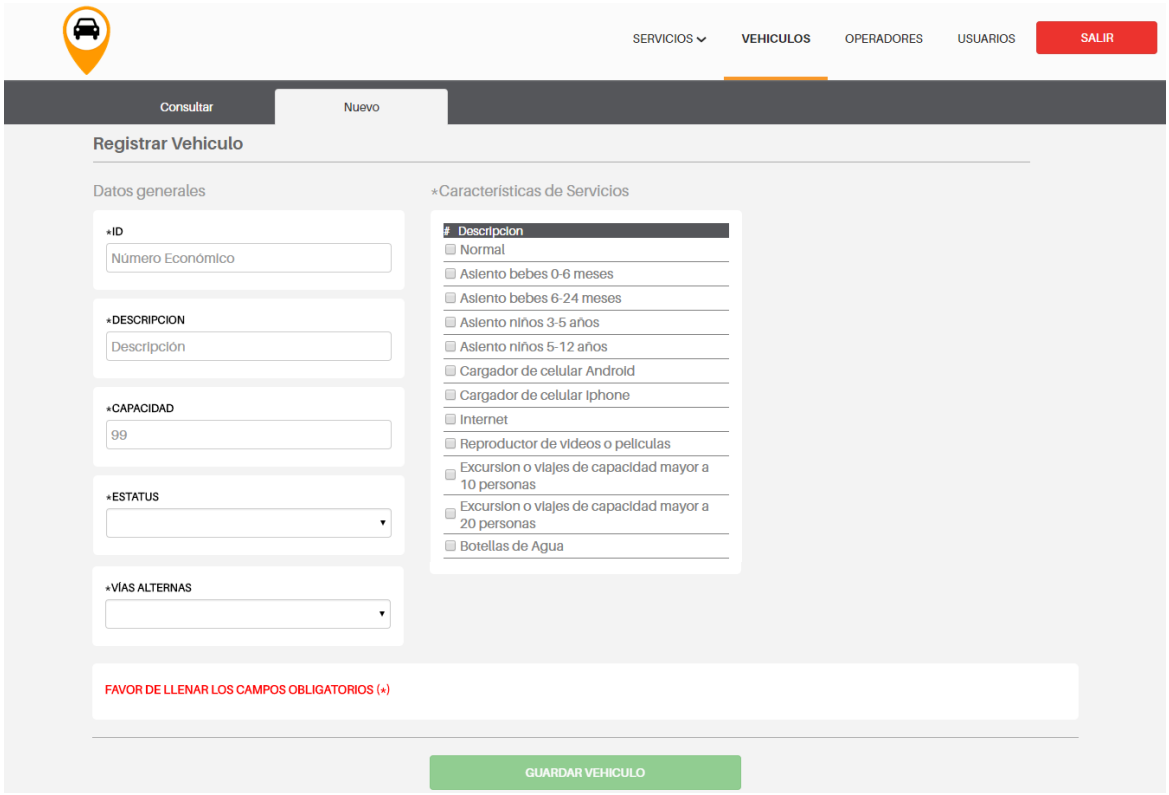

Figura 25. Interfaz de alta de vehículos para Administrador de Sucursal

Se diseñó una interfaz de consulta de los operadores que manejaran los vehículos registrados en el sistema, tal como se muestra en la Figura 26, que muestra además las opciones de Editar y Eliminar el o los registros:

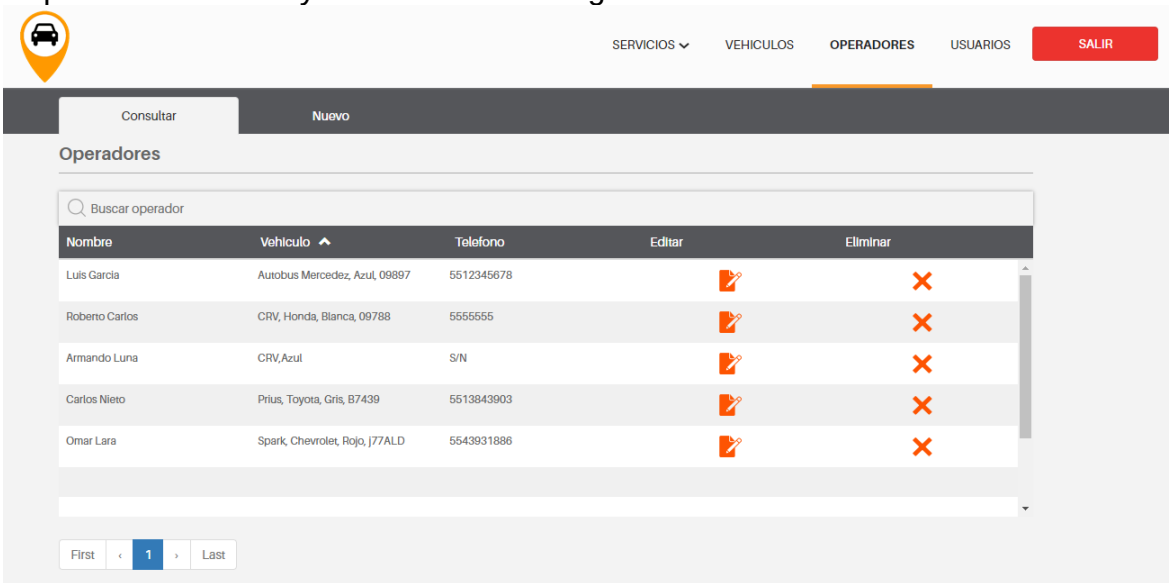

Figura 26. Interfaz de Consulta de Operadores

Para el alta o registro de operadores dentro del sistema, se diseñó una interfaz como se muestra en la Figura 27, en la cual se ingresa el nombre, el teléfono y el vehículo que tendrá asignado:

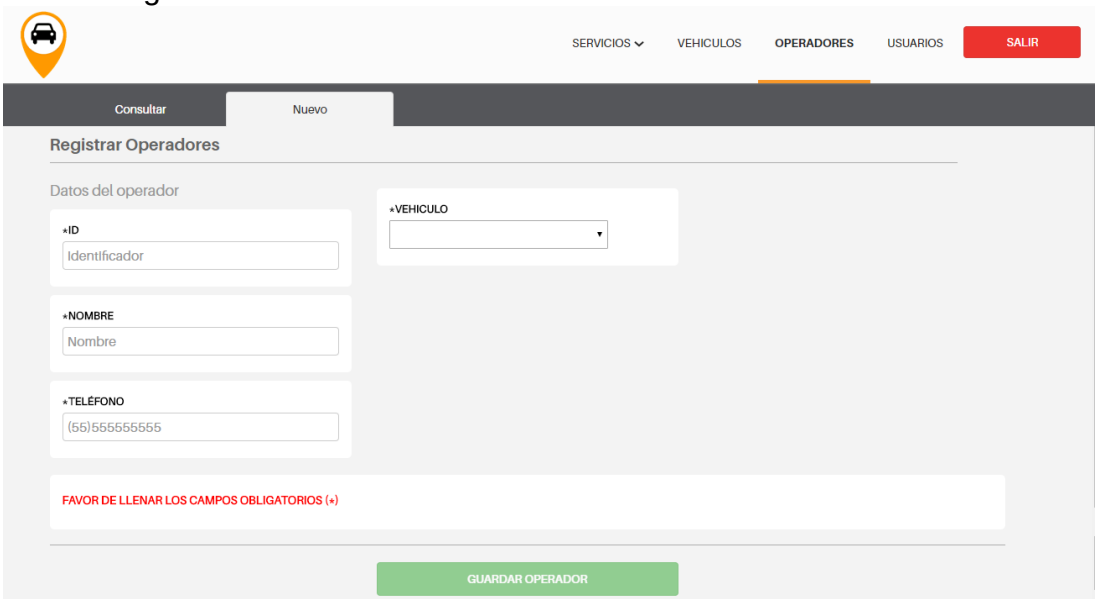

Figura 27. Interfaz para Alta de Operadores

Se diseñó una interfaz de consulta de usuarios registrados por el Administrador de Sucursal, en la que se despliega una lista como se muestra en la Figura 28, la cual incluye las opciones de Editar y Eliminar los registros:

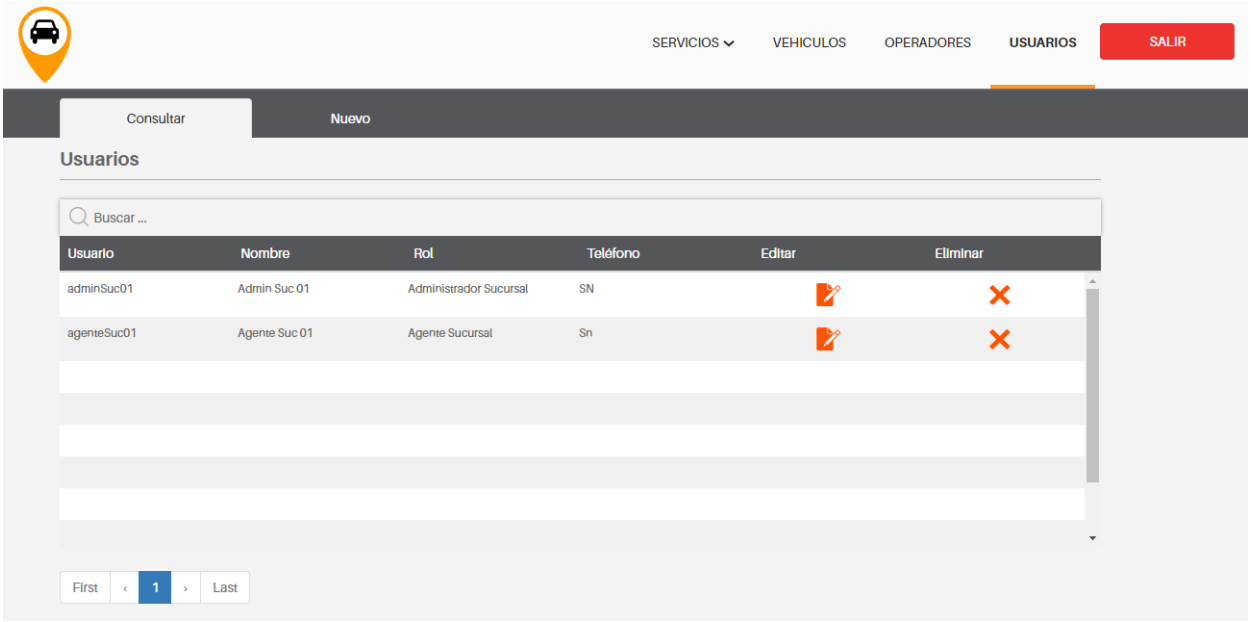

Figura 28. Interfaz de Consulta de Usuarios Registrados por Administrador de Sucursal

Para el registro de usuarios desde el rol de Administrador de Sucursal se diseñó una interfaz que incluye datos como usuario, contraseña, confirmación de contraseña, nombre, teléfono y rol, tal como se muestra en la Figura 29:

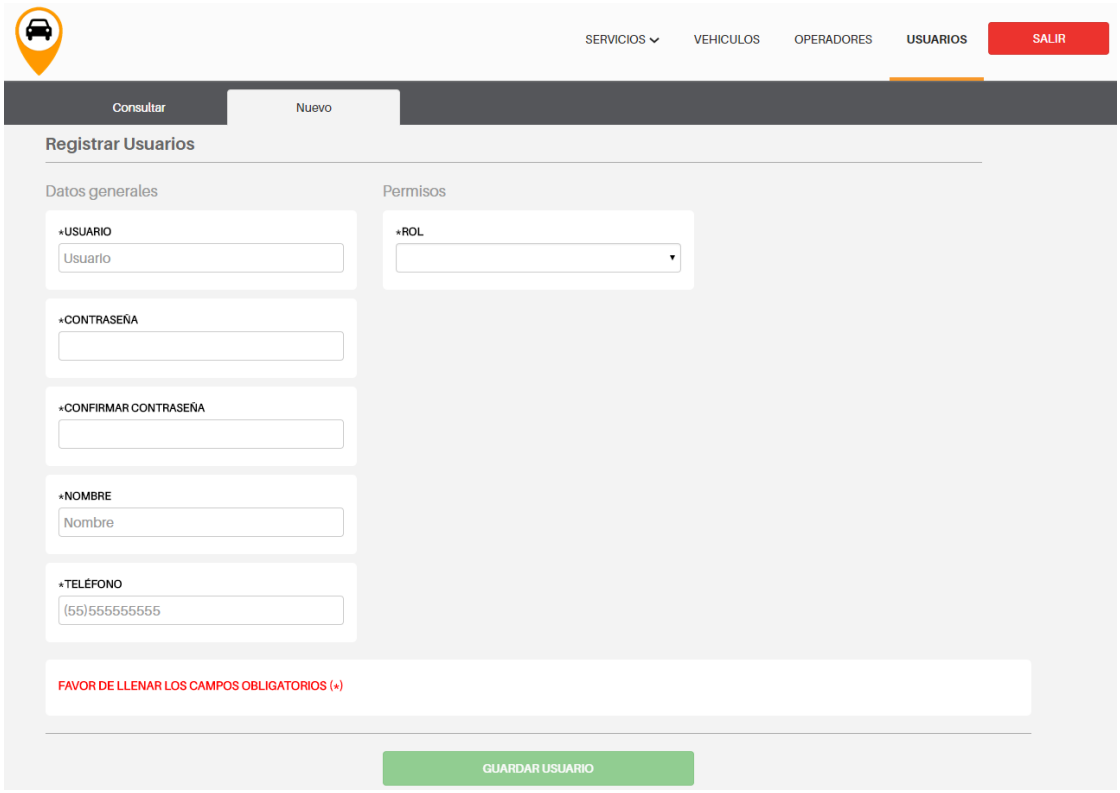

Figura 29. Interfaz de registro de usuarios para el Administrador de Sucursal

Para la opción de roles del Administrador de Sucursal, solo permite el registro del rol de Agente de Sucursal (como se muestra en la Figura 30), los cuales podrán gestionar los servicios activos y terminarlos, además de consultar el histórico de los servicios realizados:

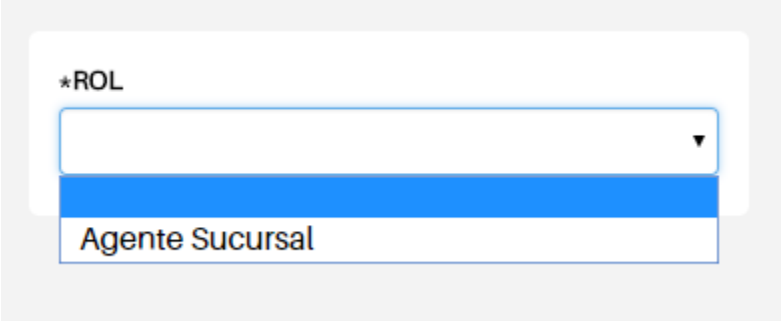

Figura 30. Lista de roles para Administrador de Sucursal

Se diseñó una interfaz para el monitoreo de servicios por parte de los agentes de sucursal, tal como se muestra en la Figura 31, donde se visualizan los datos de los servicios activos de la sucursal a la que pertenece:

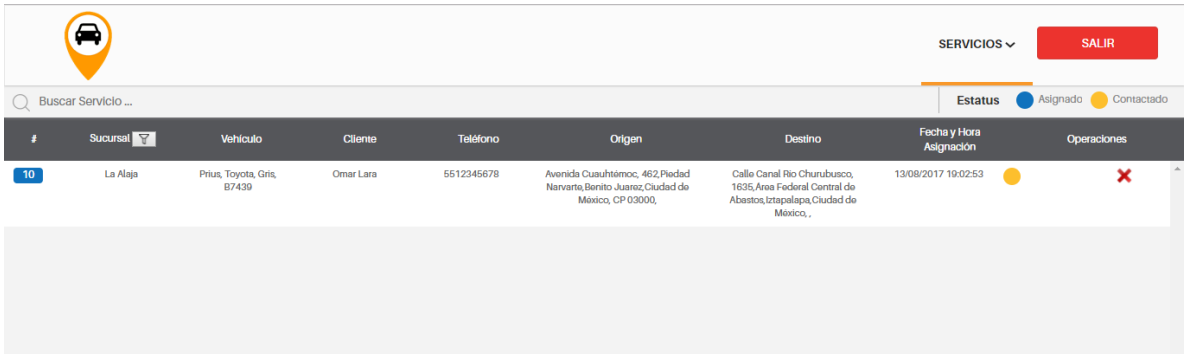

Figura 31. Interfaz de monitoreo de servicios para Agente de Sucursal

Los agentes de sucursal son capaces de avanzar los servicios, en este caso indicar cuando el vehículo llegó al punto de origen para recoger al usuario, al realizar esta acción el sistema despliega un mensaje como se muestra en la Figura 32:

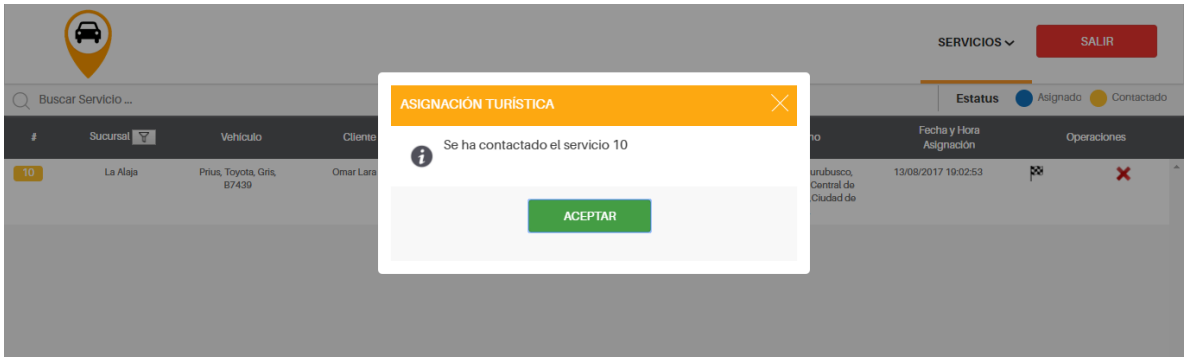

Figura 32: Interfaz que despliega mensaje de acción al Agente de Sucursal

Después de avanzar el servicio, dónde se indica que ya llegó al punto de origen el ícono cambia a una bandera como se muestra en la Figura 33, este ícono permite terminar el servicio:

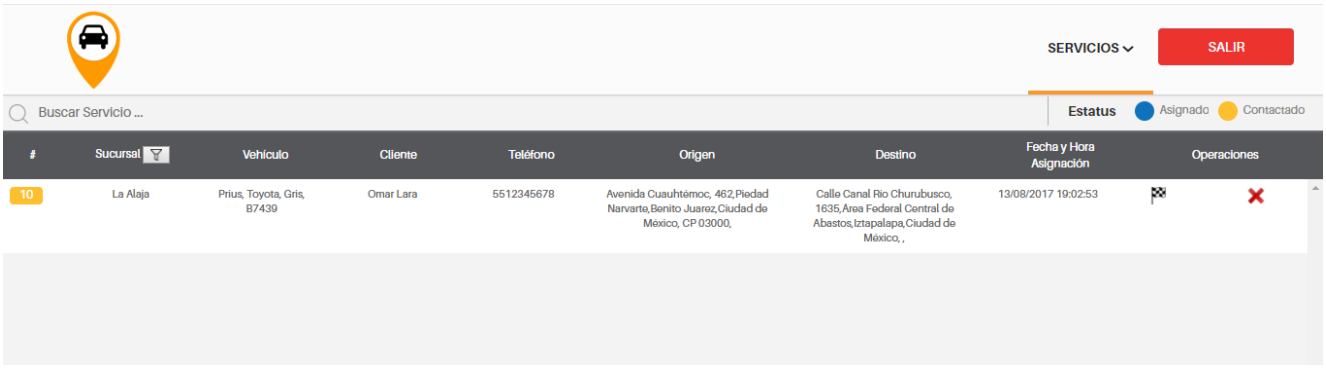

Figura 33. Interfaz que muestra ícono de término de servicio

Cuando el agente de sucursal presiona el ícono para terminar un servicio, se despliega un mensaje que indica el servicio que se está finalizando, tal como se muestra en la Figura 34:

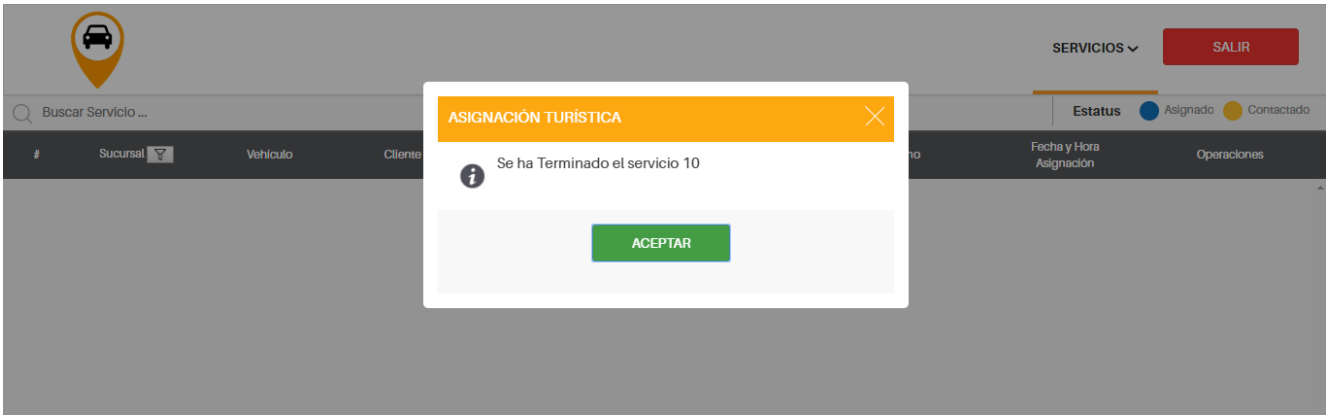

Figura 34. Interfaz que despliega mensaje de termino de Servicio

Se diseñó una interfaz como la de la Figura 35, para visualizar el historial de los servicios finalizados y cancelados, se puede especificar el rango de fecha, o solo dar click en el botón Buscar y se realizará el despliegue de los servicios:

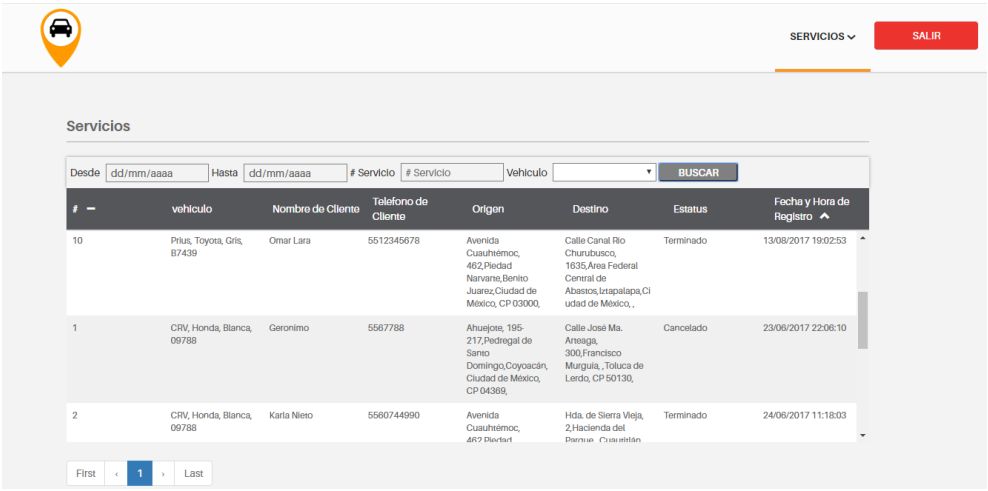

Figura 35. Interfaz para mostrar el historial de servicios

Para los agentes generales se diseñó una interfaz que permita el registro de servicios como la que se muestra en la Figura 36, donde se ingresa el nombre del cliente, su posición de origen y el destino, así como el número telefónico:

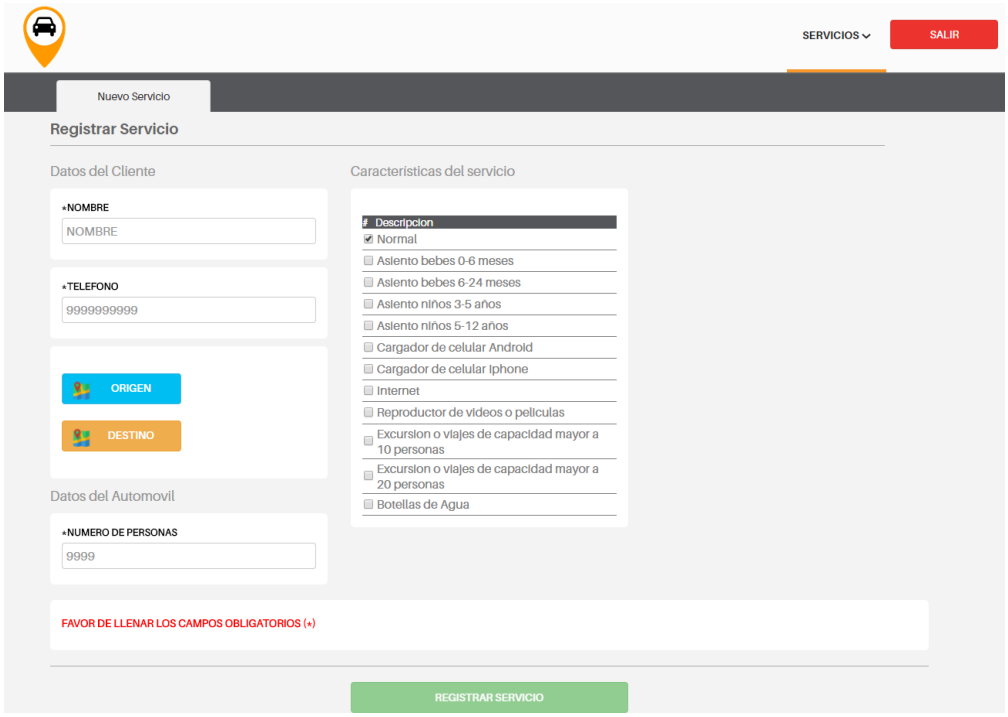

Figura 36. Interfaz de registro de servicio para agente general

Se diseñó una interfaz para el monitoreo de los servicios de todas la sucursales, donde los agentes pueden visualizar la información y qué vehículo los está atendiendo y a qué sucursal están asigandos, tal como se muestra en la Figura 37:

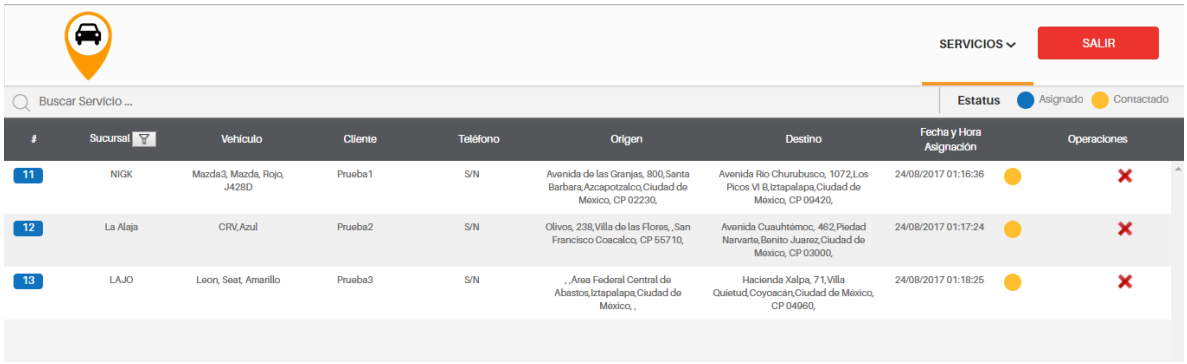

Figura 37. Interfaz de monitoreo de servicios para agentes generales

Para el rol de Agente General no está permitida la acción que avanza o termina los servicios, al intentar realizar esta acción se despliega un mensaje que notifica que no cuenta con permisos para hacerlo, este mensaje se muestra en la Figura 38:

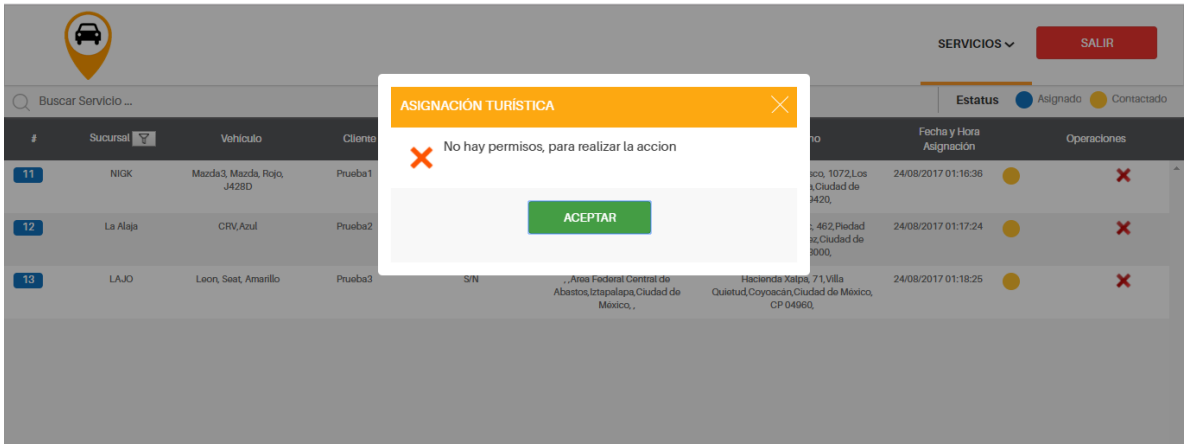

Figura 38. Interfaz que despliega el mensaje para agentes generales

DISEÑO DE LA ARQUITECTURA

Se presentan los componentes principales de la Plataforma Web, que son el motor de Asignación, el API de Google Maps, las Bases de Datos SQL y Mongo DB para el registro de operadores, vehículos, sucursales y usuarios de acceso a la plataforma, así como el registro de los servicios para el correcto monitoreo en la atención de los mismos.

Los módulos de gestión de sucursales, de los vehículos de transporte turístico, de las características de los vehículos, registran en la base de datos la información sobre qué sucursal cuenta con qué vehículos y las características que tiene cada uno. Cuando un cliente llama a la sucursal, se registra la dirección a la cual se le enviará el transporte y las características que este debe poseer. Con esta información, el módulo de despliegue de sucursales, mostrará la ubicación de las sucursales que cuenten con los vehículos que cubran los requerimientos del cliente. Ver Figura 39.

INTERACCION DE TODOS LOS COMPONENTES DE LA PLATAFORMA

Esta fase en encarga de realizar una búsqueda entre los datos registrados para verificar si alguna de las sucursales dentro del sistema cuenta con una unidad con las características que el usuario solicita para atender el servicio, y decidir cuál de ellas es la más cercana en tiempo y distancia para llegar a la posición del usuario utilizando el API de Google Maps de Matriz de Distancia[2].

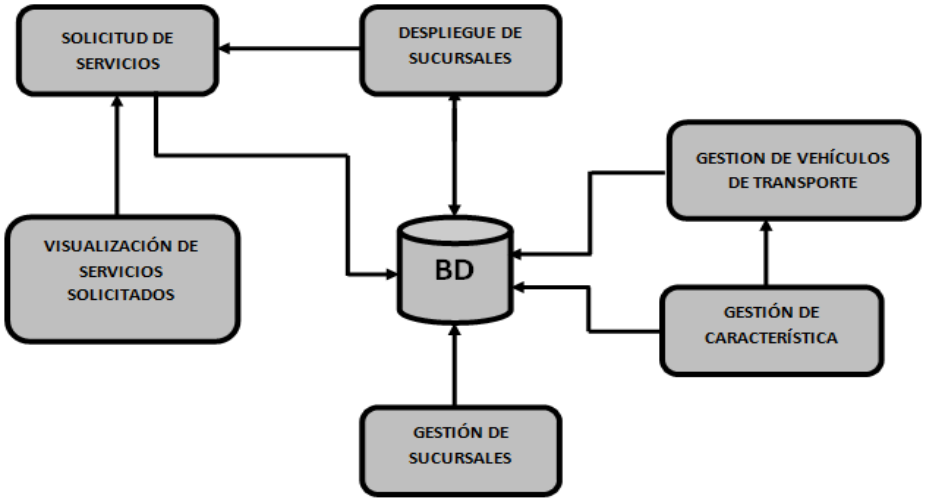

Figura 39. Interacción de los componentes y módulos de la Plataforma Digital

[2] Matriz de Distancia (Distance Matrix API) es un servicio que proporciona el tiempo y distancia de viaje de una matriz de orígenes y destinos.

CONCLUSIONES

Se concluye que el API de Google Maps es una herramienta eficaz en cuanto a trazado de rutas, estimación de tiempos y geolocalización se refiera, en el presente proyecto fue fundamental para poder realizar la mayor parte de los cálculos para la asignación de vehículos, esto implica una mejora en la calidad de atención y tiempos de respuesta a los usuarios, minimizando retrasos y desvíos del punto de origen en el cual se abordará el vehículo asignado (posición del usuario).

De los objetivos que se plantearon al inicio del proyecto se puede concluir que se cumplió con ellos, ya que:

- Se implementó un módulo para el registro correspondiente los servicios, en el cual se especifican las características solicitadas por el usuario al momento de realizar la petición del servicio.

- Se implementó un módulo en el cual se puede visualizar la lista de servicios registrados con detalles como origen (posición en la cual se recogerá al usuario), destino (lugar al cual se trasladará al usuario), vehículo que atiende el servicio…

- Se implementó un módulo en el cual se realiza el registro de sucursales y se permite especificar datos como nombre, teléfono, ubicación.

- Se implementó un módulo que contiene el despliegue de un mapa donde se pueden visualizar las sucursales registradas en el sistema.

- Se desarrolló un módulo para realizar el registro de los vehículos y la asignación de las características específicas con las que contará, en este módulo se podrá actualizar y borrar la información registrada.

- Se desarrolló un módulo para registrar las características con las cuales podrá contar cada vehículo, en este apartado, se puede realizar la edición y el borrado de cada característica.

- Se implementó un Motor de Asignación para seleccionar el vehículo óptimo para atender un servicio, teniendo como datos de entrada posición del usuario y las características que requiere tenga el vehículo que lo trasladará y la posición de las sucursales y los datos específicos de los vehículos con los que cuenta.

BIBLIOGRAFÍA

[1] L. A. Carrasco Reyes, “Sistema de información Web para la Geolocalización de Unidades Aéreas”, proyecto terminal, División de Ciencias Básicas e Ingeniería, Universidad Autónoma Metropolitana Azcapotzalco, México, 2015.

[2] J. M. Chui Guzmán, “Búsqueda de rutas más cortas entre diferentes sistemas de transporte en la Ciudad de México aplicando el enfoque del Modelo GT”, proyecto terminal, División de Ciencias Básicas e Ingeniería, Universidad Autónoma Metropolitana Azcapotzalco, México, 2011.

[3] E. Gómez Núñez, M. A. García Gómez, “Sistema de monitoreo para vehículos con una ruta predeterminada mediante GPS”, proyecto terminal, División de Ciencias Básicas e Ingeniería, Universidad Autónoma Metropolitana Azcapotzalco, México, 2007.

[4] Eldiario.es, “Uber”, http://www.eldiario.es/turing/apps/Uber-apps-taxi_0_266323526.html, 2014.

[5] EASY, “Easy Taxi”, http://www.easytaxi.com/mx/pasajero/, 2017.

[6] PC WORLD digital, “Waze”, http://www.idg.es/pcworld/estructura/VersionImprimir.asp?idArticulo=207893, 2013.

APÉNDICE A (Código Fuente)

CONFIGURACIÓN DE VARIABLE DE ENTORNO FILE_CONFIG_WS

Se acceden a las propiedades del equipo, posteriormente se da click en la opción **Configuración avanzada del sistema**, se despliega una ventana emergente y se seleccionará la opción **Variables de entorno…** Tal como se muestra en la Figura 40:

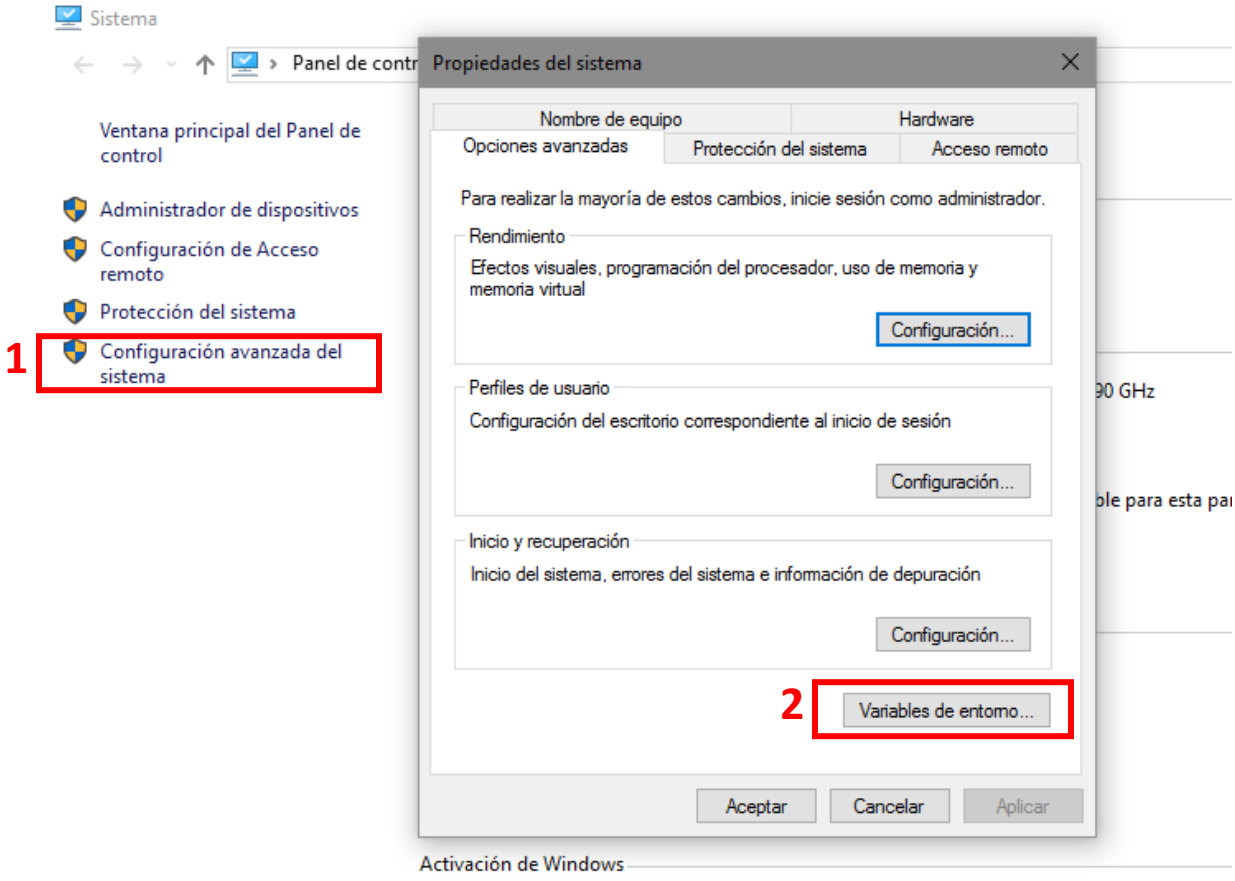

Figura 40. Propiedades del equipo y ventana emergente de Configuraciones Avanzadas

Se crea una nueva variable de entorno, en la cual se colocara la ruta de la carpeta de Archivos de Configuración dentro del proyecto, esto se puede ver en la Figura 41:

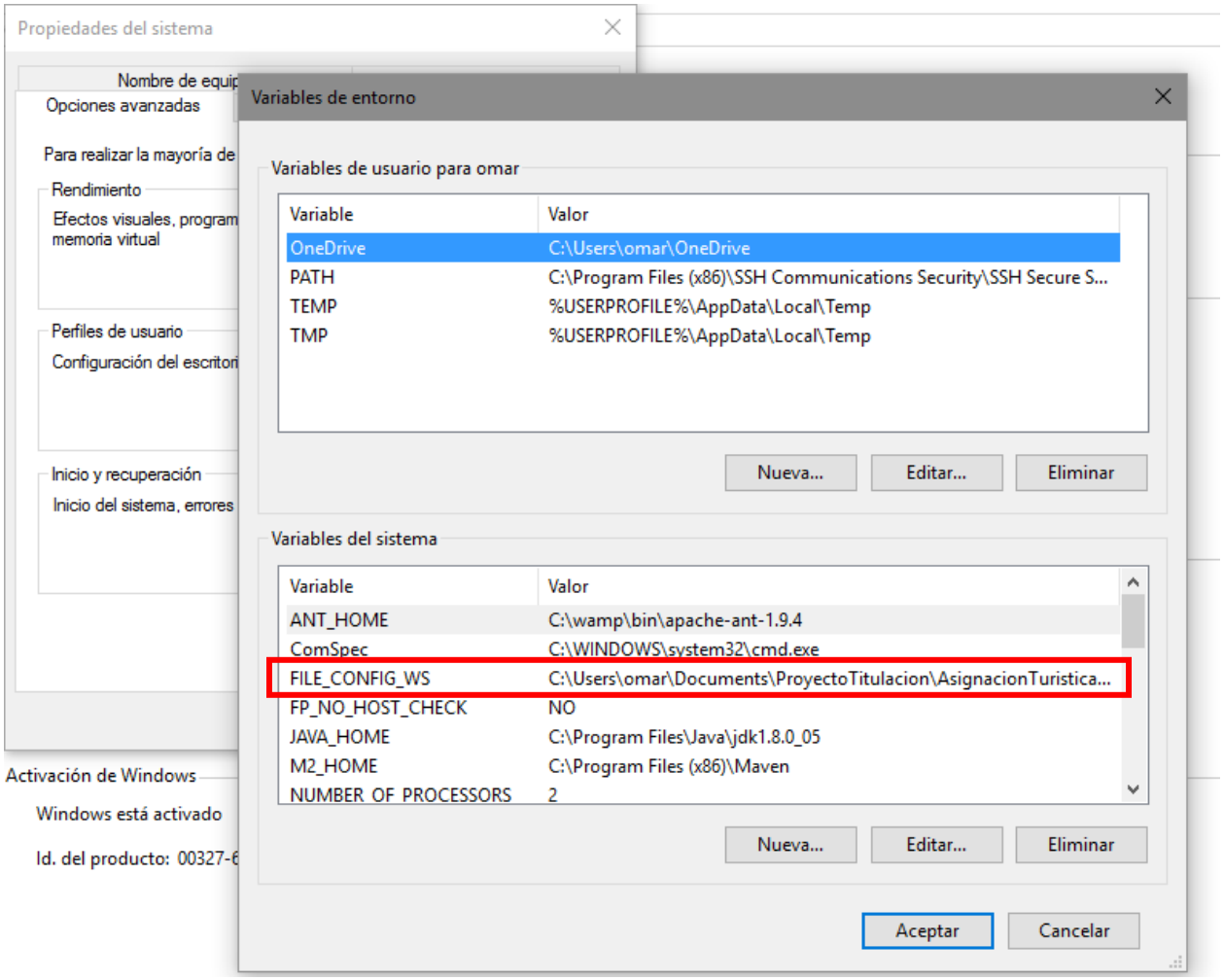

Figura 41. Variable de entorno para acceso a archivos de configuración

ARCHIVO DE CONFIGURACIÓN

MSAsignaciónServiciosConfig.properties

Se especifican las conexiones a SQL y Mongo DB, así como los mensajes de error, configuraciones adicionales para el API de Google Maps y la ruta donde se colocarán los logs de la Plataforma

```
spring.application.name= asignacion-turistica
logging.level.org.springframework.web=DEBUG
spring.data.mongodb.uri= mongodb://localhost/AsignacionTuristica
server.port= 1000
spring.datasource.url= jdbc:mysql://localhost:3306/ASIGNACION_TURISTICA
spring.datasource.username= root
spring.datasource.password= Pruebas123
spring.datasource.driver-class-name= com.mysql.jdbc.Driver
#spring.jpa.show-sql=true
# Configuraciones de la aplicacion extras
com.ms.properties.google.activar-tiempo-holgura = OFF
com.ms.properties.google.tiempo-holgura = 10
com.ms.properties.google.print-log = ON
com.ms.properties.logs.path =
C:\\Users\\omar\\Documents\\ProyectoTitulacion\\AsignacionTuristica\\Logs
\\
com.ms.properties.logs.name = MSAsignacionTuristicaLog

#Mensajes
com.ms.properties.msjs.error.nulos = Existen errores de nulos, favor de
revisar el log de la aplicacion.
com.ms.properties.msjs.error.permisos = No hay permisos, para realizar la
accion
com.ms.properties.msjs.error.sesion = El usuario no esta logueado en la
aplicacion.
com.ms.properties.msjs.error.json = El json no esta estructurado
correctamente.
```

CLASES PRINCIPALES

PrincipalApplication

```
package com.ms.main;

import javax.annotation.PostConstruct;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.orm.jpa.EntityScan;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.EnableAspectJAutoProxy;
import org.springframework.context.annotation.Import;
import org.springframework.context.annotation.PropertySource;
import org.springframework.core.env.Environment;
import
org.springframework.data.jpa.repository.config.EnableJpaRepositories;
import
org.springframework.data.mongodb.repository.config.EnableMongoRepositorie
s;
import
org.springframework.http.converter.json.MappingJackson2HttpMessageConvert
er;

import com.fasterxml.jackson.databind.ObjectMapper;
import com.ms.commons.utils.PropertiesConstants;
import com.novasolution.log.config.InitConfig;

@SpringBootApplication(scanBasePackages = "com.*")
@PropertySource("file:${FILE_CONFIG_WS}MSAsignacionServiciosConfig.proper
ties")
@EnableMongoRepositories("com.*")
@Import({ MotorAsignacionSecurityConfiguration.class })
@EnableJpaRepositories("com.ms.sql.repository")
@EntityScan("com.ms.sql.dto")
@EnableAspectJAutoProxy(proxyTargetClass = true)
public class PrincipalApplication {
    @Autowired private Environment configProperties;

    public static void main(String[] args) {
        SpringApplication.run(PrincipalApplication.class, args);
    }
    public @Bean ObjectMapper objectMapper(){
        MappingJackson2HttpMessageConverter messageConverter = new
MappingJackson2HttpMessageConverter();
        return messageConverter.getObjectMapper();
    }

    @PostConstruct
    public void configuracion(){

InitConfig.setPathLog(configProperties.getProperty(PropertiesConstants.LO
GS_PATH));
```

```
InitConfig.setNameLog(configProperties.getProperty(PropertiesConstants.LO
GS_NAME));
        String pathFileLog =
configProperties.getProperty(PropertiesConstants.LOGS_PATH) +
configProperties.getProperty(PropertiesConstants.LOGS_NAME) + "-
server.log";
        System.out.println(pathFileLog);
        /*try {
            System.setOut(new PrintStream(new File(pathFileLog)));
        } catch (Exception e) {
            e.printStackTrace();
        }*/
    }
}
```

MotorAsignacionSecurityConfiguration

```
package com.ms.main;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.authentication.AuthenticationProvider;
import
org.springframework.security.authentication.dao.DaoAuthenticationProvider
;
import
org.springframework.security.config.annotation.authentication.builders.Au
thenticationManagerBuilder;
import
org.springframework.security.config.annotation.method.configuration.Enabl
eGlobalMethodSecurity;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWe
bSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSecur
ityConfigurerAdapter;
import org.springframework.security.core.userdetails.UserDetailsService;
import
org.springframework.security.web.access.channel.ChannelProcessingFilter;
import
org.springframework.security.web.authentication.UsernamePasswordAuthentic
ationFilter;
import
org.springframework.security.web.util.matcher.AntPathRequestMatcher;
import org.springframework.session.ExpiringSession;
import org.springframework.session.web.http.CookieHttpSessionStrategy;
import org.springframework.session.web.http.HttpSessionStrategy;
```

```
import org.springframework.session.web.http.SessionRepositoryFilter;

/**
 * Clase para configurar spring security
 * @author Be
 *
 */
@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(securedEnabled = true, prePostEnabled = true)
public class MotorAsignacionSecurityConfiguration extends
WebSecurityConfigurerAdapter {
        private static final String LOGIN_PATH = "/usuarios/login";

        @Autowired private AsignacionUserDetailsService
userDetailsService;
        @Autowired private HttpAuthenticationEntryPoint
authenticationEntryPoint;
        @Autowired private AuthSuccessHandler authSuccessHandler;
        @Autowired private AuthFailureHandler authFailureHandler;
        @Autowired private HttpLogoutSuccessHandler logoutSuccessHandler;
        @Autowired private JPASessionRepository sessionRepository;

        @Bean
        public HttpSessionStrategy httpSessionStrategy() {
            return new CookieHttpSessionStrategy();
        }

        @Bean
        public SessionRepositoryFilter<ExpiringSession>
sessionRepositoryFilter() {
            HttpSessionStrategy hss = httpSessionStrategy();
            /*Se agrega para no causar ningun efecto en los desarrollos
que hayan realizado, pero no es lo mejor, ya que se debe dejar que
mantenga el nombre por default*/
            ((CookieHttpSessionStrategy)hss).setCookieName("JSESSIONID");
            SessionRepositoryFilter<ExpiringSession>
sessionRepositoryFilter = new
SessionRepositoryFilter<>(this.sessionRepository);
            sessionRepositoryFilter.setHttpSessionStrategy(hss);
            return sessionRepositoryFilter;
        }

        @Bean
        @Override
        public AuthenticationManager authenticationManagerBean() throws
Exception {
            return super.authenticationManagerBean();
        }

        @Bean
        @Override
        public UserDetailsService userDetailsServiceBean() throws
Exception {
            return super.userDetailsServiceBean();
        }
```

```
        @Bean
        public AuthenticationProvider authenticationProvider() {
            DaoAuthenticationProvider authenticationProvider = new
DaoAuthenticationProvider();
            authenticationProvider.setHideUserNotFoundExceptions(false);

authenticationProvider.setUserDetailsService(userDetailsService);
            //authenticationProvider.setPasswordEncoder(new
ShaPasswordEncoder());
            return authenticationProvider;
        }

        @Override
        protected void configure(AuthenticationManagerBuilder auth)
throws Exception {
            auth.authenticationProvider(authenticationProvider());
        }

        @Override
        protected AuthenticationManager authenticationManager() throws
Exception {
            return super.authenticationManager();
        }

        private FilterSecurity createFilterSecurity() throws Exception{
            FilterSecurity filter = new FilterSecurity();

filter.setAuthenticationManager(this.authenticationManagerBean());
            filter.setRequiresAuthenticationRequestMatcher(new
AntPathRequestMatcher(LOGIN_PATH,"POST"));
            filter.setAuthenticationSuccessHandler(authSuccessHandler);
            filter.setAuthenticationFailureHandler(authFailureHandler);
            return filter;
        }

        @Override
        protected void configure(HttpSecurity http) throws Exception {
            http.csrf()
                    .disable()
                    .addFilterBefore(createFilterSecurity(),
UsernamePasswordAuthenticationFilter.class)
                    .addFilterBefore(sessionRepositoryFilter(),
ChannelProcessingFilter.class)
                    .authenticationProvider(authenticationProvider())
                    .exceptionHandling()
                    .authenticationEntryPoint(authenticationEntryPoint)
                    .and()
                    .logout()
                    .permitAll()
                    .logoutRequestMatcher(new
AntPathRequestMatcher(LOGIN_PATH, "DELETE"))
                    .logoutSuccessHandler(logoutSuccessHandler)
                    .and()
                    .sessionManagement()
                    .maximumSessions(1)
                    ;
            http.authorizeRequests();
```

```
        }
}
```

AsignacionUserDetailsService

```
package com.ms.main;

import java.util.ArrayList;
import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.GrantedAuthority;
import
org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import
org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;

import com.ms.commons.dto.User;
import com.ms.commons.utils.Constants;
import com.ms.sql.dto.Usuario;
import com.ms.sql.repository.UsuarioRepository;

/**
 * Clase para poder validar un usuario
 * @author Be
 *
 */
@Service
public class AsignacionUserDetailsService implements UserDetailsService {
    @Autowired private UsuarioRepository usuarioRepository;

    /**
     * Metodo que sirve para consultar un usuario
     */
    public UserDetails loadUserByUsername(String usuario) throws
UsernameNotFoundException {
        Usuario userDB = usuarioRepository.findUsuarioByAlias(usuario);
        if (userDB !=null){

            User user = new User();
            user.setLogin(usuario);
            user.setPassword(userDB.getPassword());
            user.setFirstName(userDB.getNombre());
            user.setLastName(userDB.getTelefono());
            user.setIdRol(userDB.getRol());
            List<GrantedAuthority> grantedAuthorities = new
ArrayList<>();
            if(userDB.getRol()==Constants.ROL_ADMIN){
                grantedAuthorities.add(new
SimpleGrantedAuthority("ROLE_ADMIN"));
            }else if(userDB.getRol()==Constants.ROL_ADMIN_SUCURSAL){
                grantedAuthorities.add(new
SimpleGrantedAuthority("ROLE_ADMIN_SUCURSAL"));
```

```
                user.setSucursal(userDB.getSucursal());
            }else if(userDB.getRol()==Constants.ROL_AGENTE_GENERAL){
                user.setSucursal(userDB.getSucursal());
                grantedAuthorities.add(new
SimpleGrantedAuthority("ROLE_AGENTE_GENERAL"));
            }else if(userDB.getRol()==Constants.ROL_AGENTE_SUCURSAL){
                user.setSucursal(userDB.getSucursal());
                grantedAuthorities.add(new
SimpleGrantedAuthority("ROLE_AGENTE_SUCURSAL"));
            }

            return new AsignacionUserDetails(user, grantedAuthorities);
        }
        return null;
    }
}
```

HttpAuthenticationEntryPoint

```
package com.ms.main;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.web.AuthenticationEntryPoint;
import org.springframework.stereotype.Component;

import com.fasterxml.jackson.databind.ObjectMapper;
import com.ms.commons.dto.BaseResponseDto;

/**
 * Clase que genera una respuesta cuando un usuario trata de realizar una
accion a la cual no tiene permisos de ejecucion
 * @author Be
 *
 */
@Component
public class HttpAuthenticationEntryPoint implements
AuthenticationEntryPoint {
    @Autowired ObjectMapper objectMapper;

    /**
     * Metodo que crea la respuesta a la ejecucion de una accion a la
cual no dispone de permisos de ejecucion
     */
    @Override
    public void commence(HttpServletRequest request, HttpServletResponse
response,
            AuthenticationException authException) throws IOException {
        response.setContentType("application/json");
```

```
        response.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
        BaseResponseDto resp = new BaseResponseDto();
        resp.setEstatus("E008");
        resp.setMensaje("Usuario no autizado para realizar esta
operacion");
        response.setCharacterEncoding("UTF-8");
        PrintWriter writer = response.getWriter();
        objectMapper.writeValue(writer, resp);
        writer.flush();
    }
}
```

AuthSuccessHandler

```
package com.ms.main;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.Authentication;
import
org.springframework.security.web.authentication.SavedRequestAwareAuthenti
cationSuccessHandler;
import org.springframework.stereotype.Component;

import com.fasterxml.jackson.databind.ObjectMapper;
import com.ms.commons.dto.BaseResponseDto;
import com.ms.commons.dto.RespuestaResponseDto;
import com.ms.commons.dto.User;
/**
 * Clase que sirve para generar la respuesta a un acceso de login exitoso
 * @author Be
 *
 */
@Component
public class AuthSuccessHandler extends
SavedRequestAwareAuthenticationSuccessHandler {
    @Autowired ObjectMapper objectMapper;
    /**
     * Metodo que se ejecuta automaticamente, cuando un usuario se loguea
correctamente
     */
    @SuppressWarnings("unchecked")
    @Override
    public void onAuthenticationSuccess(HttpServletRequest request,
HttpServletResponse response,
            Authentication authentication) throws IOException,
ServletException {
        BaseResponseDto resp  = null;
        response.setStatus(HttpServletResponse.SC_OK);
```

```
        response.setContentType("application/json");
        response.setCharacterEncoding("UTF-8");
        if (authentication != null) {
            Object principal = authentication.getPrincipal();
            if (principal instanceof AsignacionUserDetails) {
                resp= new RespuestaResponseDto<User>();

((AsignacionUserDetails)principal).getUser().setPassword("");
                resp.setEstatus("OK");
                resp.setMensaje("OPERACION EXITOSA");
                ((RespuestaResponseDto<User>)
resp).setRespuesta(((AsignacionUserDetails)principal).getUser());

            }else{
                request.getSession().invalidate();
                resp = new BaseResponseDto();
                resp.setEstatus("E002");
                resp.setMensaje("No fue posible acceder correctamente al
login");
            }
        }else{
            request.getSession().invalidate();
            resp = new BaseResponseDto();
            resp.setEstatus("E002");
            resp.setMensaje("No fue posible acceder correctamente al
login");
        }

        PrintWriter writer = response.getWriter();
        objectMapper.writeValue(writer, resp);
        writer.flush();
    }
}
```

AuthFailureHandler

```
package com.ms.main;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.AuthenticationException;
import
org.springframework.security.web.authentication.SimpleUrlAuthenticationFa
ilureHandler;
import org.springframework.stereotype.Component;

import com.fasterxml.jackson.databind.ObjectMapper;
import com.ms.commons.dto.BaseResponseDto;
```

```
/**
 * Clase que sirve para generar la respuesta a un error de login
 * @author Be
 *
 */
@Component
public class AuthFailureHandler extends
SimpleUrlAuthenticationFailureHandler {
    @Autowired ObjectMapper objectMapper;

    /**
     *  Metodo que se ejecuta cuando un acceso de login es ejecutado
incorrectamente
     */
    @Override
    public void onAuthenticationFailure(HttpServletRequest request,
HttpServletResponse response,
            AuthenticationException exception) throws IOException,
ServletException {
        if(exception!=null){
            exception.printStackTrace();
        }
        response.setContentType("application/json");
        response.setStatus(HttpServletResponse.SC_OK);
        response.setCharacterEncoding("UTF-
8");response.setCharacterEncoding("UTF-8");
        BaseResponseDto resp = new BaseResponseDto();
        resp.setEstatus("E008");
        resp.setMensaje("El usuario no se pudo logear. Favor de verificar
los datos. " + (exception!=null? exception.getMessage():""));
        PrintWriter writer = response.getWriter();
        objectMapper.writeValue(writer, resp);
        writer.flush();
    }
}
```

HttpLogoutSuccessHandler

```
package com.ms.main;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.Authentication;
import
org.springframework.security.web.authentication.logout.LogoutSuccessHandl
er;
import org.springframework.stereotype.Component;

import com.fasterxml.jackson.databind.ObjectMapper;
import com.ms.commons.dto.BaseResponseDto;
```

```
/**
 * Clase que genera una respuesta a la accion de logout
 * @author Be
 *
 */
@Component
public class HttpLogoutSuccessHandler implements LogoutSuccessHandler {
    @Autowired ObjectMapper objectMapper;

    /**
     * Metodo que se ejecuta cuando una accion de logout es ejecutada.
     */
    @Override
    public void onLogoutSuccess(HttpServletRequest request,
HttpServletResponse response, Authentication authentication)
            throws IOException {
        response.setContentType("application/json");
        response.setCharacterEncoding("UTF-8");
        response.setStatus(HttpServletResponse.SC_OK);
        BaseResponseDto resp = new BaseResponseDto();
        resp.setEstatus("OK");
        resp.setMensaje("OPERACION EXITOSA");
        PrintWriter writer = response.getWriter();
        objectMapper.writeValue(writer, resp);
        writer.flush();
    }
}
```

JPASessionRepository

```
package com.ms.main;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.HashMap;
import java.util.Map;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.session.ExpiringSession;
import org.springframework.session.MapSession;
import org.springframework.session.SessionRepository;
import org.springframework.stereotype.Component;
import org.springframework.transaction.annotation.Transactional;

import com.ms.mongo.dto.SessionEntity;
import com.ms.mongo.repository.SessionEntityRepository;

@Component
public class JPASessionRepository implements
SessionRepository<ExpiringSession> {
```

```
    private int defaultMaxInactiveInterval = 10*60;
    @Autowired private SessionEntityRepository sessionEntityRepository;

    @Override
    public ExpiringSession createSession() {
        ExpiringSession result = new MapSession();

result.setMaxInactiveIntervalInSeconds(defaultMaxInactiveInterval);
        return result;
    }

    @Transactional
    @Override
    public void save(ExpiringSession session) {
        try {
            sessionEntityRepository.save(convertToDomain(session));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @Transactional
    @Override
    public ExpiringSession getSession(String id) {
        SessionEntity sessionEntity = null;
        try {
            sessionEntity = sessionEntityRepository.findOne(id);
        } catch (Exception e) {
            return null;
        }
        ExpiringSession saved = sessionEntity == null ? null :
convertToSession(sessionEntity);
        if(saved == null) {
            return null;
        }
        if(saved.isExpired()) {
            delete(saved.getId());
            return null;
        }
        return saved;
    }

    @Override
    public void delete(String id) {
        try {
            sessionEntityRepository.delete(id);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private SessionEntity convertToDomain(ExpiringSession session) {
        SessionEntity sessionEntity = new SessionEntity();
        sessionEntity.setId(session.getId());
        sessionEntity.setLastAccessedTime(session.getLastAccessedTime());
        sessionEntity.setAccessTime(session.getCreationTime());
```

```
sessionEntity.setMaxInactive(session.getMaxInactiveIntervalInSeconds());
        sessionEntity.setData(serializeAttributes(session));
        return sessionEntity;
    }

    //this serialize attributes of session
    private byte[] serializeAttributes(ExpiringSession session) {
        Map<String, Object> attributes = new HashMap<>();
        for (String attrName : session.getAttributeNames()) {
            attributes.put(attrName, session.getAttribute(attrName));

        }
        ByteArrayOutputStream out = new ByteArrayOutputStream();
        byte[] buffer = new byte[0];
        try {
            ObjectOutputStream objectOutputStream = new
ObjectOutputStream(out);
            objectOutputStream.writeObject(new
SessionAttributes(attributes));
            buffer = out.toByteArray();
            objectOutputStream.close();
        } catch (IOException e) {
            e.printStackTrace();
            throw new RuntimeException(e);
        }
        return buffer;
    }


    private ExpiringSession convertToSession(SessionEntity sessionEntity)
{
        MapSession mapSession = new MapSession();
        mapSession.setId(sessionEntity.getId());

mapSession.setLastAccessedTime(sessionEntity.getLastAccessedTime());
        mapSession.setCreationTime(sessionEntity.getAccessTime());

mapSession.setMaxInactiveIntervalInSeconds(this.defaultMaxInactiveInterva
l);

        SessionAttributes attributes =
deserializeAttributes(sessionEntity);
        if (attributes != null) {
            for (Map.Entry<String, Object> attribute :
attributes.getAttributes().entrySet()) {
                mapSession.setAttribute(attribute.getKey(),
attribute.getValue());
            }
        }
        return mapSession;
    }

    //this deserialize attributes from data
    private SessionAttributes deserializeAttributes(SessionEntity
sessionEntity) {
        SessionAttributes attributes = null;
```

```
        if(sessionEntity.getData() != null &&
sessionEntity.getData().length > 0) {
            try {
                ObjectInputStream objectInputStream = new
ObjectInputStream(new ByteArrayInputStream(sessionEntity.getData()));
                Object o = objectInputStream.readObject();
                attributes = (SessionAttributes) o;
                objectInputStream.close();
            } catch (IOException | ClassNotFoundException e) {
                e.printStackTrace();
            }
        }
        return attributes;
    }

    public Integer getDefaultMaxInactiveInterval() {
        return defaultMaxInactiveInterval;
    }

    public void setDefaultMaxInactiveInterval(int
defaultMaxInactiveInterval) {
        this.defaultMaxInactiveInterval = defaultMaxInactiveInterval;
    }
}
```

COLECCIONES DE MONGO DB

COLECCIÓN DE SERVICIOS

```
package com.ms.mongo.dto;

import java.io.Serializable;
import java.util.List;

import org.springframework.data.mongodb.core.mapping.Document;
import org.springframework.data.mongodb.core.mapping.Field;

@Document(collection="Servicios")
public class Servicios implements Serializable {

    /**
     *
     */
    private static final long serialVersionUID = -7764453996232422195L;
    @Field
    private Long id;

    @Field
    private Direccion origen;

    @Field
```

```
private Direccion destino;

@Field
private Cliente cliente;

@Field
private int estatus;

@Field
private String usuarioRegistro;

@Field
private String usuarioCancelacion;

@Field
private String fechaHoraRegistro;

@Field
private String fechaHoraContacto;

@Field
private String fechaHoraTermino;

@Field
private String fechaHoraCancelacion;

@Field
private Operadores operador;

@Field
private List<Integer> caracterisiticas;

@Field
private int numeroPersonas;

@Field
private long TEA;

public Long getId() {
    return id;
}

public void setId(Long id) {
    this.id = id;
}

public Direccion getOrigen() {
    return origen;
}

public void setOrigen(Direccion origen) {
    this.origen = origen;
}

public Direccion getDestino() {
    return destino;
}
```

```
public void setDestino(Direccion destino) {
    this.destino = destino;
}

public Cliente getCliente() {
    return cliente;
}

public void setCliente(Cliente cliente) {
    this.cliente = cliente;
}

public int getEstatus() {
    return estatus;
}

public void setEstatus(int estatus) {
    this.estatus = estatus;
}

public String getUsuarioRegistro() {
    return usuarioRegistro;
}

public void setUsuarioRegistro(String usuarioRegistro) {
    this.usuarioRegistro = usuarioRegistro;
}

public String getUsuarioCancelacion() {
    return usuarioCancelacion;
}

public void setUsuarioCancelacion(String usuarioCancelacion) {
    this.usuarioCancelacion = usuarioCancelacion;
}

public String getFechaHoraRegistro() {
    return fechaHoraRegistro;
}

public void setFechaHoraRegistro(String fechaHoraRegistro) {
    this.fechaHoraRegistro = fechaHoraRegistro;
}

public String getFechaHoraContacto() {
    return fechaHoraContacto;
}

public void setFechaHoraContacto(String fechaHoraContacto) {
    this.fechaHoraContacto = fechaHoraContacto;
}

public String getFechaHoraTermino() {
    return fechaHoraTermino;
}
```

```
    public void setFechaHoraTermino(String fechaHoraTermino) {
        this.fechaHoraTermino = fechaHoraTermino;
    }

    public String getFechaHoraCancelacion() {
        return fechaHoraCancelacion;
    }

    public void setFechaHoraCancelacion(String fechaHoraCancelacion) {
        this.fechaHoraCancelacion = fechaHoraCancelacion;
    }

    public Operadores getOperador() {
        return operador;
    }

    public void setOperador(Operadores operador) {
        this.operador = operador;
    }

    public List<Integer> getCaracterisiticas() {
        return caracterisiticas;
    }

    public void setCaracterisiticas(List<Integer> caracterisiticas) {
        this.caracterisiticas = caracterisiticas;
    }

    public int getNumeroPersonas() {
        return numeroPersonas;
    }

    public void setNumeroPersonas(int numeroPersonas) {
        this.numeroPersonas = numeroPersonas;
    }

    public long getTEA() {
        return TEA;
    }

    public void setTEA(long tEA) {
        TEA = tEA;
    }
}
```

COLECCIÓN DE SESSIONSPRING

```
package com.ms.mongo.dto;

import org.springframework.data.mongodb.core.mapping.Document;
import org.springframework.data.mongodb.core.mapping.Field;

@Document(collection="SessionSpring")
```

```
public class SessionEntity {

    @Field
    private String id;
    @Field
    private Long accessTime;
    @Field
    private Long lastAccessedTime;
    @Field
    private int maxInactive;
    @Field
    private byte[] data;

    public SessionEntity() {
    }

    public SessionEntity(String id, long accessTime, long
lastAccessedTime, int maxInactive, byte[] data) {
        this.id = id;
        this.accessTime = accessTime;
        this.lastAccessedTime = lastAccessedTime;
        this.data = data;
        this.maxInactive = maxInactive;
    }

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public Long getAccessTime() {
        return accessTime;
    }

    public void setAccessTime(Long accessTime) {
        this.accessTime = accessTime;
    }

    public Long getLastAccessedTime() {
        return lastAccessedTime;
    }

    public void setLastAccessedTime(Long lastAccessedTime) {
        this.lastAccessedTime = lastAccessedTime;
    }

    public int getMaxInactive() {
        return maxInactive;
    }

    public void setMaxInactive(int maxInactive) {
        this.maxInactive = maxInactive;
    }
```

```
    public byte[] getData() {
        return data;
    }

    public void setData(byte[] data) {
        this.data = data;
    }
}
```

REPOSITORIOS DE MONGO DB

REPOSITORIO DE SERVICIOS

```
package com.ms.mongo.repository;

import java.util.List;

import org.springframework.data.mongodb.repository.MongoRepository;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;

import com.ms.mongo.dto.Servicios;

/**
 * Interfaz de persistencia en mongo de el documento ApiKey
 * @author Be
 *
 */
@Repository
@Transactional
public interface ServiciosRepository extends MongoRepository<Servicios,
Long>{
    List<Servicios> findByOperadorIdSucursal(int idSucursal);
    List<Servicios>
findByOperadorIdSucursalAndOperadorVehiculoIdVehiculo(int idSucursal, int
idVehiculo);
}
```

REPOSITORIO DE SESSIONSPRING

```
package com.ms.mongo.repository;

import java.util.List;

import org.springframework.data.mongodb.repository.MongoRepository;
import org.springframework.data.mongodb.repository.Query;
import org.springframework.stereotype.Repository;
```

```java
import org.springframework.transaction.annotation.Transactional;

import com.ms.mongo.dto.SessionEntity;

/**
 * Interfaz de persistencia en mongo de el documento ApiKey
 * @author Be
 *
 */
@Repository
@Transactional
public interface SessionEntityRepository extends
MongoRepository<SessionEntity, String>{
    @Query(value = "{'$where':'(NumberLong(?0) - this.lastAccessedTime) >
NumberLong(?1)'}", delete = true)
    List<SessionEntity> deleteByDateActualAndFecha(long timeActual, long
diferencia);
}
```