UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Sistema Integral aplicado a la mejora de la movilidad en Ciudad Universitaria

TESIS

Que para obtener el título de

Ingeniero en Computación

P R E S E N T A

Cristian Jovany Vargas Flores

DIRECTOR DE TESIS

Ing. Jorge Alberto Rodríguez Campos

Ciudad Universitaria, Cd. Mx., 2020

Contenido

DEDICATORIA.

Dedico el presente trabajo a todas las personas que me han apoyado a lograr este objetivo el cual es importante en mi formación profesional. A quien fuera mi padre y abuelo Ernesto Flores Montoya quien fue el pilar más importante para lograr la conclusión de mis estudios académicos y que a lo largo de mi vida me hizo ver la importancia de estudiar, quien me apoyo para salir adelante en las circunstancias de cualquier índole.

De igual manera, le dedico dicho trabajo a mi madre Juana Flores Ramírez y a mi abuela Inés Ramírez Pineda por brindarme su apoyo y ánimos a lo largo de mi vida, a ellas quienes me han sabido aconsejar para tomar buenas decisiones, agradezco los desvelos y preocupaciones que tuvimos juntos. A estas tres personas les estoy infinitamente agradecido por todo su apoyo incondicional y por haber forjado al hombre que soy ahora.

Así mismo, les dedico el trabajo a mis hermanos, padrinos y tíos que siempre han visto por mi bienestar y siempre estar cuando los necesito.

También dedico este trabajo a mis amigos Irvin, Efrén, Carlos, Yaritza y Carolina que conocí dentro de la universidad y que ellos saben los sacrificios que se deben de tomar al estudiar en la máxima casa de estudios: desvelos, momentos difíciles en la escuela, pero también momentos divertidos y que han hecho más amena mi estancia dentro de la universidad.

A todos los profesores que he tenido por brindar su conocimiento con la finalidad de que otras personas puedan adquirirlos, principalmente a mi director el Ingeniero Jorge Alberto Campos Rodríguez por haberme brindado su apoyo y conocimientos para poder realizar este proyecto.

A todas las personas mencionadas anteriormente quiero darles las gracias por estar en mi camino y sé que están tan orgullosos como yo lo estoy.

Vargas Flores Cristian Jovany

1. INTRODUCCIÓN

1.1 Antecedentes y estructura del trabajo

El presente documento se refiere al problema que en los últimos años ha ido en aumento dentro de Ciudad Universitaria: El difícil traslado de un punto a otro dentro del campus universitario. Dicho problema es originado por diferentes causas, principalmente por el aumento de personas, tanto comunidad universitaria como visitantes, que se dan a la necesidad de acudir al campus universitario. Aunado a esto, se puede mencionar que los medios de transporte no se dan abasto para el traslado de todas las personas.

Dicho proyecto se elabora con la finalidad de brindar a las personas que se encuentran dentro de Ciudad Universitaria una herramienta para tomar la mejor decisión posible y facilite el traslado dentro del campus. Además, esto podría ayudar a evitar el embotellamiento en ciertas partes del campus buscando tanto rutas alternas como opciones de traslado.

Para fundamentar el problema que persiste dentro del campus universitario se realizó una encuesta a los alumnos de diferentes facultades dentro de Ciudad Universitaria (presentadas en el capítulo 2. Marco teórico) en donde se logra observar que los medios de transporte son de suma importancia dentro del campus.

Por otro lado, al tratarse de un desarrollo de software se implementaron las etapas del ciclo de vida del software utilizando conceptos de la metodología tradicional así como conceptos de las metodologías ágiles.

En el capítulo 2 se presenta el planteamiento del problema de movilidad dentro de Ciudad Universitaria, se analizan las soluciones existentes que tengan un mismo enfoque al presente proyecto así como algunas diferencias entre dichas soluciones, además, se realiza una breve explicación de las herramientas utilizadas en la construcción del sistema de software.

En el capítulo 3 se habla sobre la importancia de las metodologías empleadas en el desarrollo de un proyecto y se mencionan algunos ejemplos de dos tipos de metodologías (tradicionales y ágiles). Además, se definen los requerimientos funcionales y no funcionales para el sistema. Para los requerimientos funcionales se hacen uso de diagramas de casos de uso y diagramas de actividades principalmente, se presentan prototipos para el bosquejo de los componentes del sistema.

El capítulo 4 es el encargado de explicar el diseño del sistema. En este capítulo se describe la forma en que interactúan los diferentes componentes y como está constituido cada uno de ellos. Se utilizan diagramas de despliegue, diagramas de clases y diagramas de secuencia para ejemplificar dicha interacción. Se explican conceptos básicos y necesarios para realizar el diseño de la Base de datos y se hace énfasis en la diferencia de utilizar una Base de datos sin extensión espacial (PostgreSQL) con una que si tiene extensión espacial (PostGIS).

En el capítulo 5 se explican los pasos más importantes para poder instalar PostGIS, también, se explica la estructura que tendrá la Base de datos en Firebase. Este capítulo está encargado de explicar cómo se lleva a cabo la construcción del código para el sistema, pero se limitará en el proceso que se lleva a cabo para mostrar en la aplicación del usuario una ruta del Pumabús, las unidades de dicha ruta, las estaciones relacionadas a la ruta y el tiempo aproximado de llegada del Pumabús.

Finalmente en el capítulo 6, se habla sobre la importancia de realizar pruebas al sistema construido para así asegurar la calidad del software. Se ejemplifican los diferentes tipos de pruebas.

1.2 Situación actual

1.2.1. Telefonía celular en la actualidad

En los últimos años, la tecnología se ha desarrollado de manera drástica con aportes a diferentes ámbitos, como pueden ser: medicina, investigación, educación, entre otros. Una de las aportaciones que ha beneficiado a la satisfacción de las necesidades de los seres humanos han sido los teléfonos celulares. Al paso de los años nos hemos percatado de los grandes cambios que se han presentado en estos dispositivos. Las primeras versiones de los teléfonos celulares permitían a los usuarios simplemente realizar llamadas a otros usuarios; hoy en día se nos permite compartir contenido multimedia, conexión a internet, entre muchos otros servicios.

A todos estos avances tecnológicos se les han denominado “TIC” (Tecnologías de la información y comunicación) que tienen como finalidad proporcionar un fácil y mejor manejo de la información. A continuación, se presenta un análisis realizado por el INEGI (Instituto Nacional de Estadística y Geografía) del segundo trimestre el año 2016 [1]:

[1] http://www.inegi.org.mx/saladeprensa/aproposito/2017/internet2017_Nal.pdf

En dicho estudio, el INEGI menciona que la telefonía celular es un servicio de suma importancia para la población, ya que permite la comunicación efectiva con una respuesta rápida. Los resultados mostrados son de un estudio realizado a nivel nacional (México).

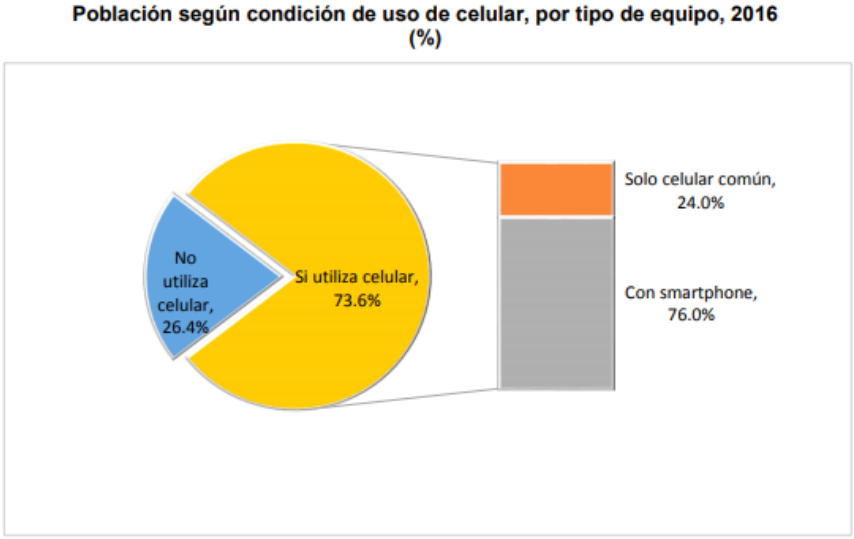


Figura 1.1 Porcentaje de la población que ocupa teléfonos celulares.

Como podemos observar en la Figura 1.1 tres cuartos de la población nacional cuentan con un teléfono celular, actualmente estas cifras han ido creciendo ya que las compañías han reducido sus costos en los dispositivos para aumentar sus utilidades y clientes.

El 76% de las personas cuentan con un Smartphone, pero ¿Qué diferencias hay entre un Smartphone y un celular común? Ambos cumplen con las funciones básicas de un teléfono celular; permiten realizar llamadas y envío de mensajes de texto (SMS). La diferencia radica en que los Smartphone permiten realizar tareas pertenecientes a una computadora como: enviar correos electrónicos, intercambiar contenido multimedia, conexión a internet, mensajes de textos por medio de aplicaciones, reproductores de música, etc. En otras palabras, los Smartphone brindan mayores servicios, es por eso por lo que la población se ha inclinado a su elección.

De las personas que cuentan con un teléfono celular, INEGI nos proporciona la siguiente gráfica:

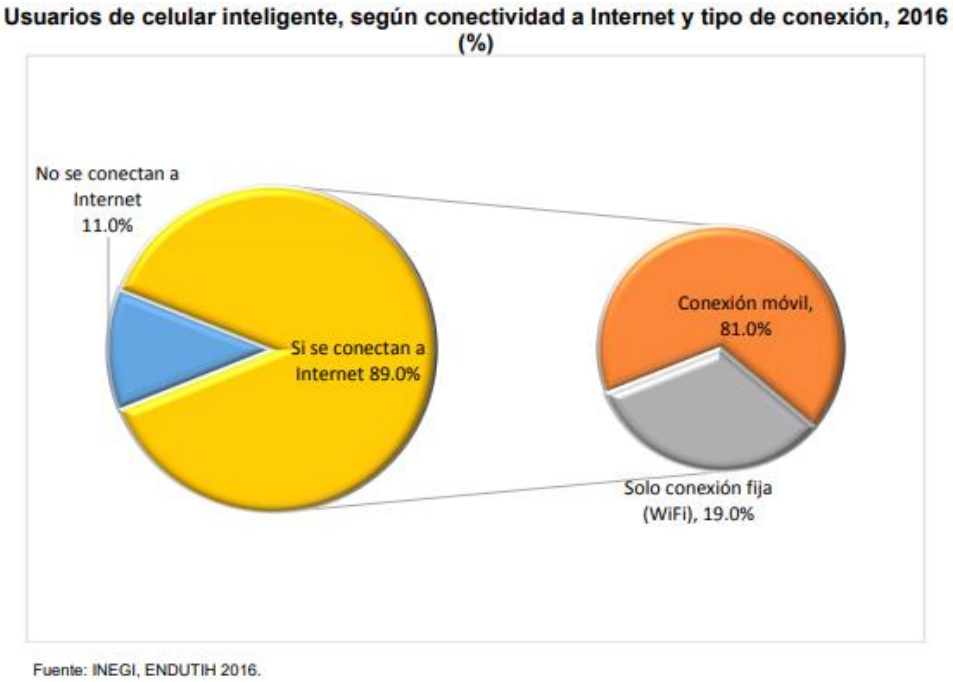


Figura 1.2 Usuarios que se pueden conectar a servicios de internet.

En la Figura 1.2 se aprecia que el 89% de las personas que cuentan con un teléfono celular se conectan a internet, probablemente esta cifra es alta ya que las compañías suelen vender los equipos con este servicio u optan por realizar promociones con la finalidad de que las personas adquieran un equipo a cambio de un costo menor a lo normal, además, hay que tomar en cuenta que la mayoría de las aplicaciones para los Smartphone necesitan de una conexión a internet, un claro ejemplo son las aplicaciones orientadas a redes sociales. Del porcentaje mencionado anteriormente el 81% de las personas tienen la capacidad de poder pagar por este servicio, mientras el restante se conecta por medio de WI-FI.

1.2.2. Medios de transporte en Ciudad Universitaria

Uno de los aspectos que se estará manejando en el presente documento son los diferentes medios de transporte que brinda la Universidad Nacional Autónoma de México (UNAM), específicamente en el campus de Ciudad Universitaria. Los medios de transporte son los siguientes:

- Pumabús.

"Es un servicio seguro y gratuito, que permite a la Comunidad Universitaria realizar traslados hacia las Facultades, Institutos, Dependencias Administrativas e Instalaciones Deportivas, a través de 12 rutas, y su recorrido exprés Filos - Unidad de Posgrado." [2]

Actualmente este servicio cuenta con:

- 50 unidades de transporte.
- 12 rutas distribuidas a lo largo de ciudad universitaria.
- Una unidad especial para realizar traslados a personas con capacidades diferentes.
- 96 estaciones dentro de Ciudad Universitaria.
- Cámaras de vigilancia dentro de las unidades del Pumabús.

- Bicipuma.

"Sistema de transporte gratuito que fomenta la movilidad sustentable y la salud de la Comunidad Universitaria." [3]

Actualmente este servicio cuenta con:

- Préstamo gratuito de bicicletas a alumnos, ex alumnos, académicos y/o trabajadores.
- 1,000 bicicletas.
- 8 km de ciclopista.
- 14 módulos.

Las siguientes encuestas[4] muestran una visión general de la importancia que tienen los servicios de transporte dentro del Campus Universitario así como el uso de las TICs a través de dispositivos móviles. La muestra fue aproximadamente de 1000 estudiantes de diferentes facultades. A continuación se presentan los resultados obtenidos:

[2] http://dgsgm.unam.mx//pumabus.html
[3] http://dgsgm.unam.mx//bicipuma.html
[4] Estas encuestas fueron desarrolladas como parte de un proyecto de investigación de la asignatura Probabilidad y Estadística y corresponden a datos del año 2015.

- Facultad a la que pertenecen los estudiantes.

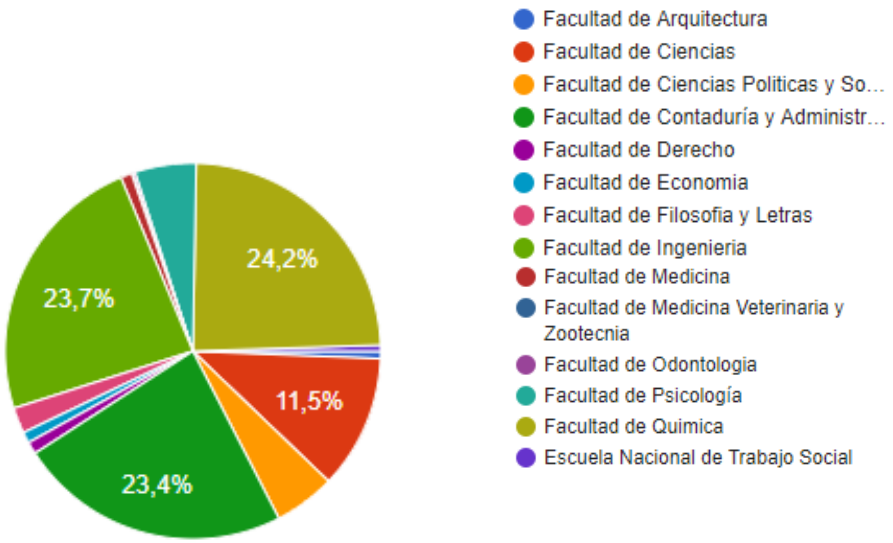


Figura 1.3 Porcentaje de los alumnos encuestados por facultad

- Sistema operativo con que cuentan los celulares de los encuestados.

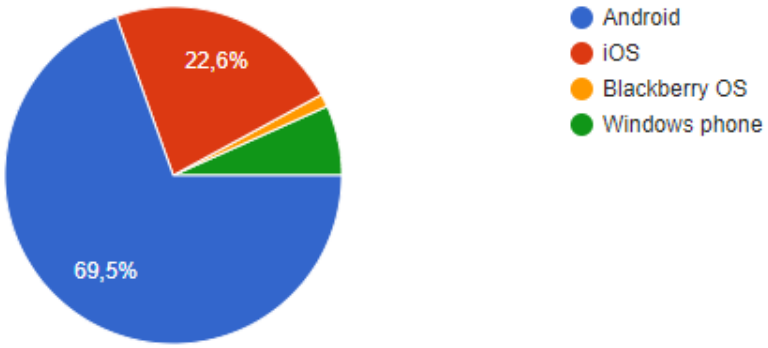


Figura 1.4 Porcentaje del sistema operativo que cuenta el celular de los encuestados.

- Frecuencia con la que ocupan el Pumabús

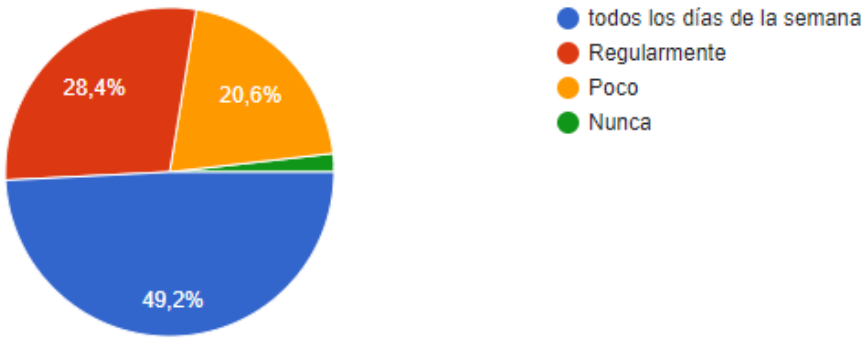


Figura 1.5 porcentaje de uso por semana del Pumabus.

- Ruta que frecuentan del Pumabús.

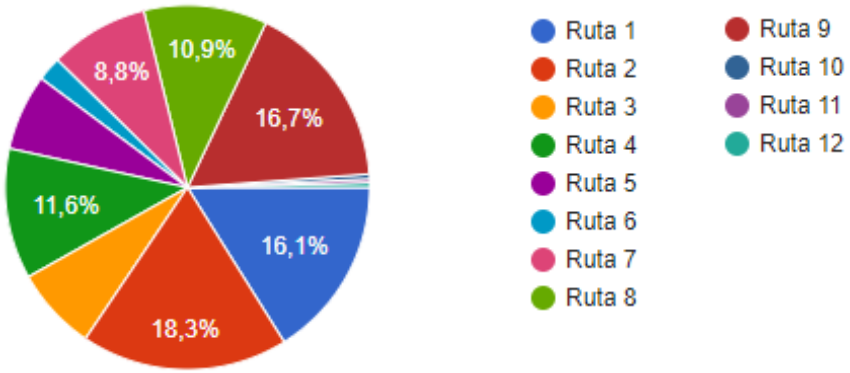


Figura 1.6 Porcentaje por ruta de su uso.

- Tiempo aproximado en que tarda el Pumabús.

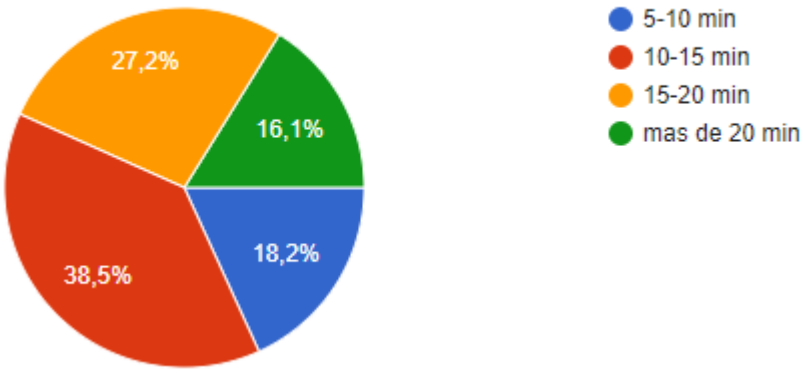


Figura 1.7 Porcentaje por tiempo que tarda en pasar el Pumabús.

- Alternativas de transporte utilizadas.

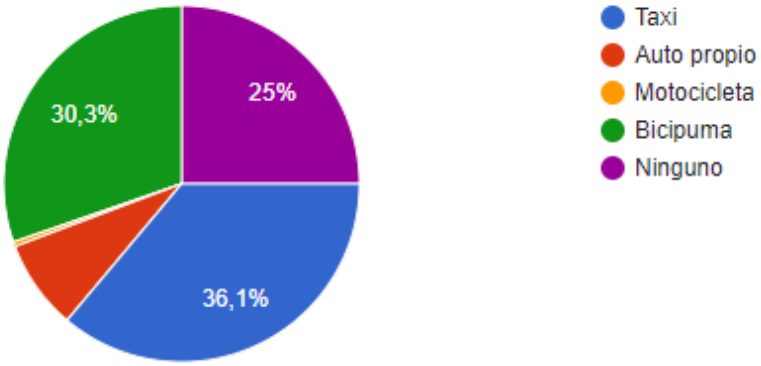


Figura 1.8 Porcentaje por opciones de traslado.

1.2.3. Definición del problema

Uno de los grandes problemas a los que nos enfrentamos como comunidad estudiantil y académica es la movilidad dentro de Ciudad Universitaria. Dentro de dicho campus se cuenta con diversos medios de transporte que nos proporciona la universidad, por un lado se encuentran los Pumabus; dicho servicio permite a estudiantes, académicos y personas ajenas a la Universidad llegar a su destino, sin embargo, hay ocasiones en que el tiempo de espera es demasiado o las unidades destinadas a una cierta ruta no se dan abasto con la gran población dentro de Ciudad Universitaria causando un retraso a las personas, el cual puede ser perjudicial obligando a tomar alguna otra alternativa.

Otro de los medios de transporte proporcionados por la Universidad son las bicipumas; las cuales son destinadas para alumnos y personal que labora dentro de las instalaciones. El problema que se presenta para este servicio es que en ciertos lapsos del día hay estaciones donde no se cuenta con alguna unidad (bicicleta), provocando que las personas tengan que ir a pie a su destino, generando un mayor tiempo de traslado.

Los medios de transporte mencionados anteriormente son proporcionados por la universidad de manera gratuita. Por el contrario, existen taxis dentro del campus universitario. Cuando los estudiantes, personal académico y personas ajenas no pueden hacer uso de los Pumabus y las bicipumas a causa de los inconvenientes mencionados anteriormente, una opción es utilizar este medio de transporte que involucra pagar una tarifa, la cual varía dependiendo de la unidad abordada. Si bien, es una opción viable para poder llegar a tiempo al destino de una persona no sabemos si es seguro abordar alguna unidad, ya que no hay ninguna entidad que regule este medio de transporte poniendo en peligro la integridad de cada una de las personas. Adicionalmente, si tomamos en cuenta que la población ha ido incrementando drásticamente dentro de Ciudad Universitaria, este servicio contribuye a la alta densidad vehicular dentro de los circuitos universitarios impidiendo la circulación de manera fluida.

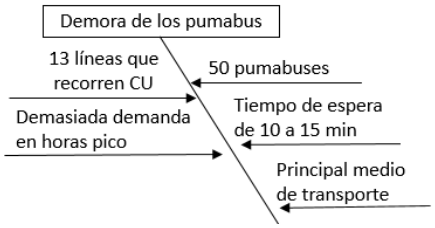
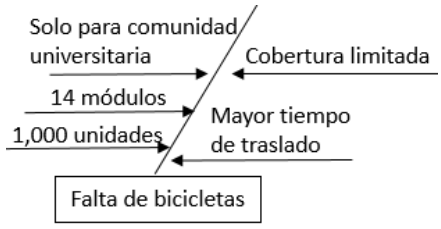
Finalmente, para los alumnos y profesores que no se ven necesitados a utilizar los medios de transporte mencionados, esto debido a que cuentan con un auto particular, la Universidad pone a disposición los diferentes estacionamientos distribuidos a lo largo del campus. El problema radica en saber el momento en que un estacionamiento ha llegado a su cupo máximo de vehículos, pues en ocasiones aunque ya no existan lugares disponibles en un cierto estacionamiento, los conductores siguen llegando a ese estacionamiento ya que no se les ha informado que no podrán ingresar, provocando tráfico y perdiendo su tiempo al buscar otro estacionamiento disponible.

Para los estudiantes y personas que laboran dentro de Ciudad Universitaria es importante el poder llegar a tiempo a su destino, sin embargo, hay factores a los que nos enfrentamos día con día y que no podemos controlar.

Para tener una visión clara de los problemas que se han mencionado anteriormente se empleara el uso de un diagrama de Ishikawa. El diagrama de Ishikawa fue desarrollado por el Dr. Kaoru Ishikawa y también es conocido como diagrama de causa-efecto. Este diagrama permite identificar las causas que dan origen a un problema, por cada causa identificada es posible agregar una subcausa o datos e información relevante que confirmen la causa.

Este diagrama puede ser utilizado en cualquier tipo de problema, además, puede ser adaptado por los usuarios dependiendo de las circunstancias a las que se estén presentando. La elaboración de este diagrama es fácil y sencillo, permite mantenerse enfocado al problema principal tomando en cuenta las causas para así establecer una solución más óptima y concisa.

En la Figura 1.9 se establece el problema "Dificultad de traslado dentro de Ciudad Universitaria" y se identifican las causas que dan origen al problema. Por cada causa también se proporcionan subcausas así como la representación de la causa en el diagrama de Ishikawa:

Problema	**Dificultad de traslado dentro de Ciudad Universitaria**	
Causa	**Subcausa o información**	**Representación**
Demora de las unidades del Pumabus.	• 50 Unidades del pumabus.[5] • 13 Líneas que recorren CU.[6] • Tiempo de espera de 10 a 15 min. • Demasiada demanda en horas pico. • Principal medio de transporte.	
Falta de bicicletas	• Solo para comunidad Universitaria. • Cobertura limitada. • 14 módulos de recepción.[7]	

[5] http://www.dgsgm.unam.mx/pumabus
[6] http://www.dgsgm.unam.mx/pumabus
[7] http://www.dgsgm.unam.mx/bicipuma

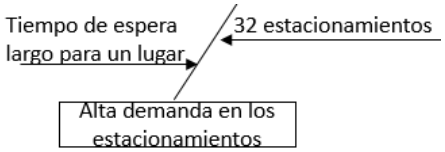
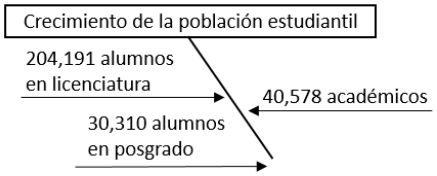
Problema	**Dificultad de traslado dentro de Ciudad Universitaria**	
Causa	**Subcausa o información**	**Representación**
	• 1000 unidades.[8] • Mayor tiempo de traslado.	
Costo elevado en las tarifas para taxis	• Falta de homologación en los precios. • No hay entidad regulatoria. • Escasa seguridad.	
Alta demanda de los estacionamientos	• 32 estacionamientos[9] • Tiempo de espera largo para obtener un lugar.	
Crecimiento de la población estudiantil	• 204,191 alumnos en licenciatura.[10] • 40,578 académicos.[11] • 30,310 alumnos en posgrado.[12]	

Figura 1.9 Causas para el diagrama de Ishikawa.

Una vez identificadas las causas del problema, es necesario unirlas para formar el diagrama de Ishikawa como se muestra en la Figura 1.10:

[8] http://www.dgsgm.unam.mx/bicipuma
[9] http://www.dgsgm.unam.mx/estacionamientos-controlados
[10] http://www.estadistica.unam.mx/numeralia/
[11] http://www.estadistica.unam.mx/numeralia/
[12] http://www.estadistica.unam.mx/numeralia/

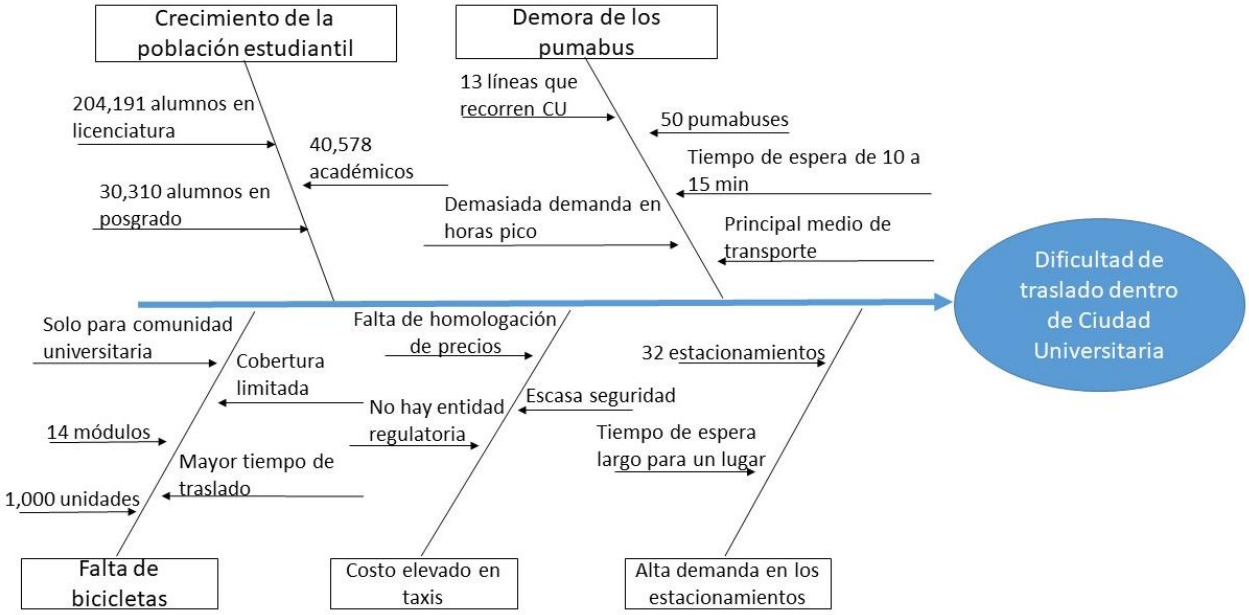


Figura 1.10 Diagrama de Ishikawa (Causa-Efecto)

Recapitulando un poco de lo que se ha hablado en esta sección, en seguida se en listaran los problemas con que se cuentan dentro de Ciudad Universitaria referente a la movilidad:

- El tiempo de espera de los usuarios para abordar el Pumabus es demasiado, retrasando al usuario a su punto de llegada.
- Falta de conocimiento de la ubicación de los Pumabus asignados a una ruta para tomar la decisión de esperar o tomar una alternativa.
- Cantidad de unidades del Pumabus insuficientes para dar abastecimiento a su demanda por parte de la comunidad estudiantil y externos.
- Falta de organización al distribuir adecuadamente las bicicletas en los diferentes puntos de préstamo, ya que la demanda de bicicletas en algunos puntos de distribución son mayores que en otros.
- Falta de conocimiento del tiempo restante para el uso de una bicicleta.
- No hay conocimiento de las estaciones de préstamo disponibles para las bicicletas.
- Dificultad para determinar que ruta del Pumabus pasa por el punto de llegada para un usuario.
- Dificultad para saber las estaciones cercanas a la ubicación de un usuario o cercanas a una facultad, instituto, museo, o dependencia de la Universidad.

- No se tiene una idea clara de la conexión entre las diferentes rutas del Pumabus.
- Cuando se aborda una unidad del Pumabus no se cuenta con el tiempo aproximado de llegada.
- La tarifa de los taxis no es constante.
- No se conoce la información de la unidad y del conductor para los taxis, incrementando la inseguridad dentro del campus.
- No existe una entidad regulatoria para los servicios de transporte externos al campus.
- Falta de conocimiento de los estacionamientos disponibles para estudiantes y personas externas a la universidad.
- No existe una herramienta que permita saber a los usuarios que estacionamientos cuentan con lugares disponibles, así como la ubicación del más cercano.
- Dificultad para conocer la mejor opción de transporte dentro de Ciudad Universitaria para llegar a un punto específico.

1.2.4. Revisión de soluciones existentes.

Hoy en día, uno de los dispositivos electrónicos con más auge en el mercado son los teléfonos celulares. Dichos dispositivos podemos encontrarlos con diferentes precios dependiendo de sus características y servicios que proporcionan a los usuarios. Gracias a esto la mayoría de la población cuenta con teléfonos celulares con la finalidad de poder satisfacer alguna necesidad. El mercado en esta área no se ha quedado atrás, pues la comercialización de los dispositivos está integrada con un conjunto de servicios como: entretenimiento, comida, transporte, cultura, etc.

En el área de transporte podemos encontrar diferentes aplicaciones que brindan servicios de movilidad con la finalidad de reducir el tiempo de traslado y aumentar la seguridad para el usuario. En México y especialmente en la Ciudad de México existen grandes empresas que brindan el servicio mencionado anteriormente: Uber, Cabify , Easy Taxi entre otras. Estas compañías son quienes dominan ampliamente el mercado nacional ya sea por la confianza que brindan a sus usuarios, así como el precio a comparación con otros servicios.

Es importante mencionar que estas empresas son extranjeras. Existen aplicaciones nacionales que tratan de competir en el mercado, sin embargo, no se han abierto paso con tanto éxito como las mencionadas anteriormente.

Ahora bien, en las distintas tiendas de aplicaciones móviles existe una gran cantidad de aplicaciones enfocadas a los diferentes sistemas de transporte dentro de la Ciudad de México. Para el caso de Ciudad Universitaria no pasa por desapercibido, ya que en dichas aplicaciones se toma en cuenta el principal transporte de este campus que es el “Pumabus”. Un claro

ejemplo es la aplicación llamada “Metro y Metrobús de México”[13] (Figura 1.11) la cual tiene un apartado para el transporte “Pumabús” en la cual podemos ver las diferentes paradas existentes y las rutas, todo esto puede ser observado en un listado o en un mapa para saber con mayor exactitud su localización, además, se permite la visualización de todas las rutas por medio de una imagen.

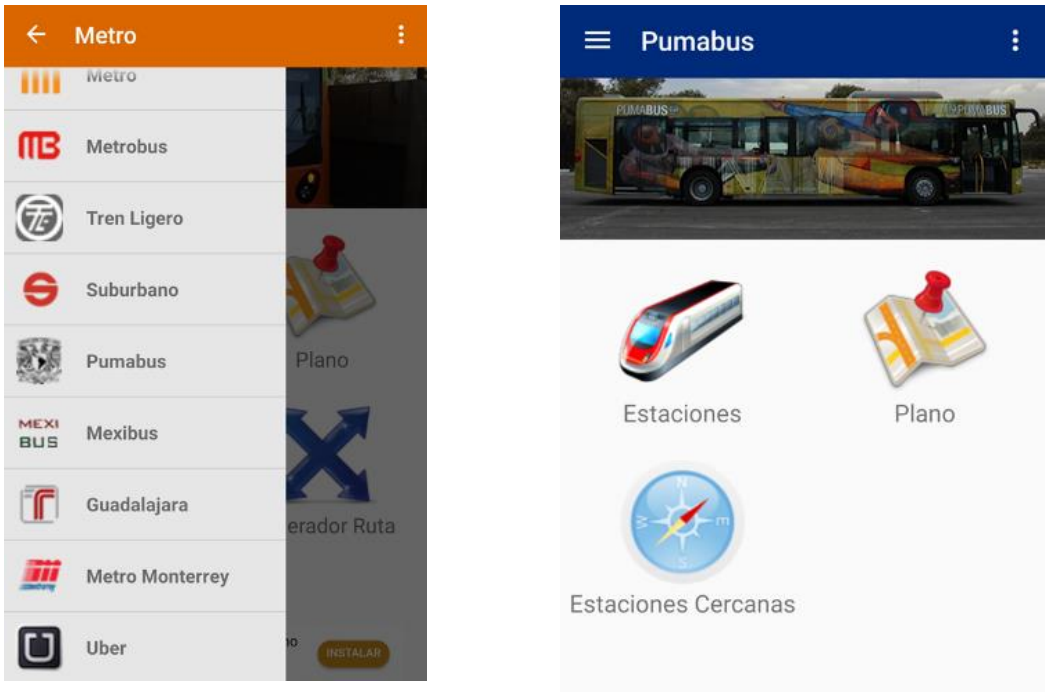


Figura 1.11 Imágenes de la aplicación “Metro y Metrobús de México”.

[13] Aplicación de la Play Store

1.3 Objetivos

1.3.1. Objetivos generales para la mejora de la movilidad en Ciudad Universitaria

- Crear una aplicación que facilite la movilidad dentro de Ciudad Universitaria.
- Reducción de tiempos de traslado en Ciudad Universitaria.
- Crear una entidad que regule los medios de transporte dentro de Ciudad Universitaria.
- Crear un sistema que permita la administración de los distintos medios de transporte en Ciudad Universitaria.
- Homologar rutas, estaciones del Pumabús, bicipuma y el servicio de taxis.
- Aumentar la seguridad en los medios de transporte de Ciudad Universitaria.

1.3.2. Objetivos específicos del sistema de información en tiempo real del transporte interno en el campus de Ciudad Universitaria.

- Uso del lenguaje de programación JAVA y Spring Framework.
- Uso de herramientas web para la creación de una aplicación hibrida.
- Implementación de una Base de datos con extensión espacial para el manejo de información geográfica de manera rápida y eficaz.
- Implementación de una Base de datos que hace uso de los llamados “Servicios en la Nube” (Firebase) cuya principal característica es el manejo de información en tiempo real.
- Manejo de una metodología para la creación del proyecto.

2. MARCO TEÓRICO

2.1 Principales tecnologías a utilizar

2.1.1. Ionic Framework

Ionic es un framework enfocado al desarrollo de aplicaciones híbridas con base en tecnologías web como pueden ser: HTML, JavaScript y CSS. Una de las principales ventajas que nos ofrece Ionic es su fácil implementación en diferentes sistemas operativos. En este caso como se trata de aplicaciones móviles estamos hablando de Android, IOS y Windows Phone. Estas tres plataformas mencionadas anteriormente son las que hoy en día dominan el mercado.

Una parte fundamental para la creación correcta de aplicaciones móviles con dicho framework es la herramienta "Cordova", quien es el encargado de brindar el soporte de las herramientas web para la construcción, visualización y apto funcionamiento de las aplicaciones y así evitar la programación del lenguaje nativo (para Android java y para IOS tanto Swift como Objective C).

Una de las limitantes de Ionic es que al no utilizar el lenguaje de programación nativo correspondiente, el acceso a los componentes de hardware (cámara, teclado, brújula, etc.) se vuelve más complejo, sin embargo, Cordova ofrece una serie de plugings para estos componentes. Dichos plugings han sido actualizados de tal forma que su implementación es idéntica a la utilización de los componentes nativos.

Así mismo, Cordova permite crear las aplicaciones para las diferentes plataformas; la aplicación se desarrolla con Ionic (herramientas web) y posteriormente se realiza una compilación de dicho código para la creación de las aplicaciones en las diferentes plataformas respetando los patrones de diseño: Material Design para Android y los patrones establecidos para IOS.

Teniendo en cuenta lo anterior, después de haber realizado la compilación de nuestra aplicación móvil se hace uso de los IDE`s (Entorno de Desarrollo Integrado) Android Studio y Excode para Android y IOS respectivamente, ya que ahora se cuenta con aplicaciones destinadas a un sistema operativo. Con estos IDE`s se logra el despliegue de las aplicaciones en dispositivos físicos.

2.1.2. Android Studio

Android Studio es el entorno de desarrollo integrado (IDE) oficial para el desarrollo de aplicaciones móviles en la plataforma Android. Dicho software fue proporcionado por Google. Android Studio nos permite cubrir una gran gama de dispositivos tecnológicos en los cuales se puedan instalar aplicaciones, pues además de poder realizar aplicaciones para celulares y tabletas también nos proporciona la capacidad de crear aplicaciones para relojes inteligentes y TV`s.

El lenguaje de programación para aplicaciones móviles en Android es Java. Por esta razón, es necesario tener instalado el JDK de Java para el funcionamiento adecuado del IDE. Por otro lado, existe un elemento de suma importancia llamado SDK (Software Development Kit), que es un grupo de herramientas empleadas para la programación de aplicaciones móviles. El SDK de Android tiene las herramientas necesarias para las diferentes versiones del sistema operativo Android. Cuando se realiza una aplicación en este IDE es necesario indicar en que rango de versiones del sistema operativo que la aplicación soportará. Si se presentara el caso en que una aplicación se intentara desplegar en una versión de Android fuera de rango la aplicación, esta podría no funcionar correctamente.

Cuando se realiza una aplicación con el IDE oficial se tiene acceso a un mayor número de servicios, por ejemplo: es posible sincronizar Android Studio con servicios externos como puede ser GitHub, Kotlin, Firebase, etc. Permitiendo la robustez de las aplicaciones móviles.

A lo largo del desarrollo de una aplicación móvil, la ejecución de pruebas en especial para verificar el correcto funcionamiento y apariencia de los elementos gráficos es fundamental. Android Studio ofrece dos opciones: instalar un dispositivo virtual o el uso de un dispositivo físico.

Para el caso de la primera opción el dispositivo virtual emulará un teléfono físico, esto se logra gracias al SDK. Con este dispositivo virtual se realiza el despliegue de la aplicación móvil permitiéndonos interactuar con su funcionamiento. Por otro lado, una opción real es la instalación de la aplicación en algún dispositivo móvil físico. Para esto, solo se necesita un cable USB, Android Studio está disponible para las diferentes plataformas (Windows, MAC Y Linux). Para el caso de Windows si se desea instalar la aplicación en un dispositivo real es necesario instalar un controlador (podrá ser encontrado en las páginas oficiales de la compañía que fabrico el celular), por lo contrario, para Mac y Linux no se necesita de ningún controlador.

Cuando se realiza una instalación de cualquier aplicación lo que se descarga de la tienda oficial es el APK (Android Application Package). El APK es el medio por el cual se distribuyen e instalan las aplicaciones para Android. Android Studio tiene la capacidad de generar el APK de cualquier aplicación siempre y cuando el proceso de compilación no haya generado algún error.

2.1.3. Spring

Spring es un framework escrito en Java que permite el desarrollo de aplicaciones empresariales. Una característica principal de este framework es que proporciona soporte para las diferentes arquitecturas de las aplicaciones: mensajería, datos transaccionales, persistencia y web.

En el libro “Spring in Action” se menciona que el framework Spring emplea cuatro estrategias:

- Desarrollo ligero y uso de POJOs.
- Uso de la inyección de dependencias y orientación de interfaz.
- Programación declarativa
- Eliminación de código repetitivo.

En los puntos anteriores se mencionaron dos conceptos importantes para el desarrollo de aplicaciones con el framework Spring:

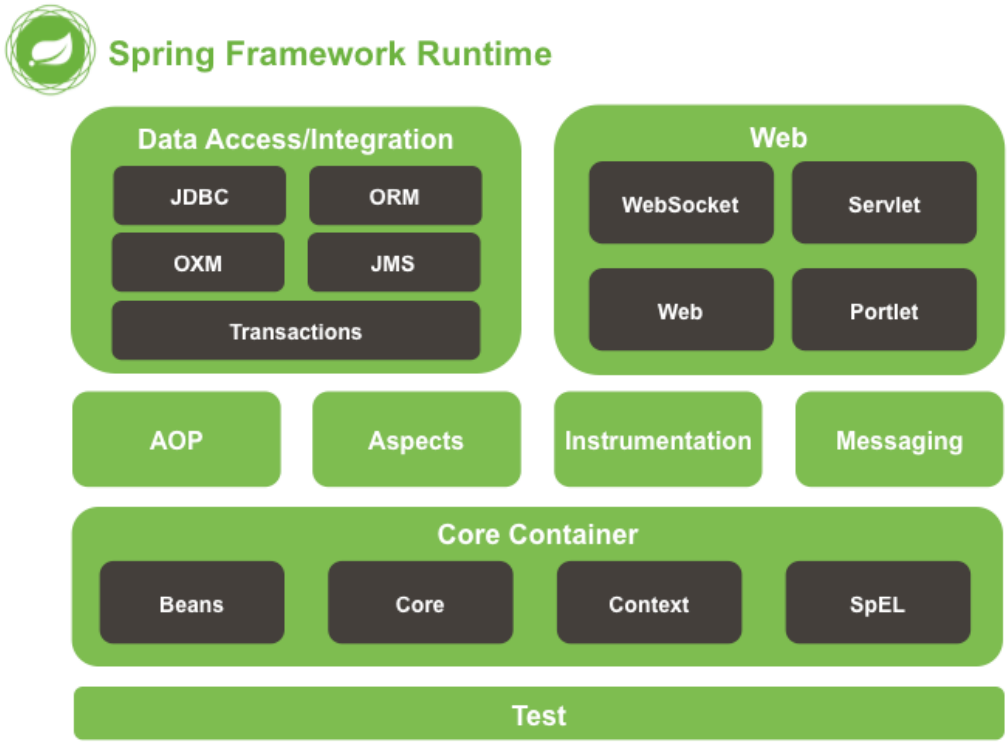
Los POJOs (Plain Old Java Object). Son instancias de clases que pueden contener la definición atributos o métodos definidos en dicha clase. Los POJOs no deberían de extender de otra clase o implementar alguna interfaz.

La inyección de dependencias está relacionada con el manejo de los objetos que típicamente ofrecen algún tipo de servicio o funcionalidad específica. Estos objetos cuentan o requieren de ciertas dependencias necesarias para realizar su función. El framework se encarga de instanciar estas dependencias y proporcionárselas o ‘inyectarlas’ al objeto que las requiera. Ejemplos de dependencias: servicios de negocio utilizados en el sistema, objetos que permiten realizar la conexión a la base de datos, etc.

Al implementar la inyección de dependencias el código se vuelve simple, fácil de entender y fácil de probar. La función principal de la Inyección de dependencias es crear un código desacoplado para que este sea reusable y este más modulado. Gracias a lo mencionado anteriormente es posible realizar pruebas unitarias de una manera más fácil ya que el

programa puede ser dividido en componentes y a cada uno realizar pruebas independientes de los demás componentes.

Spring framework asegura la estandarización y permite el desarrollo de aplicaciones de manera rápida y eficaz. Spring está formado por una serie de módulos, dependiendo de las necesidades, así como requerimientos de la aplicación a desarrollar se podrá elegir los módulos necesarios para cumplir con dichos requisitos.



Anexo 2.1 Componentes de Spring Framework

- Core Container
  Este módulo está formado por: Spring-core, Spring-beans, Spring-context y Spring-expression.
  Tanto Spring-core y Spring-beans brindan aspectos fundamentales y básicos para el framework ya que incluyen el patrón de Inversión de control (IoC) y la Inyección de dependencias (Dependency Injection).

- AOP
  La Programación Orientada a Aspectos (AOP) es un paradigma de programación que permite aumentar o mejorar la funcionalidad de un componente a través del concepto de “aspectos”. Un aspecto puede representar una funcionalidad que no

necesariamente ha sido implementada directamente en el código del componente. Dicha funcionalidad puede ser “anexada” o “agregada” al componente para mejorar o robustecer su funcionalidad sin la necesidad de modificar una sola línea de código del componente original.

Un ejemplo muy representativo de la programación orientada a aspectos dentro del framework de Spring es el manejo transaccional. El programador realiza la construcción de servicios de negocio (componentes del sistema) sin la necesidad de preocuparse por los requerimientos en cuanto al manejo transaccional que debe tener su servicio al interactuar con la base de datos (niveles de aislamiento, concurrencia, etc.). AOP permite agregar o adicionar estos requerimientos sin la necesidad de modificar el código del componente.

- Messaging
  Módulo enfocado a aplicaciones basadas en mensajería con ayuda de elementos como Message, MessageChanel y MessageHandler, además, incluye un conjunto de anotaciones para métodos.

- Data Access/Integration
  Este módulo está conformado por submódulos:

  Spring-jdbc: Permite realizar la configuración necesaria para establecer la conexión a la base de datos de algún manejador específico.

  Spring-tx: permite el manejo de transacciones.

  Spring-orm: proporciona la integración de diferentes API´s, incluyen JPA, JDO e Hibernate.

  Spring- oxm: permite la asignación de objetos XML.

- Web
  Este módulo proporciona la funcionalidad y conceptos básicos de una aplicación web. Contiene un cliente HTTP.

  Por otro lado, este módulo contiene el modelo-vista-controlador (MVC) así como la implementación de los servicios web REST.

- Testing
  Permite la realización de pruebas unitarias, así como pruebas de integración.

Una de las principales características que se ha implementado en Spring es la Inyección de dependencias. Para poder entender dicho concepto es necesario tener una idea de lo que es el contenedor de Spring: el contenedor de Spring es el encargado de la creación de objetos (que permitan la conexión con otros servicios fuera del sistema como puede ser la base de datos), permite establecer una conexión entre dichos objetos de la aplicación y controla el ciclo de vida de los objetos por medio de la inyección de dependencias (Dependency Injection).

Actualmente Spring ha dado a conocer una nueva versión llamada Spring Boot. Como podemos darnos cuenta, Spring integra una gran variedad de módulos, los cuales si son necesarios para la aplicación deben de ser configurados cada uno de ellos, en cambio, Spring Boot está diseñado para crear aplicaciones lo más rápido posible y con una configuración mínima y automática.

Un aspecto que facilita el desarrollo de aplicaciones en Spring boot es que esta herramienta se encarga de resolver la incorporación de librerías/dependencias que sean necesarias para que la aplicación funcione de manera correcta. Es posible ejecutar aplicaciones web mediante un servidor web integrado.

2.1.4. Web Service

Un Web Service o también conocido como servicio web, es un servicio ofrecido por una aplicación, con la finalidad de poder intercambiar información (datos) por medio de protocolos a distintas aplicaciones o sistemas. Dichas aplicaciones pueden estar desarrolladas en diferentes lenguajes de programación y su comunicación se realiza por medio de una red de computadoras: Internet.

Por otro lado, es importante mencionar que un web service no provee a los usuarios una interfaz gráfica para su interacción, ya que la finalidad es compartir la lógica de negocio, datos, así como ciertos procesos por medio de una red.

Típicamente un web service es desarrollado con un conjunto de tecnologías:

- XML
  Extensible Markup Language (XML) es un formato que es utilizado para datos estructurados. Su extensión para estos archivos es .xml y ocupa las etiquetas “<” y “>” para realizar la estructuración de los documentos.

- SOAP
  Simple Object Access Protocol es un protocolo enfocado a la mensajería y que está basado en xml. Los archivos XML contienen la configuración y definición de reglas que permiten codificar las respuestas a las peticiones provenientes de otros sistemas. Estos mensajes son procesados por medio de protocolos de internet como: SMTP, HTTP, etc.

- WSDL
  Web Services Description Lenguage, lenguaje basado en XML que permite la definición de un web service con la finalidad de intercambiar información por medio de mensajes.

- UDDI
  Universal Description, Discovery and Integration, es un medio (directorio) donde se puede publicar y buscar información acerca de los web service registrados por las empresas.

2.1.5. Firebase – Realtime Data Base

Firebase es una plataforma conformada por un conjunto de herramientas orientadas al desarrollo móvil con la finalidad de brindar servicios que permitan la creación de aplicaciones robustas. Firebase fue creado por Google y sus servicios son alojados en la nube.

Una de las principales características de Firebase es que sus servicios pueden ser utilizados en las diferentes plataformas. Hoy en día al hablar acerca de la implementación de sistemas tecnológicos se deben de considerar dos brechas de suma importancia: aplicaciones móviles o sistemas web, la ventaja de Firebase es que sus servicios pueden ser utilizados para ambos casos: IOS, Android o WEB.

A continuación, se presentan los servicios proporcionados por Firebase:

Por un lado, podemos encontrar un conjunto de herramientas orientadas al desarrollo y testing de las aplicaciones móviles:

- Realtime Database: Almacena y sincroniza datos entre la base de datos almacenada en la nube y los dispositivos conectados en milisegundos.

- Crashlytics: Permite la administración de errores a los que se enfrentan los usuarios en tiempo real.
- Cloud Firestore: Permite la sincronización de datos entre usuarios y dispositivos almacenados en la nube.
- Authentication: Provee diferentes métodos para la autentificación de usuarios en la aplicación.
- Cloud Functions: Creación de funciones para mejorar el rendimiento de la aplicación.
- Cloud Storage: Permite almacenar y compartir recursos de multimedia.
- Hosting: Brinda el alojamiento para sistemas web de manera confiable y segura.
- Test Lab: Permite la ejecución de pruebas de la aplicación en dispositivos virtuales y físicos (banco de dispositivos con diferentes características). Las pruebas se basan en métodos creados por Google que ponen a prueba la aplicación.
- Performance Monitoring: Posibilita el manejo de problemas presentados en la aplicación de cada uno de los usuarios.

Y finalmente se encuentra el conjunto de herramientas orientadas al crecimiento de la audiencia en la aplicación:

- Google Analytics: Analiza el comportamiento de los usuarios y provee estadísticas.
- Cloud Messaging: Permite el envió de mensajes y notificaciones a los dispositivos que contengan la aplicación.
- Predictions: Crea grupo de usuarios con base a los analizados por medio de aprendizaje automático.
- Dinamics Links: Creación de vínculos directos para mejorar la experiencia de usuario.
- Remote Config: Permite la configuración de contenido para ciertos usuarios, así como realizar una actualización sin implementar una versión nueva.
- Invites: Brinda la facilidad de compartir la aplicación por medio se correo electrónico o SMS.
- App Indexing: Involucra la aplicación y su contenido en las búsquedas de Google, aumentando el índice de instalación.
- AdMob: Brinda la posibilidad de incorporar publicidad dentro de la aplicación.
- AdWords

Realtime- Database: se trata de una base de datos NoSQL que almacena y gestiona datos en la nube. Los procesos que involucran a los datos son manejados en tiempo real permitiendo un acceso a la información de manera inmediata.

Los datos que son almacenados son tratados como objetos JSON, de esta manera podemos referenciar a la base de datos como un árbol JSON en donde cada vez que se almacena un dato se está creando un nodo a este árbol con la estructura ya mencionada.

Cuando un dispositivo se encuentra sin conexión a internet los datos que hayan sufrido alguna alteración son almacenados en la memoria cache del dispositivo. Una vez restablecida la conexión, se enviarán los cambios a los servidores donde se encuentra almacenada la base de datos. Por otro lado, si en un nodo específico del árbol se produce alguna modificación, este dato será enviado a todos los dispositivos que estén conectados a una instancia para que su valor sea actualizado.

Como podemos darnos cuenta, la sincronización de los datos que maneja Realtime-Database está organizada permitiendo la reducción de costo en procesamiento, si se compara con una base de datos SQL, por cada determinado tiempo se tendría que realizar consultas a los datos, provocando el consumo de recursos.

2.1.6. PostgreSQL y PostGIS

PostgreSQL es un sistema para base de datos de tipo relacional que soporta la mayor parte del lenguaje SQL estándar. PostgreSQL está disponible para los sistemas operativos más populares en el mercado (Windows o cualquier distribución de Linux).

Como se ha mencionado anteriormente, PostgreSQL es un sistema para bases de datos, permitiéndonos su administración que incluye, creación de tablas, procedimientos, triggers, secuencias, entre otros elementos disponibles. Además, brinda una gran cantidad de tipo de datos para el almacenaje de la información. Si bien, PostgreSQL nos brinda estos servicios, para poder administrar una base de datos se puede hacer por medio de la consola o emplear una herramienta gráfica, pgAdmin (Figura 2.2) es la herramienta oficial para la administración de BD en PostgreSQL, nos permite desarrollar una base de datos completa de manera sencilla y fácil, usando su interfaz gráfica.

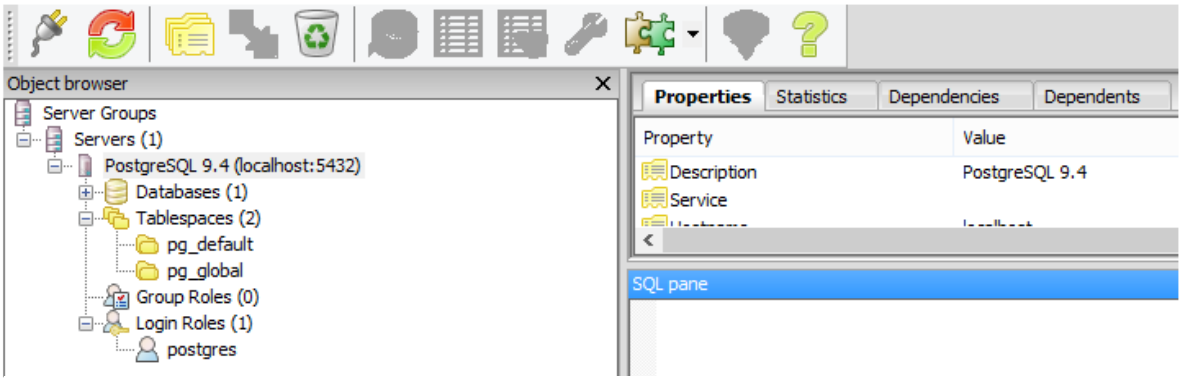


Figura 2.2 Entorno gráfico para administrar una BD PostgreSQL.

PostGIS es una extensión espacial para PostgreSQL. PostGIS proporciona funcionalidades para el manejo, almacenamiento, gestión y mantenimiento de datos espaciales como puntos, líneas, polígonos, entre otros.

Algunas de las características de PostGIS son:

- Software libre y compatible con Open Geospatial Consortium(OGC).
- Multiusuario.
- Brinda un conjunto de funciones espaciales para facilitar las consultas.
- Proporciona diferentes tipos de geometrías.
- Las consultas se ejecutan en poco tiempo.

Además, pgAdmin ha incorporado en sus últimas versiones un visualizador de geometrías para las consultas como se puede observar en la Figura 2.3

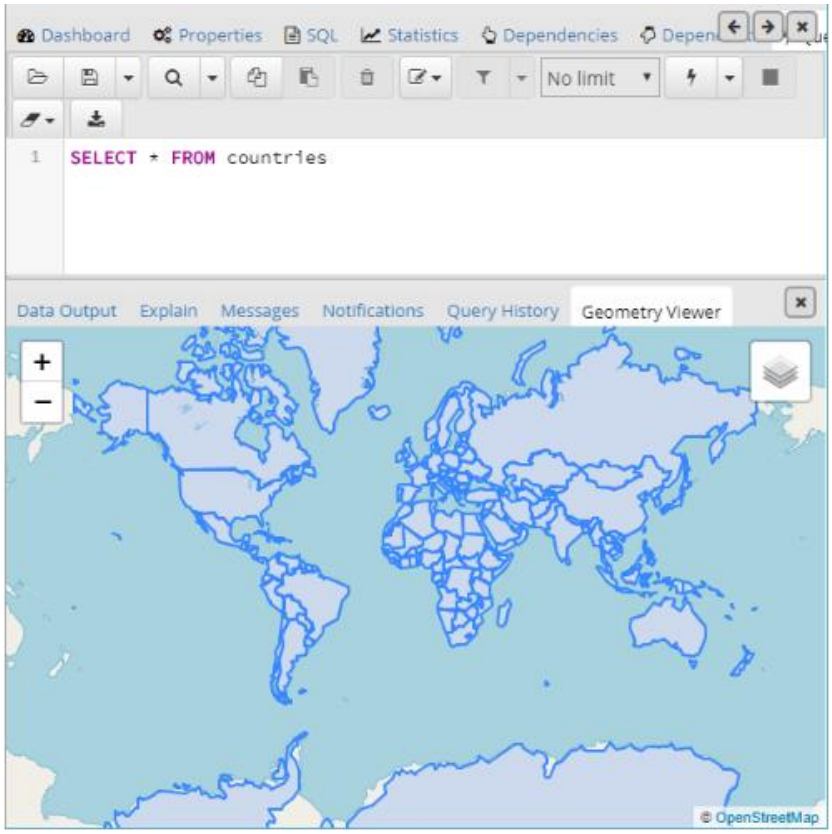


Figura 2.3 Ejemplo de una consulta utilizando PostGIS.

2.2 Metodologías en el desarrollo de software

Un aspecto importante que se debe de tomar en cuenta antes de empezar con el desarrollo de software es la metodología que se implementará. Una metodología en el desarrollo de software está integrada por procedimientos, técnicas, documentos que permiten a los miembros del equipo de desarrollo elaborar sistemas, en este caso, sistemas de software. Las metodologías permiten planificar, gestionar, controlar y evaluar el proceso de desarrollo de software.

Las metodologías tienen como objetivo:

- Satisfacer las necesidades del usuario.
- Cumplir con los requisitos en tiempo y forma.
- Ajustarse a los plazos y costos planificados.
- Facilitar el mantenimiento del sistema.
- Definir la secuencia de actividades durante el desarrollo del software.
- Busca la uniformidad y calidad del sistema.

- Tener documentación que sustente al sistema.

En los últimos años, las metodologías se han clasificado en dos grupos principales: las metodologías tradicionales, las cuales en su proceso se definen roles, actividades, hacen uso de modelado y documentación detallada, y por otro lado se encuentran las metodologías ágiles que buscan una adaptación a los cambios, hacen énfasis en las habilidades del equipo así como la importancia de mantener una buena relación con el usuario.

2.2.1. Metodologías tradicionales

Al inicio del desarrollo de software no se contaba con procesos definidos a seguir para cumplir los objetivos, a causa de esto surgió la necesidad de mejorar el proceso y así poder cumplir con las metas deseadas. Estas nuevas metodologías se basaban en etapas secuenciales que mejoraban el proceso de desarrollo de software.

Las metodologías tradicionales hacen énfasis en la elaboración de documentación exhaustiva y detallada de todo el proyecto, así como el cumplimiento del plan de proyecto definido en la fase inicial de desarrollo de software.

Entre las principales metodologías tradicionales se presenta RUP (Rational Unified Process), MSF (Microsoft Solution Framework) y Modelo espiral.

- RUP

RUP se basa en la asignación de tareas y responsabilidades asignadas a los miembros de una organización. Su principal objetivo es desarrollar software de alta calidad cumpliendo con los requerimientos del cliente, tomando en cuenta el tiempo estimado en el cronograma y el presupuesto.

Los elementos principales de esta metodología son los casos de uso, definición de la arquitectura y utilizando los diagramas UML. Dentro de esta metodología se definen cuatro fases:

  - Concepción.
  - Elaboración.
  - Construcción.
  - Transición.

- MSF

Más que una metodología, MSF está definida como una serie de modelos que pueden adaptarse a cualquier tipo de proyecto de software. Está formado por cinco fases primordiales:

  - Visión y alcances: Se identifican las metas, se definen los roles dentro del equipo, permite tener un panorama claro del proyecto.

  - Planificación: Se realiza el diseño del sistema, se estiman costos, realización de cronogramas.

  - Desarrollo: Realización de componentes basados en la documentación obtenida en los puntos anteriores.

  - Estabilización: Realización de pruebas sobre el sistema, en caso de que ocurra algo fuera de las funcionalidades se debe de corregir.

  - Implantación: Se realiza la instalación del sistema, se pone al tanto del funcionamiento al área de soporte de los clientes.

- Modelo espiral

El modelo espiral está basado en la realización de prototipos y fases secuenciales. Una de las principales características de este modelo es que al presentar la necesidad de un cambio o mejora se analiza las alternativas lo cual provoca recorrer las fases de esta modelo hasta que el producto quede terminado.

2.2.2. Metodologías ágiles

Las metodologías ágiles surgieron a causa de que el entorno de los proyectos cambiaba frecuentemente y se tenía la exigencia de reducir los tiempos de desarrollo sin aminorar la alta calidad de los productos. Las principales ideas de las metodologías ágiles son:

- Principal valor a los individuos, así como la interacción entre ellos.
- Mayor importancia en la creación de un producto que funcione de manera correcta a tener una documentación exhaustiva.
- La colaboración con el cliente e interacción con el equipo de desarrollo.
- Capacidad de respuesta a los cambios.

Dentro de las metodologías ágiles podemos encontrar XP (Extreme Programming) y SCRUM.

- XP.

La metodología XP o programación extrema se enfoca en las relaciones y capacidades intrapersonales de los miembros del equipo de desarrollo, así como fortalecer el trabajo en equipo tomando en cuentas aspectos como el aprendizaje de cada miembro hasta proporcionar un buen ambiente de trabajo.

Algunas de las características de esta metodología es la retroalimentación entre el cliente y el equipo de desarrollo, comunicación fluida entre los miembros, aceptación de los cambios, etc.

A continuación, se mencionan algunos aspectos tomados en cuenta en la programación extrema:

  - Desarrollo iterativo e incremental.
  - Pruebas unitarias establecidas antes del desarrollo.
  - Programación por parejas.
  - Interacción frecuente.
  - Refactorización.

- SCRUM.

SCRUM está enfocada a proyectos que presentan un rápido cambio de requisitos. Por esta razón el desarrollo de software se realiza basado en iteraciones llamadas sprints, con una duración entre 15 días a 30 días. Cada sprint es un resultado que deberá ser entregado al cliente, con funcionalidad y calidad.

Otra de las principales características dentro de SCRUM es la comunicación o transparencia dentro del equipo de desarrollo, a lo largo del desarrollo de software se tienen diversas reuniones, por ejemplo, los miembros del equipo deberán reunirse diario con una duración aproximadamente de 15 minutos en la cual cada miembro explicará las actividades que ha realizado, las que realizará y si se ha enfrentado a un inconveniente que le prohíba alcanzar los objetivos planteados en el sprint.

SCRUM implementa una herramienta llamada SCRUM Board que permite una mejor organización tanto para cada miembro, así como para el equipo en conjunto. En el SCRUM Board se suelen colocar las siguientes columnas:

- To do: Actividades por realizar.
- In progress: Actividades que se están realizando por algún miembro del equipo.
- Testing: Actividades terminadas y listas para el ciclo de pruebas.
- Done: Actividades que están listas para formar parte de un entregable ya que están validadas por el equipo de pruebas.

Figura 2.4 SCRUM board

3. ANÁLISIS DE REQUERIMIENTOS.

3.1 Selección de la metodología de software

Una de las controversias que existen en el desarrollo de software es la elección de la metodología, si bien, no existe una metodología universal para garantizar el éxito del desarrollo de software, una solución es adaptar la metodología al contexto del proyecto.

Para la elección correcta de la metodología se puede tomar en cuenta aspectos como el enfoque del proyecto, duración, planificación, presupuesto, cualidades de los integrantes del equipo de desarrollo, etc. A continuación (Figura 3.1) se presentan diferencias entre las metodologías tradicionales y ágiles.

Metodologías tradicionales	Metodologías ágiles
Se basan en estándares establecidas por el entorno de desarrollo.	Se basan en heurísticas.
Cierta resistencia a los cambios	Aceptación a los cambios.
Proceso muy controlado	Proceso poco controlado.
Interacción con el cliente por medio de reuniones	El cliente es parte del grupo de desarrollo
Soporta grupos de trabajo grandes	Preferible para grupos pequeños(menos de 10 personas)
Importancia en la documentación	Importancia en los entregables.

Figura 3.1 Diferencias entre metodologías tradicionales y ágiles

Para el presente proyecto se ha decidido utilizar una combinación entre componentes de las metodologías tradicionales como de las metodologías ágiles, esto con la finalidad de poder cumplir con los objetivos planteados.

Por parte de las metodologías tradicionales se tomarán componentes de la documentación con el objetivo de tener respaldado la construcción del sistema, se harán uso de elementos como:

- Caso de uso.
- Casos de uso detallado.
- Diagrama de actividades.
- Diagramas para el diseño de arquitectura.
- Diagramas de secuencia.
- Entre otros.

Por parte de las metodologías ágiles es importante tomar en cuenta que el sistema podría presentar cambios durante la fase de desarrollo y se debe estar preparado para actuar de la manera correcta, por otro lado, el avance en el desarrollo dependerá del conocimiento que se tenga sobre las herramientas a utilizar, así que las capacidades intrapersonales de los miembros será de suma importancia.

Se podrán hacer uso de elementos como:

- Alta comunicación entre los miembros del equipo.
- Programación por parejas.
- Refactorización.
- Organización de las actividades.
- Liberaciones constantes, ciclos de poca duración.

3.2 ¿Qué es un requerimiento?

Un requerimiento da a conocer cómo debe funcionar un sistema y sus características principales. Generalmente, los requerimientos son definidos por el usuario final y serán transmitidos al equipo de desarrollo. Es importante mencionar que los requerimientos especifican las acciones que un sistema debe realizar dejando a un lado el cómo lo realizará.
La correcta interpretación e implementación de los requerimientos constituyen la base de un sistema exitoso, tanto para el usuario final como para el equipo de desarrollo. Los requerimientos deben ser claros, entendibles, concisos, específicos libres de cualquier tipo de ambigüedad. Esto ayudará a la reducción de costos y riesgo en el desarrollo.

Como se mencionó anteriormente, para la obtención de requerimientos se necesita principalmente de dos actores: los usuarios finales y el equipo de desarrollo. Los usuarios finales son los autores de las reglas de negocio, saben el funcionamiento y el comportamiento del sistema. En ocasiones, el usuario puede tener una idea confusa, y es ahí donde entra el segundo actor, el equipo de desarrollo: encargado de entender las ideas de usuario, procesarlas y crear ideas más concisas y claras. El equipo de desarrollo actuará como entrevistador para la obtención de requerimientos.

La comunicación entre estos dos actores es de suma importancia, son ellos quienes pueden evitar malas interpretaciones, falta de información o ambigüedad en los requerimientos. Es común que los usuarios y los miembros del equipo de desarrollo acudan a juntas con la finalidad de aclarar dudas o agregar nuevos requerimientos del sistema.

El análisis de requerimientos se puede dividir en dos tipos: requerimientos funcionales y no funcionales.

3.3 Requerimientos funcionales.

Los requerimientos funcionales son aquellos que describen lo que el sistema debe de hacer, como debe de reaccionar al recibir entradas específicas, cómo comportarse al estar en interacción con usuarios, los servicios que proveerá, etc. En ocasiones los requerimientos funcionales también especifican lo que el sistema no debe de hacer.

Algunos ejemplos de requerimientos funcionales para el "Sistema integral aplicado a la mejora de la movilidad en Ciudad Universitaria" se en listan a continuación:

- Aplicación móvil usuarios.
    - Los usuarios podrán iniciar sesión proporcionando su usuario y contraseña.
    - En dado caso que no se cuente con un usuario y contraseña, los usuarios podrán crear su cuenta desde la aplicación.
    - La aplicación permitirá a los usuarios consultar en un mapa las diferentes rutas de los Pumabús, así como la ubicación de los camiones asociadas a una ruta y las estaciones donde el Pumabús se detendrá para el abordaje y descenso de pasajeros.
    - La aplicación permitirá a los usuarios consultar en un mapa la ciclopista y las terminales de préstamo y devolución de las bicicletas, además, se podrá consultar las unidades disponibles por cada terminal.
    - Los usuarios solo podrán consultar la información, no se podrá realizar la edición de los datos proporcionados.
- Aplicación móvil conductores
    - Aplicación móvil orientada para conductores de Pumabús o taxis.
    - La aplicación restringirá el acceso solo a los usuarios que se encuentren registrados en el sistema y que funjan un rol de conductor.

- Los conductores registrados estarán asociados a una sola unidad, ya sea Pumabús o taxi.

- La aplicación será utilizada solo en horario de trabajo para los conductores.

- La aplicación permite al usuario el cierre de sesión.

- Sistema Movilidad UNAM

    - El sistema restringirá el acceso solo a los usuarios que se encuentren registrados y con permisos para acceder al sistema.

    - El sistema tendrá los siguientes módulos: Pumabús, Estacionamientos, Estaciones, Rutas y Usuarios.

    - Para el módulo de Pumabús, el sistema permitirá el registro de una nueva unidad.

    - Para el módulo de Estaciones, se permitirá a los usuarios el registro de una nueva estación, se solicitará colocar un marcador en un mapa para la ubicación exacta de la nueva estación.

    - Para el módulo de Estacionamientos, se permitirá a los usuarios el registro de un nuevo estacionamiento, se solicitará que se dibuje un polígono en un mapa que represente el estacionamiento.

Los requerimientos mencionados anteriormente están generalizados y descritos en un lenguaje entendible, sin embargo, es poco común solo especificar de esta forma los requerimientos funcionales. Actualmente, se utilizan diferentes estrategias como los diagramas UML (Lenguaje Unificado de Modelado) para la representación de estos tipos de requerimientos.

3.3.1 Casos de uso

Los casos de uso describen el comportamiento entre el sistema y la interacción de algún participante con la finalidad de cumplir un objetivo. Dicho comportamiento del sistema debe ser narrado de forma simple, eficaz, entendible, escalable y fácil de usar. Haciendo saber al usuario o participante lo que el sistema deberá hacer, excluyendo, lo que no realizará.

Una parte fundamental para la realización de los casos de uso es identificar los diferentes actores que se tendrán en el sistema. Un actor puede ser una persona o un dispositivo que realice o ejerza algún tipo de interacción.

En el libro "Ingeniería del software Un enfoque práctico" Roger S. Pressman[14], se menciona, que una vez identificado los actores del sistema, para la definición y realización correcta de un caso de uso es necesario realizar una serie de preguntas:

- ¿Quién es el actor principal y quiénes los actores secundarios?
- ¿Cuáles son los objetivos de los actores?
- ¿Qué precondiciones deben de existir antes de comenzar la historia?
- ¿Qué tareas o funciones principales son realizadas por el actor?
- ¿Qué excepciones deben considerarse al describir la historia?
- ¿Cuáles variaciones son posibles en la interacción del actor?
- ¿Qué información del sistema adquiere, produce o cambia el actor?
- ¿Tendrá que informar el actor al sistema acerca de cambios en el ambiente externo?
- ¿Qué información desea obtener el actor del sistema?
- ¿Requiere el actor ser informado sobre cambios inesperados?

En la figura 3.2 se muestran los elementos principales para la elaboración de los diagramas de casos de uso.

Símbolo	Nombre	Descripción
	Actor	Representa un rol. Es quien tendrá interacción con el sistema.
	Caso de uso	Representa el caso de uso, función o acción que se realizará dentro del sistema.

[14] Pressman, Roger,2010, *INGENIERÍA DE SOFTWARE UN ENFOQUE PRÁCTICO*, pag. 114

Símbolo	Nombre	Descripción
———————	Relación	La relación es la conexión entre el actor y el caso de uso.

Figura 3.2 Elementos para realizar un caso de uso.

Una vez contestadas las preguntas anteriores y con la noción de los elementos principales para la elaboración de un caso de uso, se contará con la información necesaria para su elaboración.

3.3.2 Identificación de actores.

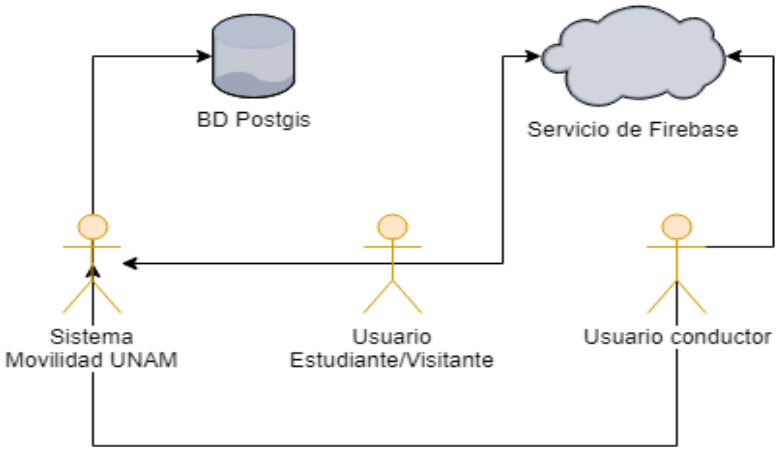


Figura 3.3 Identificación de actores

En la figura 3.3 se presenta un diagrama que permite identificar a los actores principales del sistema, además, se muestra la interacción que habrá entre ellos y los servicios o componentes externos.

3.3.3 Casos de uso para los componentes del sistema.

En la figura 3.3 se muestran los actores principales del sistema, así como los casos de uso con los que interactúan:

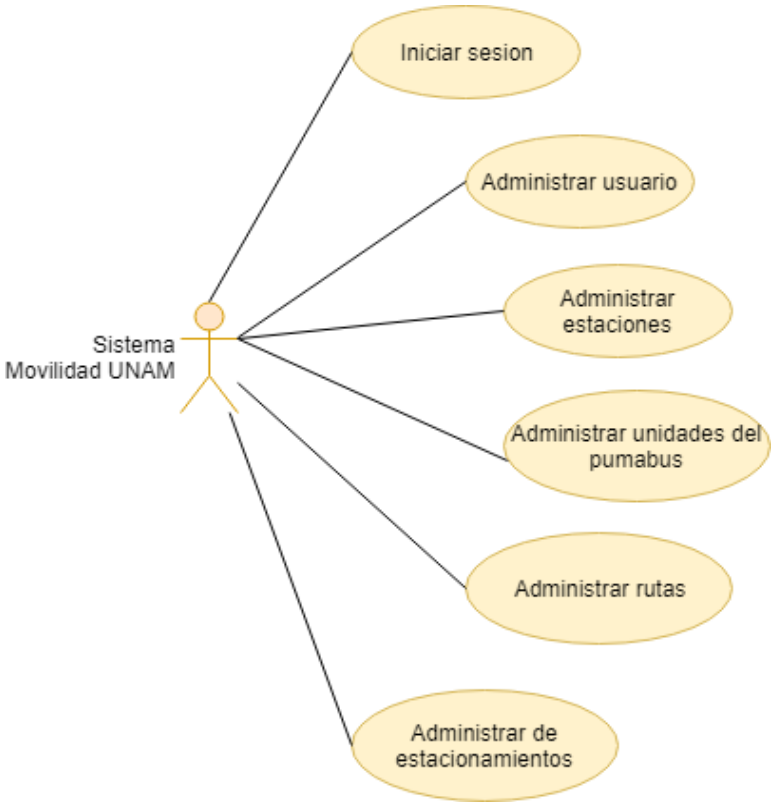


Figura 3.4 Caso de uso para el sistema Movilidad UNAM

Para el caso de uso del Sistema Movilidad UNAM (Figura 3.4) a continuación se muestran las acciones a realizar:

- Iniciar sesión.

  Los usuarios podrán acceder al sistema con su correo electrónico registrado y su contraseña. Solo los usuarios que tengan la autorización para administrar el sistema podrán acceder. Recordemos que el sistema maneja información de unidades de transporte, conductores, estacionamientos, etc. Por esta razón los usuarios que administrarán el sistema deberán de pertenecer a la universidad.

- Administrar usuarios.

  En el módulo de administración de usuarios se permitirá el registro ya sea para el acceso al sistema, conductores del Pumabús o taxis. Para el registro de usuarios se pedirá información como nombre, apellidos, correo electrónico entre más información personal.

  Aunado a lo anterior, se mostrará un listado de todos los usuarios registrados en el sistema, así mismo este listado contendrá a los usuarios que se han registrado por medio de la aplicación móvil para usuarios.

  Este módulo contendrá una opción de búsqueda para hacer un filtrado de los usuarios, ya sea por nombre, apellidos o correo electrónico. Además, por cada usuario es posible editar su información y realizar el guardado con los cambios.

- Administrar estaciones.

  En el módulo de administración de estaciones se permite el registro de estaciones para el Pumabús y las Bicipuma. Se almacenará información como el nombre de la estación y su ubicación. La ubicación de cada estación será especificada con un marcador (latitud y longitud) en un mapa proporcionado por el sistema. La ubicación de la estación deberá estar dentro de Ciudad Universitaria.

  Al igual que en el módulo de usuarios, se mostrará un listado de todas las estaciones registradas, permitiendo al usuario editar la información de cada estación, además, se tendrá la opción de buscar por nombre las estaciones para ser filtradas.

- Administrar unidades del Pumabús y taxis.

  En este módulo se permite el registro de vehículos que brindan el servicio de Pumabús y el servicio de taxis dentro de Ciudad Universitaria. Se almacenará información del vehículo como año de incorporación, modelo, placas y alguna clave que permita identificar al vehículo.

  Se muestra un listado de todas las unidades registradas, por cada unidad es posible editar su información y cambiar de estado activo a inactivo. El estado activo indica que la unidad puede ser ocupada para dar servicio de transporte, sin embargo, si la unidad tiene un estado inactivo significa que por alguna razón no puede ser utilizada. Si la unidad tiene un estado inactivo no es posible editar la información del vehículo. Otra funcionalidad disponible en este módulo es la búsqueda por campos almacenados del vehículo, para poder hacer un filtrado en el listado.

- Administrar rutas.

  En el módulo de administración de rutas se permite el registro de las rutas para el servicio del Pumabús y de la ciclopista para las Bicipuma. Para el registro de una ruta se registrará el nombre y una línea (secuencia de puntos) que representará su geometría. La línea será dibujada en un mapa proporcionado por el sistema.

  Se mostrará un listado de las rutas almacenadas en este módulo, además, por cada ruta será posible editar su información.

- Administrar estacionamientos.

  En este módulo se permite el registro de los estacionamientos que se encuentran dentro de Ciudad Universitaria disponibles para estudiantes, profesores y personas externas a la Universidad. Para cada estacionamiento se deberá registrar información como nombre, capacidad máxima y un polígono que represente la geometría del estacionamiento. Dicho polígono deberá ser dibujado en un mapa proporcionado por el sistema.

  Se mostrará un listado de los estacionamientos registrados en el sistema y por cada estacionamiento es posible editar la información almacenada.

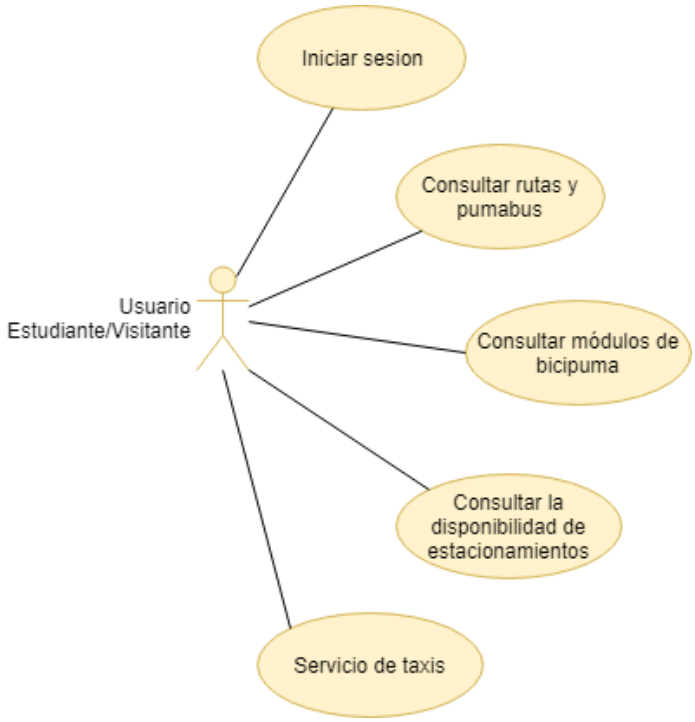


Figura 3.5 Caso de uso para la aplicación de usuarios

Otro de los componentes es la aplicación móvil para la comunidad universitaria o usuarios externos (Figura 3.3). En la figura 3.5 se logra visualizar las acciones disponibles, a continuación se profundizan cada una de ellas:

- Iniciar sesión.

  El acceso a esta aplicación solo se permitirá a los usuarios que estén registrados en el sistema. En dado caso que el usuario no cuente con dicho registro la aplicación permite el registro del usuario, solicitando su información como correo electrónico y una contraseña.

- Consultar rutas y Pumabús.

  La aplicación presentará un listado del nombre de las rutas del Pumabús almacenadas en el sistema, el usuario puede seleccionar alguna de este listado y se mostrarán 3 elementos en un mapa proporcionado por la aplicación: la representación de la ruta en una línea, las estaciones pertenecientes a esa ruta y la ubicación de los vehículos (Pumabus) en circulación de esa ruta.

  La ubicación de los Pumabus cambiará conforme el vehículo se traslada dentro de Ciudad Universitaria.

- Consultar los módulos de Bicipuma.

  Para dicho modulo la aplicación mostrará un mapa con la ruta de la ciclopista (representada por una línea), los módulos de entrega y recepción de bicicletas (representados por un punto), por cada módulo es posible visualizar el número de bicicletas disponibles.

- Consultar la disponibilidad de estacionamientos.

  En este módulo la aplicación mostrará un mapa con todos los estacionamientos (representados por un polígono) dentro de Ciudad Universitaria. Además, por cada estacionamiento es posible visualizar los lugares disponibles. La aplicación deberá sugerir un estacionamiento cercano con lugares disponibles tomando como referencia la ubicación actual del usuario.

- Servicio de taxis.

  En este módulo la aplicación mostrará un mapa con la ubicación del usuario y unidades del servicio de taxis que se encuentren cerca de su ubicación. Si el usuario lo desea, podrá solicitar un servicio al taxista proporcionándole su ubicación. Antes de solicitar el servicio de taxi, el usuario podrá ver la información de la unidad vehicular, así como la información del conductor que tendrá que estar registrado en el sistema.

  Finalmente se encuentra la aplicación móvil para los conductores (Figura 3.3), en la figura 3.6 se muestras las acciones permitidas para estos usuarios, a continuación de describirán:

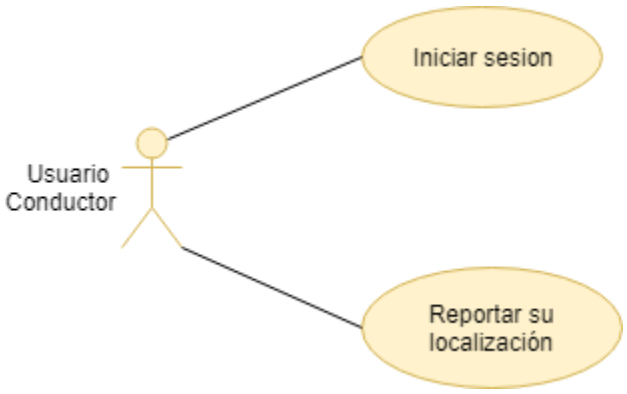


Figura 3.6 Caso de uso para los conductores

- Iniciar sesión.

  El acceso a la aplicación solo se podrá para los usuarios que estén registrados en el sistema y sean conductores las unidades del Pumabús o de los taxis. Si un usuario no tiene su cuenta tendrá que presentarse en alguna dependencia de la UNAM con información personal para que sea registrado en el sistema y se le permita el acceso a la aplicación.

- Reportar localización.

  Cuando el conductor se encuentre en horario laboral, su ubicación será proporcionada para ser mostrada en los mapas de la aplicación para los usuarios.

  En esta aplicación se mostrará un mapa con la ubicación del conductor y se podrá ver el movimiento que tiene al conducir su vehículo. En el caso de que el conductor trabaje en el

servicio de Pumabus, dependiendo de la ruta en la que esté dando servicio, se mostrará la ruta (representada por una línea) a seguir.

Para el caso de los taxistas, si un usuario solicita un servicio de taxi, la aplicación le mostrará la ubicación del cliente para que el taxista pueda pasar a recogerlo y llevarlo a su destino.

3.3.4 Caso de uso detallado.

Ya se mostró un caso de uso con los actores principales del sistema (Figura 3.3), después se profundizó un nivel más con un caso de uso por cada actor identificado (p. ej. figura 3.4), ahora, si se pretende ser más explícito se puede realizar un documento detallado de los casos de uso.

A continuación, se presenta un documento detallado del caso de uso (Figura 3.7) para el módulo de Pumabús para el sistema denominado “Movilidad UNAM”:

	Caso de uso detallado para el registro de una unidad del Pumabús.	
Versión	1.0 (23/04/2018)	
Precondición	Que el usuario este registrado en el sistema. El usuario tiene que autentificarse con su correo electrónico y contraseña elegida.	
Descripción	El sistema permitirá el registro de los camiones a utilizar para el transporte de usuarios, el comportamiento para esta sección se describe en el siguiente caso de uso.	
Secuencia	Paso	Acción
	1	El sistema mostrará un listado con los siguientes campos: ➢ **Nombre de la unidad.** ➢ **Placas** ➢ **Ruta** ➢ **Estatus** El estatus nos permite saber si la unidad se encuentra en uso o en mantenimiento.
	2	Para el listado mostrado se permitirá ordenar de manera ascendente o descendente la información. Por default será ascendente. Además, el listado tendrá un paginador el cual dará la posibilidad al usuario de visualizar completamente la información por páginas.
	4	Se permite registrar unidades del Pumabús, al dar clic en el botón [+ Agregar Unidad] (ver flujo alterno No.1)

	Caso de uso detallado para el registro de una unidad del Pumabús.	
	5	Se podrá editar la información de alguna unidad específica. Esto se podrá realizar al dar clic en el botón [editar] (Ver flujo alterno No.2)
	6	Se cuenta con la posibilidad de ver la información almacenada por cada unidad del Pumabús dando clic en el botón [detalle] (ver flujo alterno No.3)
	7	Se podrá eliminar alguna unidad del Pumabús, se deberá dar clic en el botón [eliminar] (ver flujo alterno No.4)
Flujo alterno No.1 Registro	Al dar clic en el botón [+ Agregar Unidad] el sistema mostrará un modal en el que se presentarán los siguientes campos: ➢ Nombre ➢ Placas ➢ Ruta Los campos son obligatorios para permitir el guardado de la información, de lo contrario, no será posible realizar el almacenamiento y se mostrará el siguiente mensaje "*Para poder guardar la información de manera exitosa es necesario llenar todos los campos*". Por otro lado, es importante verificar que las placas no se repitan con los registros ya almacenados, si se repiten presentar el mensaje "*Las placas ya se encuentran almacenadas en el sistema.*" Al almacenar la información de manera exitosa, se mostrará el siguiente mensaje: "¡Éxito! la información se ha almacenado de manera correcta." Posterior al guardado se muestra el listado actualizado.	
Flujo alterno No.2 Edición	Al dar clic en la opción de editar se abrirá un modal con los mismos campos que para la opción de registro: ➢ Nombre ➢ Placas ➢ Ruta Se precarga la información almacenada para dicho registro. Se permite realizar cambios en la información y almacenarlos (haciendo clic en el botón [Guardar]). Para descartar los cambios bastará con hacer clic en el botón [Cancelar], el cual regresa al listado.	

	Caso de uso detallado para el registro de una unidad del Pumabús.
Flujo alterno No.3 Detalle	Para este caso, de igual manera que para la edición y registro de unidades se presentará un modal con los siguientes campos: ➢ Nombre ➢ Placas ➢ Ruta La información mostrada en estos campos no se permite su edición. Para este caso solo se cuenta con el botón de [Cancelar] en cual cerrara el modal de detalle y mostrara el listado de las unidades.
Flujo alterno No.4 Eliminar	Se permite la eliminación de la información referente a alguna unidad ya almacenada, al dar clic en el botón [Eliminar] se mostrará un modal con el siguiente mensaje: "*¿Desea eliminar la información?*" Si el usuario confirma su petición, la información será eliminada y se mostrará el listado.
Postcondición	Todas las acciones realizadas (registro, edición y eliminación) deberán ser guardadas en la bitácora con la finalidad de llevar un control de la información.
Comentarios	

Figura 3.7 Caso de uso detallado

3.3.5 Diagramas de actividades.

Los diagramas de actividades muestran la secuencia de eventos desde un punto inicial hasta llegar a un punto final y las rutas existentes entre estos dos puntos. Este tipo de diagramas permite visualizar todos los procesos que puede llegar a ejecutar un sistema y como es que interactúan entre sí.

Los diagramas de actividades muestran la lógica de los algoritmos, lógica del negocio, la interacción del usuario con el sistema, además, permiten simplificar procesos que parecen complejos.

A continuación, se presentan los símbolos (Figura 3.8) que permiten la creación de diagramas de actividades.

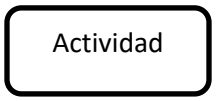
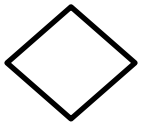
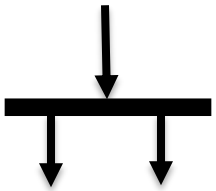
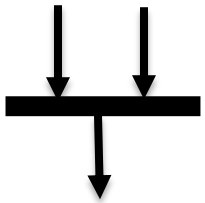
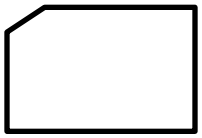
Símbolo	Nombre	Descripción
	Inicio	Representa el inicio de un proceso.
	Actividad	Representa las actividades o acciones permitidas a realizar.
	Condición	Representa la toma de una decisión o elección de una salida dado un conjunto de posibles salidas.
	Transición	Permite realizar conexiones entre los diferentes elementos de un diagrama de actividades.
	Fork	Indica que un conjunto de actividades puede realizarse en paralelo.
	Join	Para poder realizar la siguiente actividad se tuvo que terminar un conjunto de actividades.
	Comentario	Permite especificar un comentario en algún elemento de los diagramas de actividades
	Fin	Representa la finalización de un proceso.

Figura 3.8 Elementos para realizar diagramas de actividades.

3.3.6 Implementación de diagramas de actividades para el sistema

En la actualidad, la seguridad es un aspecto de suma importancia en el desarrollo de software debido a que se maneja información sensible y su acceso debe estar restringido a un conjunto de usuarios. Para los tres componentes principales: Sistema Movilidad UNAM, aplicación móvil para usuarios y aplicación móvil para conductores contarán con un sistema de acceso (autenticación). En la figura 3.9 se especifica el diagrama de actividad de autenticación para estos componentes.

Cada componente del sistema (sistema movilidad UNAM, aplicación móvil para usuarios y aplicación móvil para conductores) tendrá su propio método de autenticación y verificación, sin embargo, estos métodos tendrán la misma lógica.

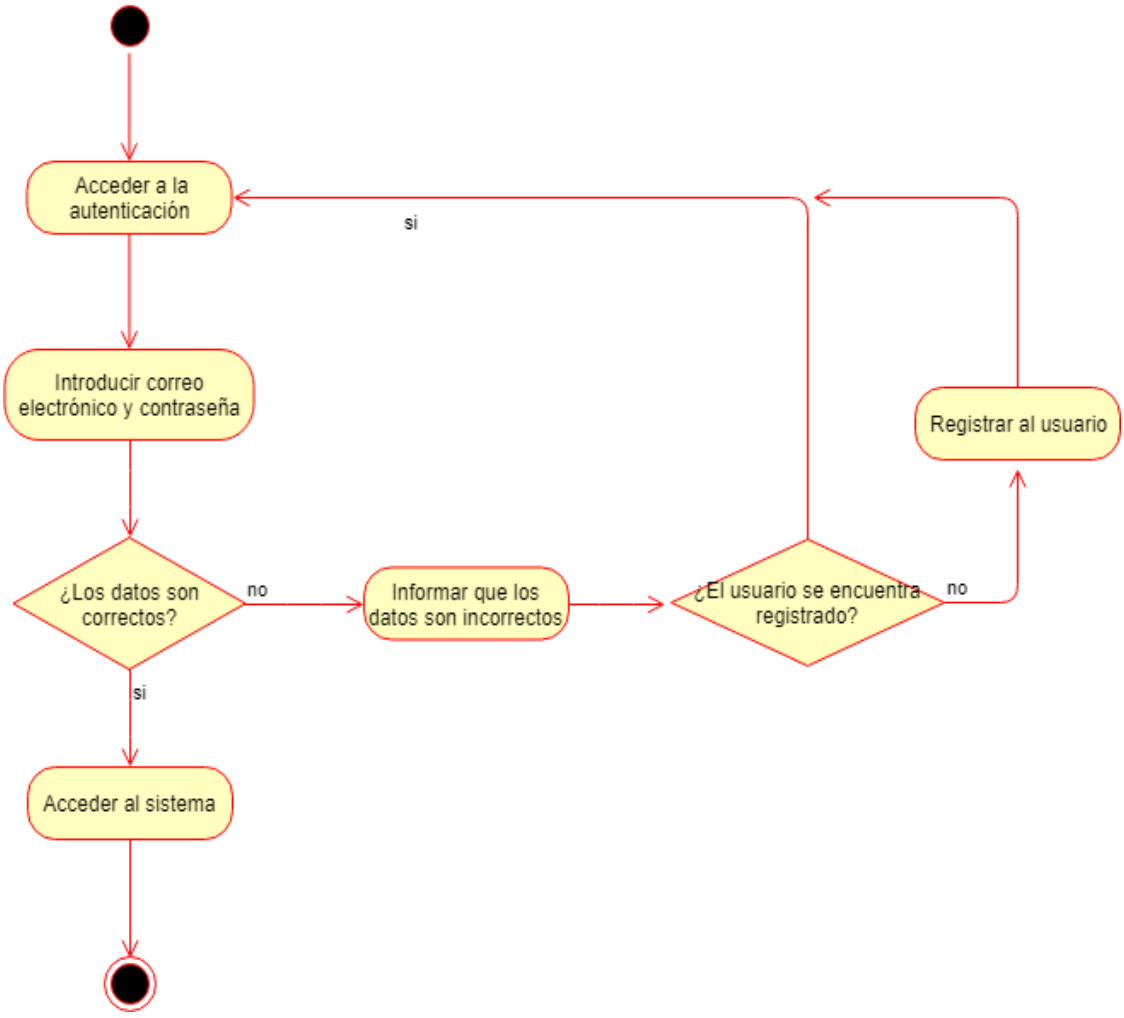


Figura 3.9 Diagrama de actividad para la autenticación.

En seguida se mostrarán diagramas de actividades para los tres componentes principales del sistema.

Para el sistema “Movilidad UNAM” se muestra el diagrama de actividades (figura 3.10) para la acción de registros de usuarios.

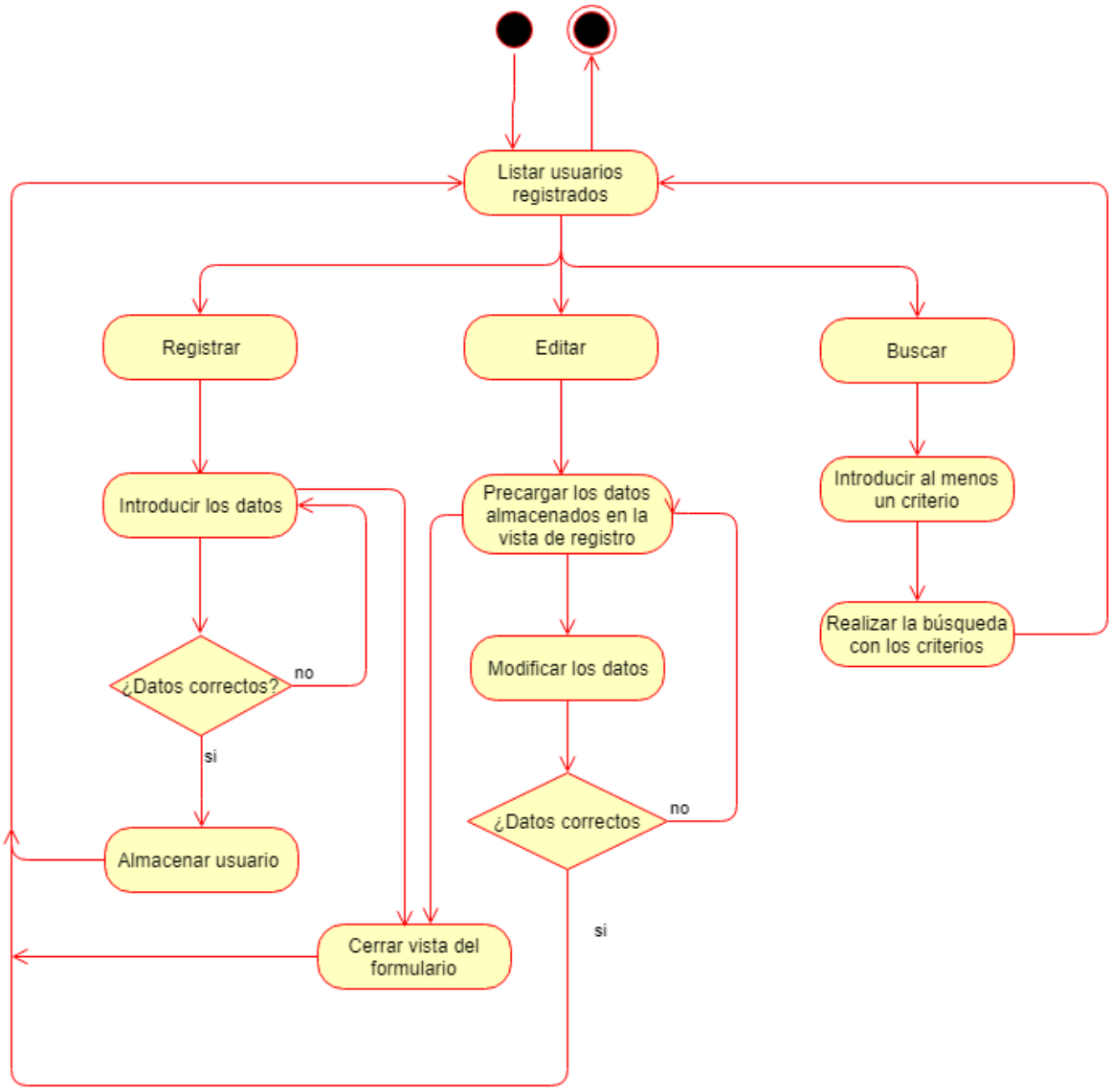


Figura 3.10 Diagrama de actividades para el registro de usuarios.

En la Figura 3.11 se muestra el diagrama de actividades para el registro de estaciones que pertenecen a una ruta del Pumabús. Dicho diagrama pertenece al sistema “Movilidad UNAM”.

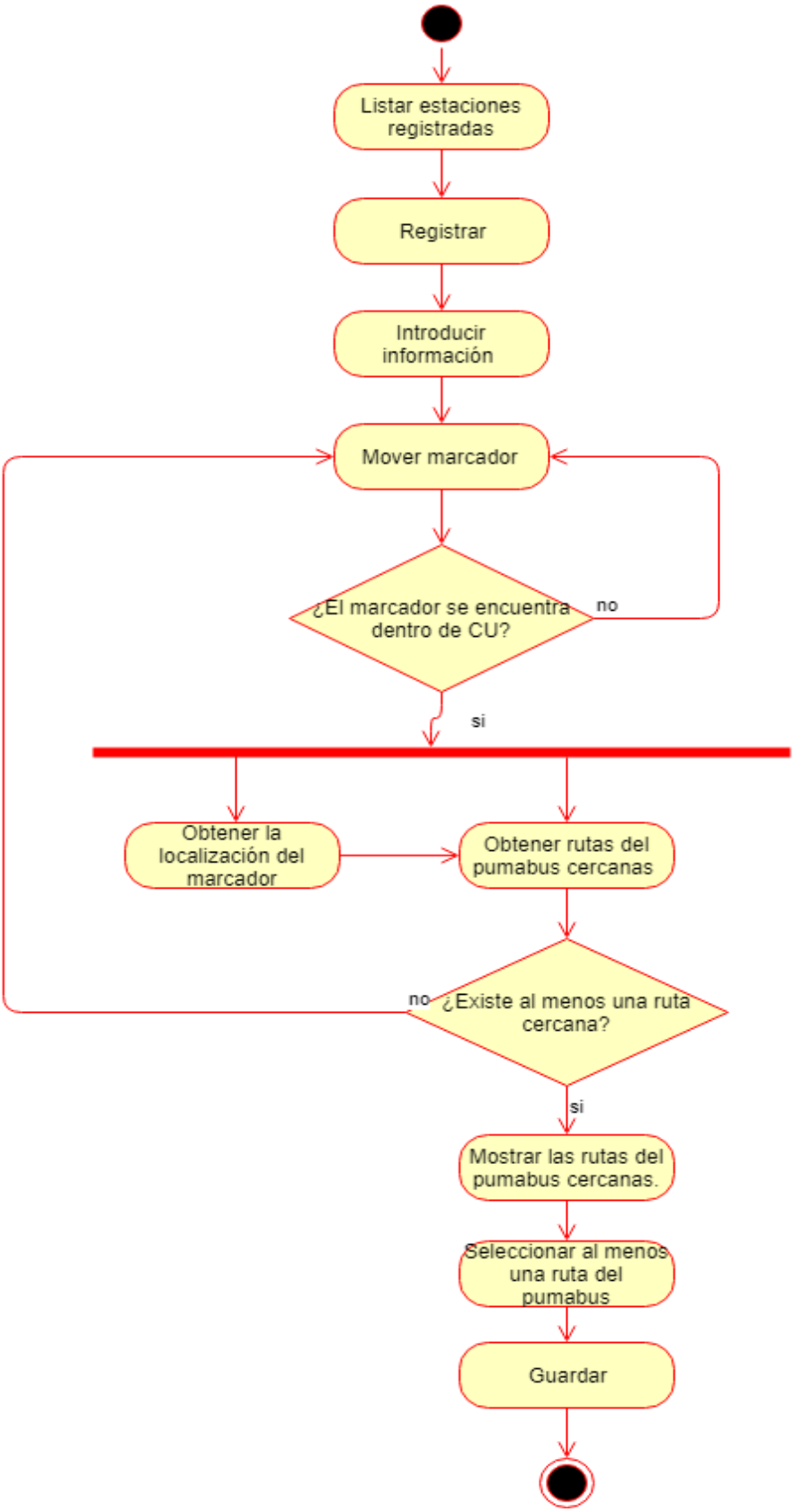


Figura 3.11 Registro de estaciones

En la figura 3.12 se muestra el diagrama de actividades para la aplicación móvil enfocada a los usuarios (estudiantes, académicos o personas externas a la universidad), este diagrama expresa el flujo para obtener la geometría de la ruta, estaciones y ubicación de los Pumabús asociados a dicha ruta.

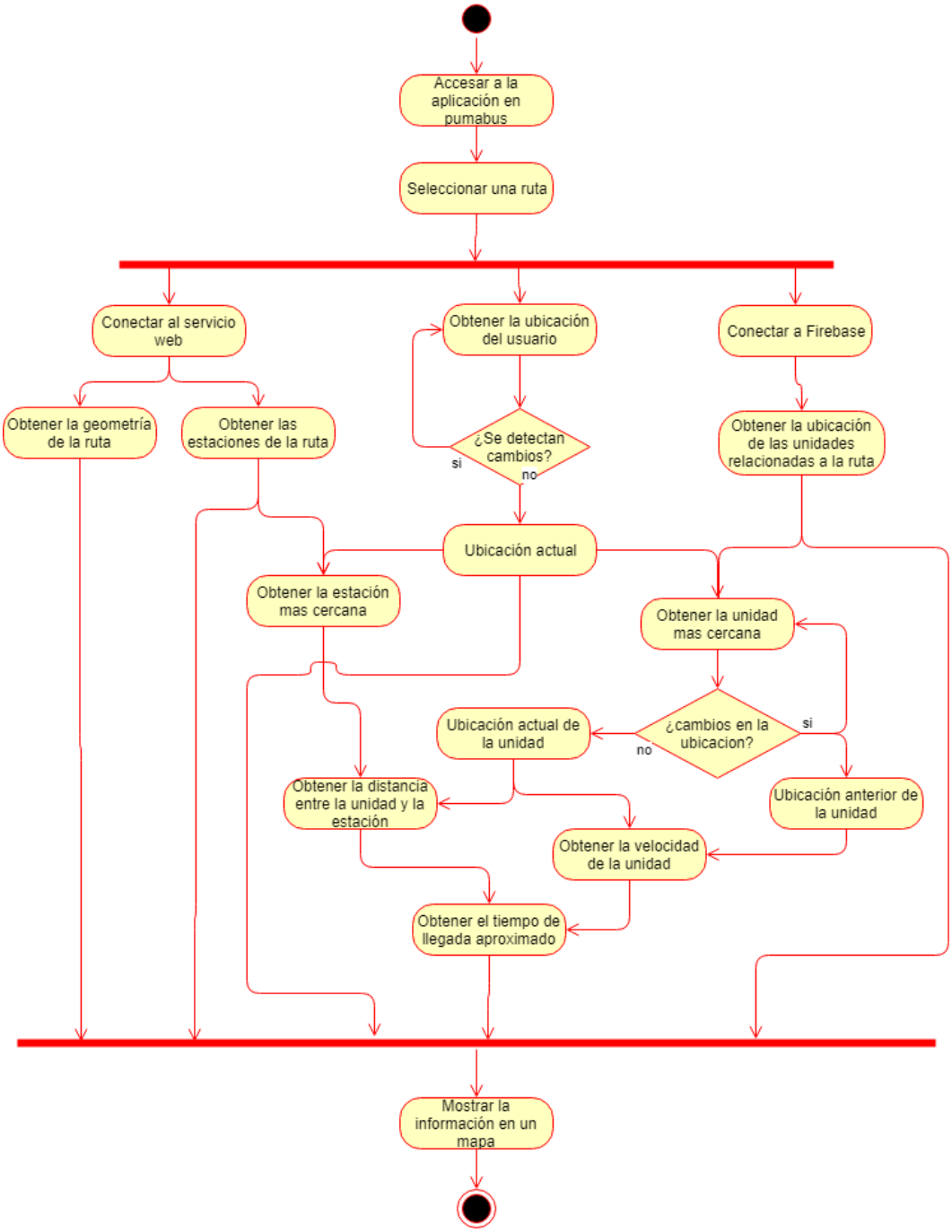


Figura 3.12 Diagrama de actividades para la consulta de las rutas del pumabús.

En el caso de los usuarios que buscan un estacionamiento dentro de Ciudad Universitaria, en la figura 3.13 se muestra el flujo de acciones que se realizarán en esta sección de la aplicación móvil.

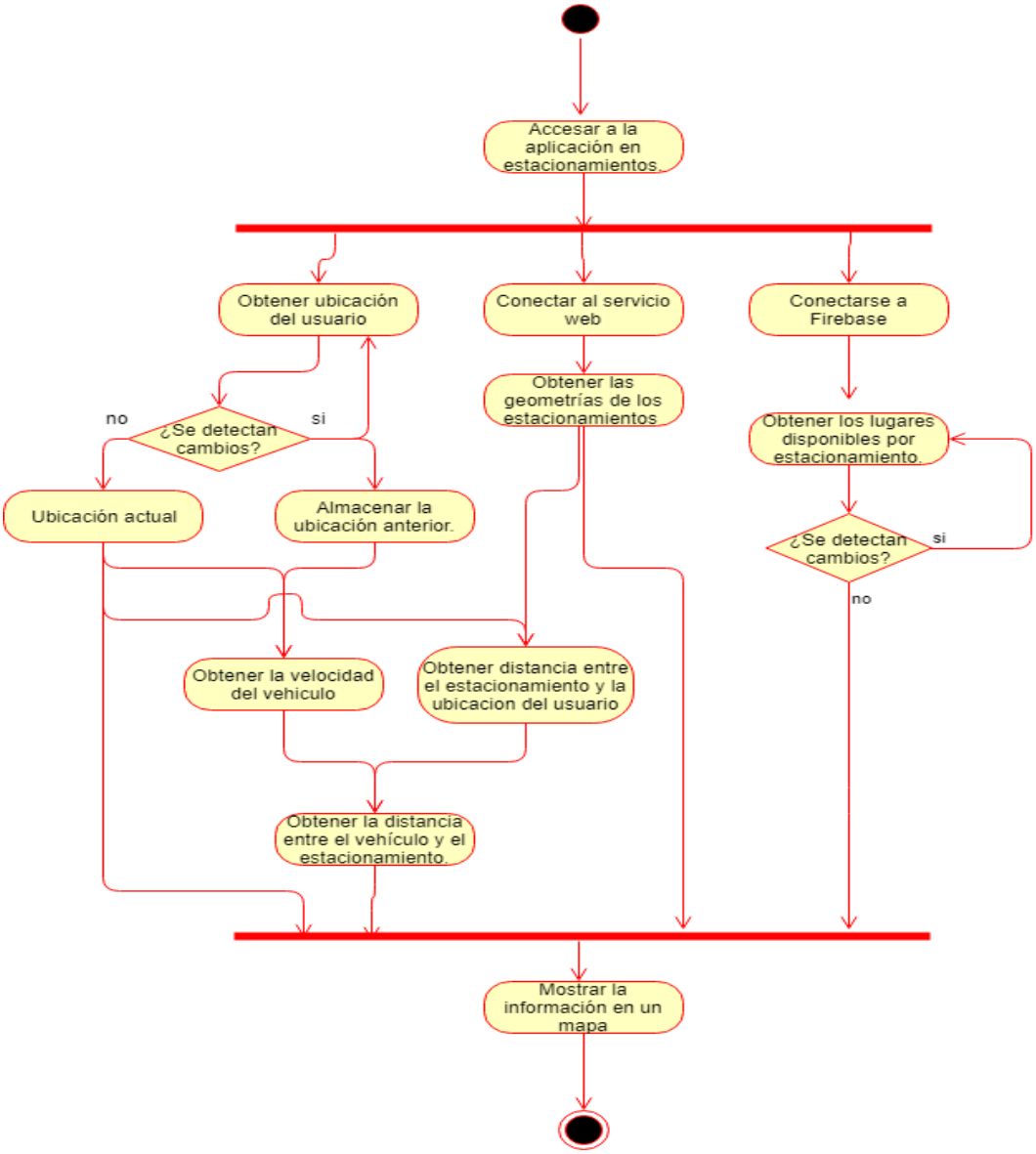


Figura 3.13 Diagrama de actividades para la consulta de estacionamientos.

3.3.7 Prototipos

Un prototipo es una representación visual previa a la versión final de un sistema. Los prototipos son elaborados con la información recaudada en los requerimientos y basados en los diagramas elaborados.

Los prototipos permiten percatarse de los errores que podrían presentarse en el sistema y poder corregirlos a tiempo. Es importante la interacción con el cliente para que esté de acuerdo con lo presentado en los prototipos, ya que con esto se reducirían los cambios en el sistema. Si bien, al elaborar los prototipos se está invirtiendo tiempo, el tiempo es menor al que se le invertiría al realizar un cambio ya al sistema en funcionamiento.

Los prototipos pueden ser elaborados de diferentes maneras, desde hojas de papel y lápiz, hasta la utilización de un software especializado en donde se brinde funcionalidades que permitan cubrir las necesidades.

Además, los prototipos representan una herramienta que permite el fortalecimiento de la comunicación entre el equipo de desarrollo y el cliente, logrando un entendimiento de lo que se quiere en el sistema, tener ideas concretas, aclaración de dudas o aspectos no tan claros.

A continuación, se presentan los prototipos realizados para la aplicación móvil destinada a los usuarios y para el sistema “Movilidad UNAM”.

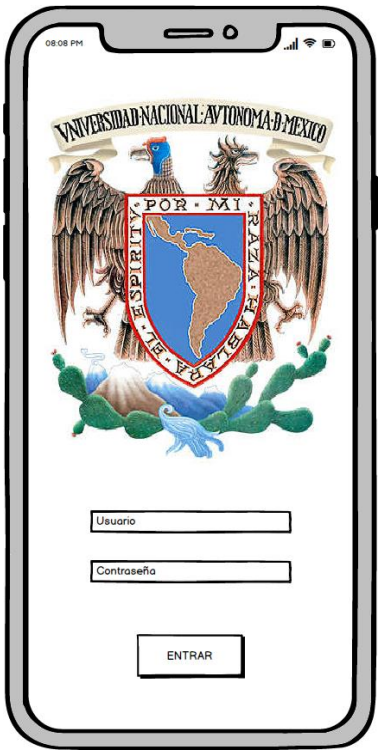


Figura 3.14 Pantalla de Login para los usuarios.

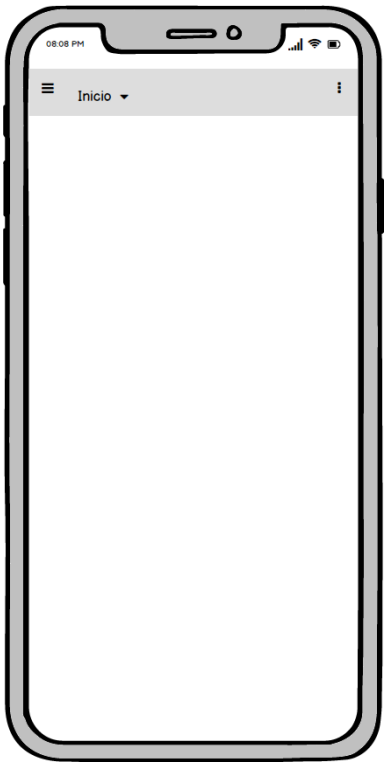


Figura 3.15 Pantalla de inicio, se cuenta con un menú lateral.

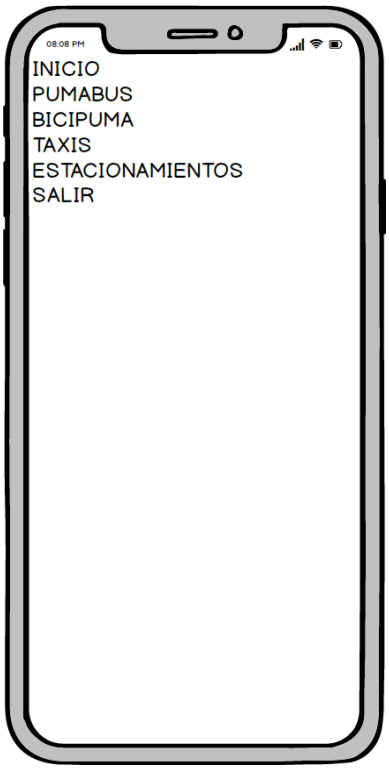


Figura 3.16 Menú lateral, se presentan los servicios proporcionados por la aplicación.

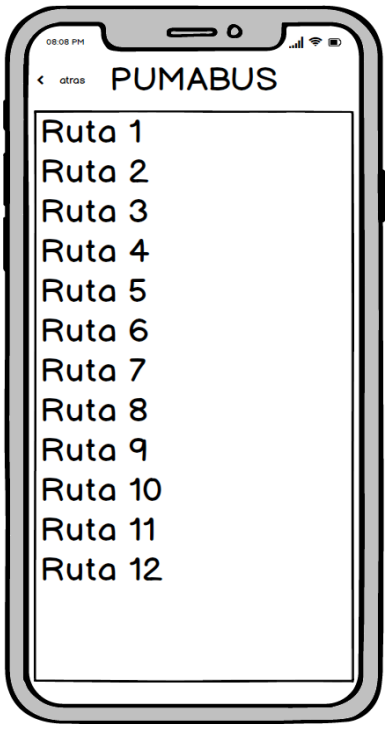


Figura 3.17 Al seleccionar "Pumabús" del menú lateral se mostrarán las rutas disponibles.

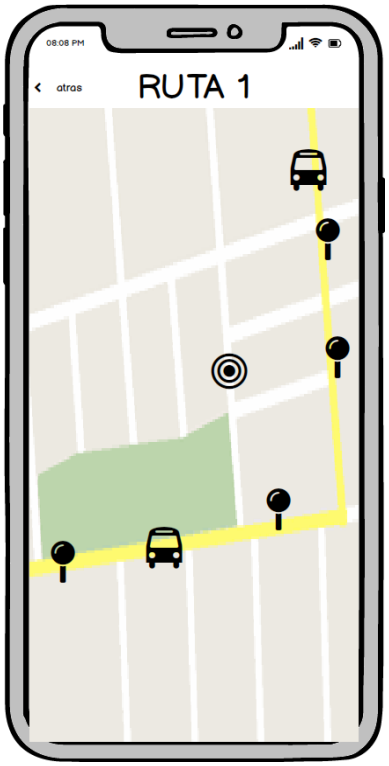


Figura 3.18 Al seleccionar alguna ruta, en un mapa se muestran las unidades, paradas y ubicación del usuario.

Figura 3.19 Al seleccionar "Bicipuma" se visualizarán los centros de bicicletas dentro de Ciudad Universitaria.

Figura 3.20 Al seleccionar "Taxis" del menú lateral se visualizarán las taxis cercanas a la ubicación del usuario

Figura 3.21 Al seleccionar "Estacionamientos" se visualizarán los estacionamientos cercanos y los lugares disponibles.

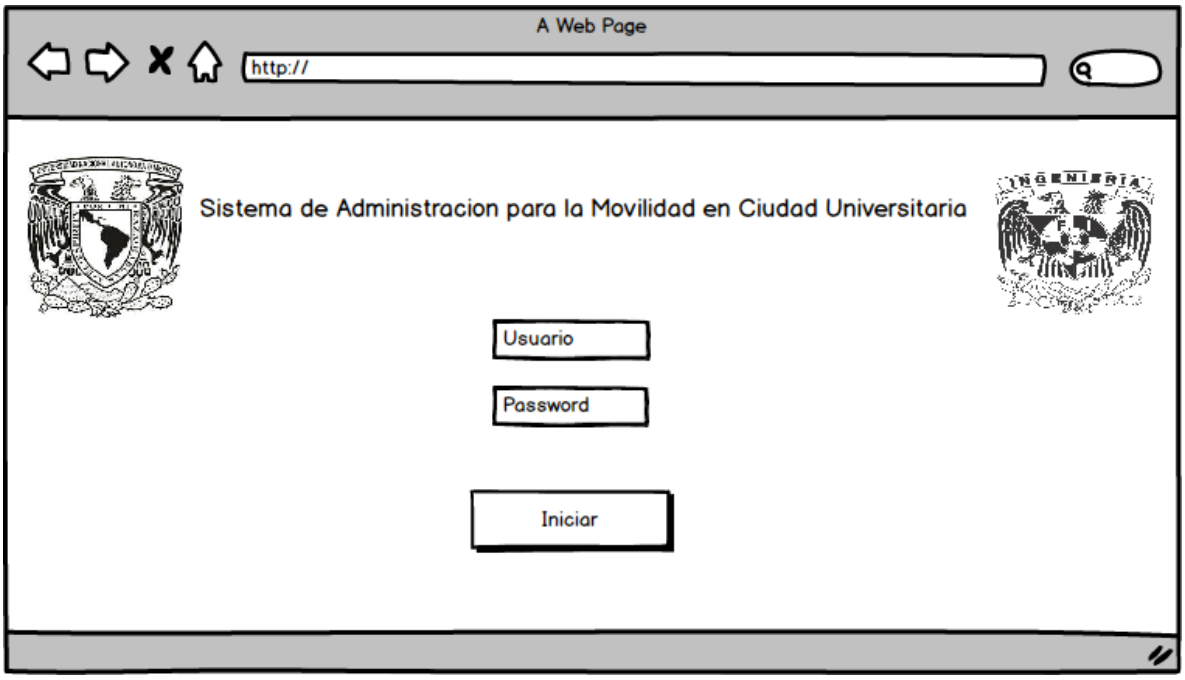


Figura 3.22 Pantalla de Login.

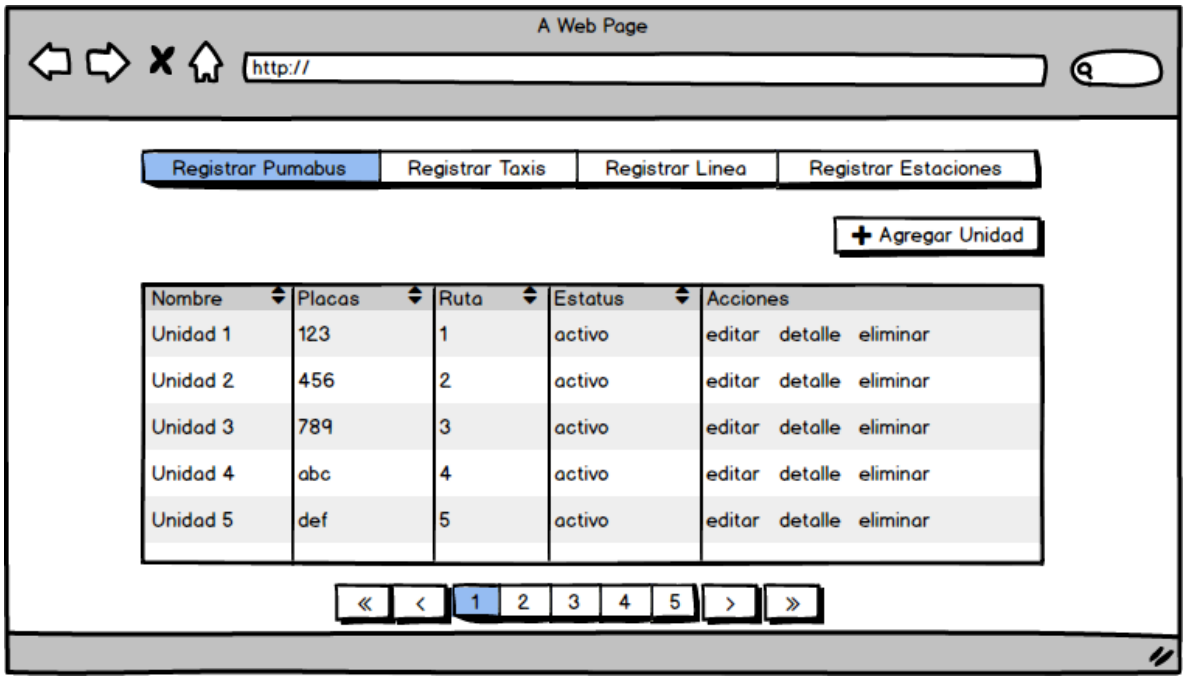


Figura 3.23 Módulo de pumabús: Se en listan las unidades existentes.

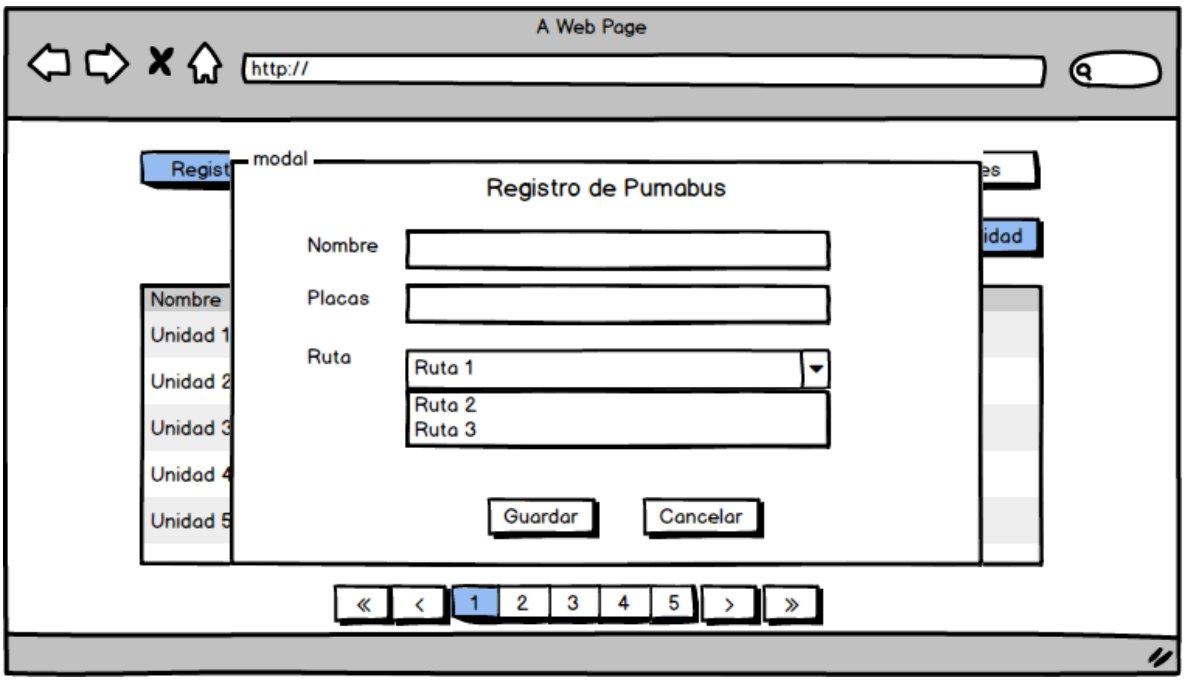


Figura 3.24 Módulo de pumabús: Al dar clic en el botón Agregar unidad se permite el registro de una nueva unidad.

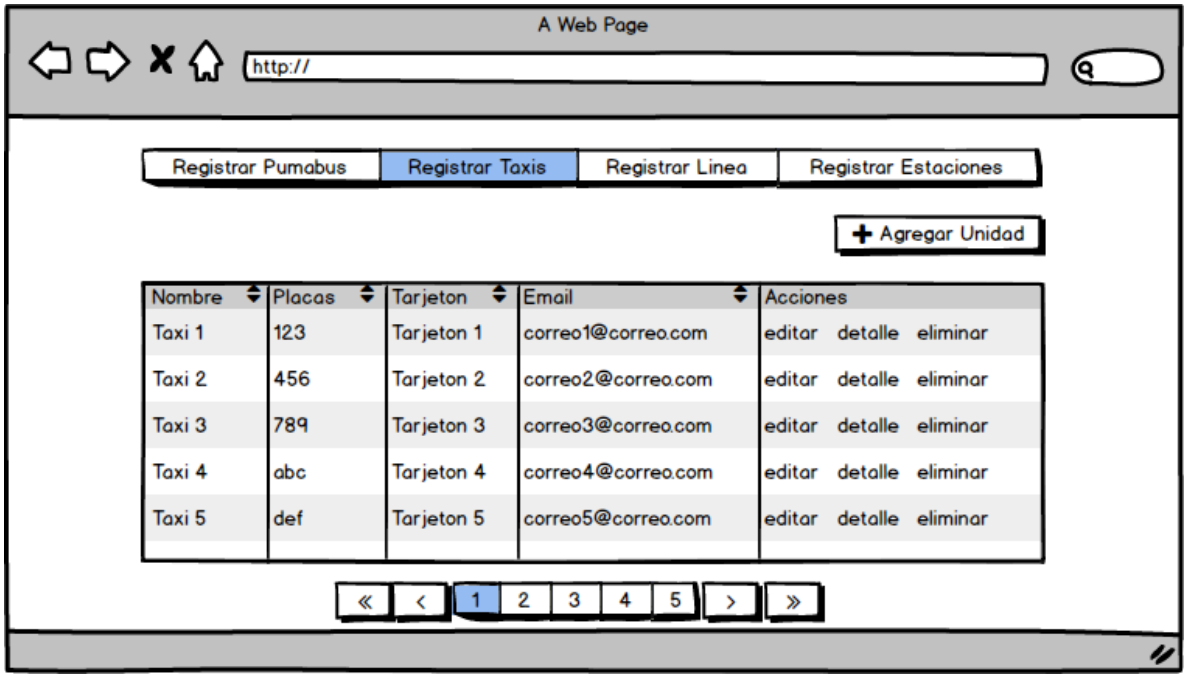


Figura 3.25 Módulo de taxis: Se en listan las unidades existentes.

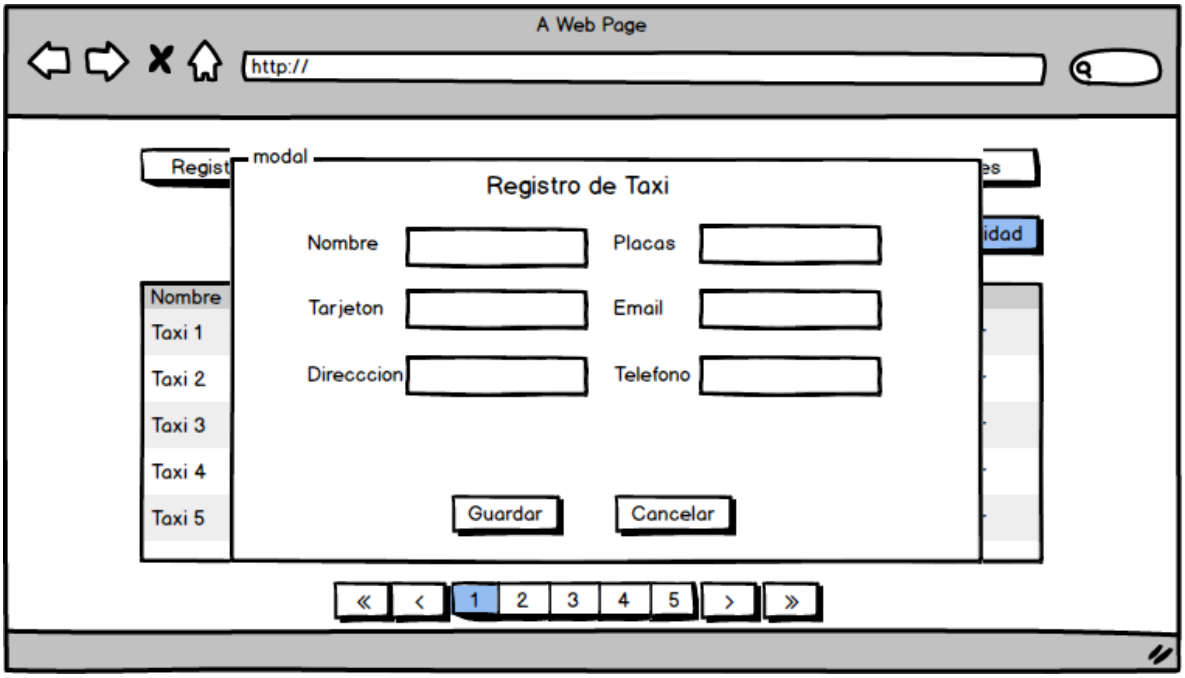


Figura 3.26 Módulo de taxis: Al dar clic en el botón Agregar unidad se permite el registro de una nueva unidad.

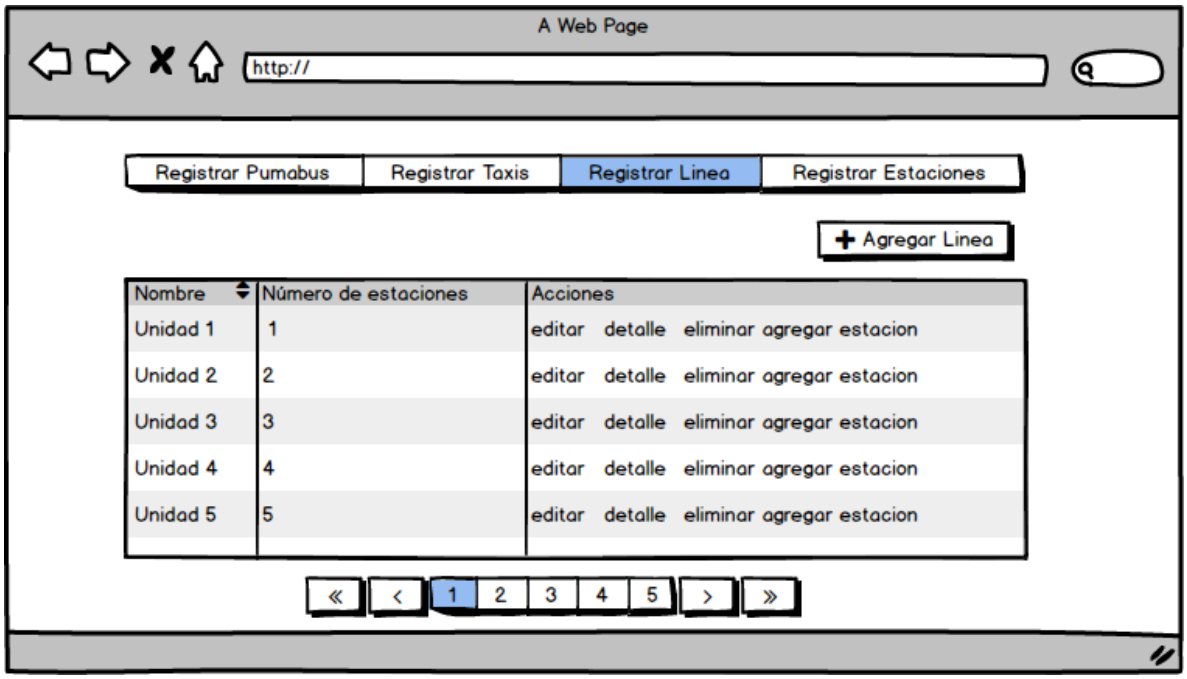


Figura 3.27 Módulo de líneas o rutas del pumabús: Se en listan las rutas existentes.

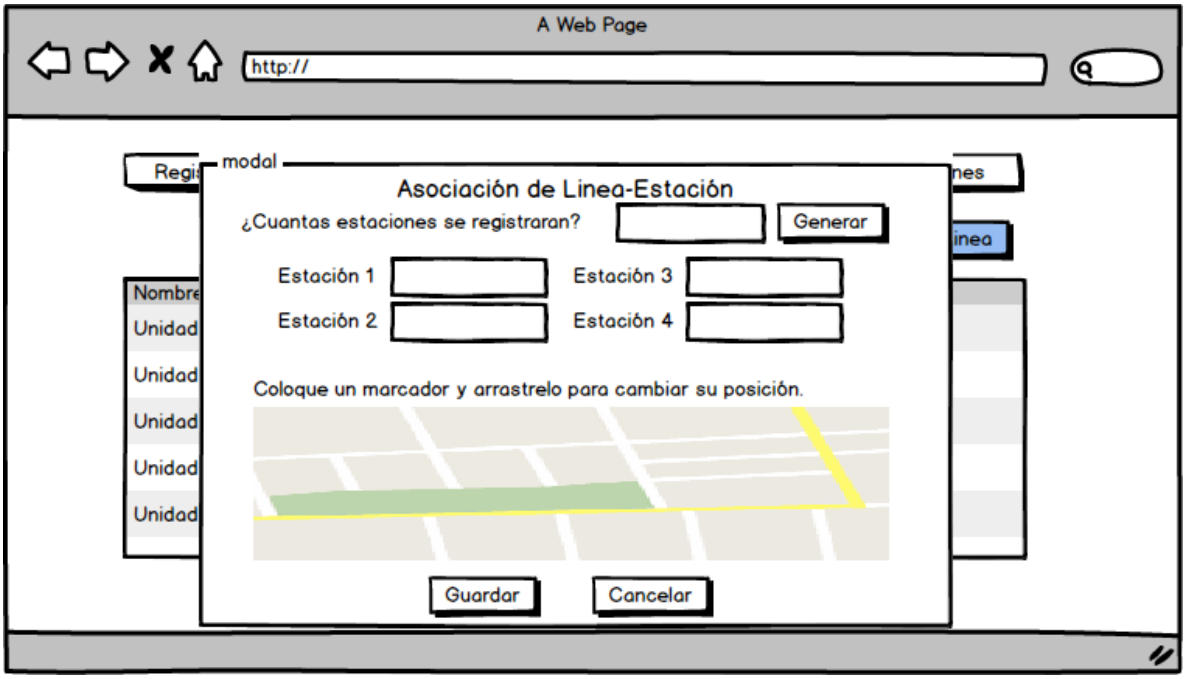


Figura 3.28 Se permite asociar a una ruta del pumabús estaciones existentes.

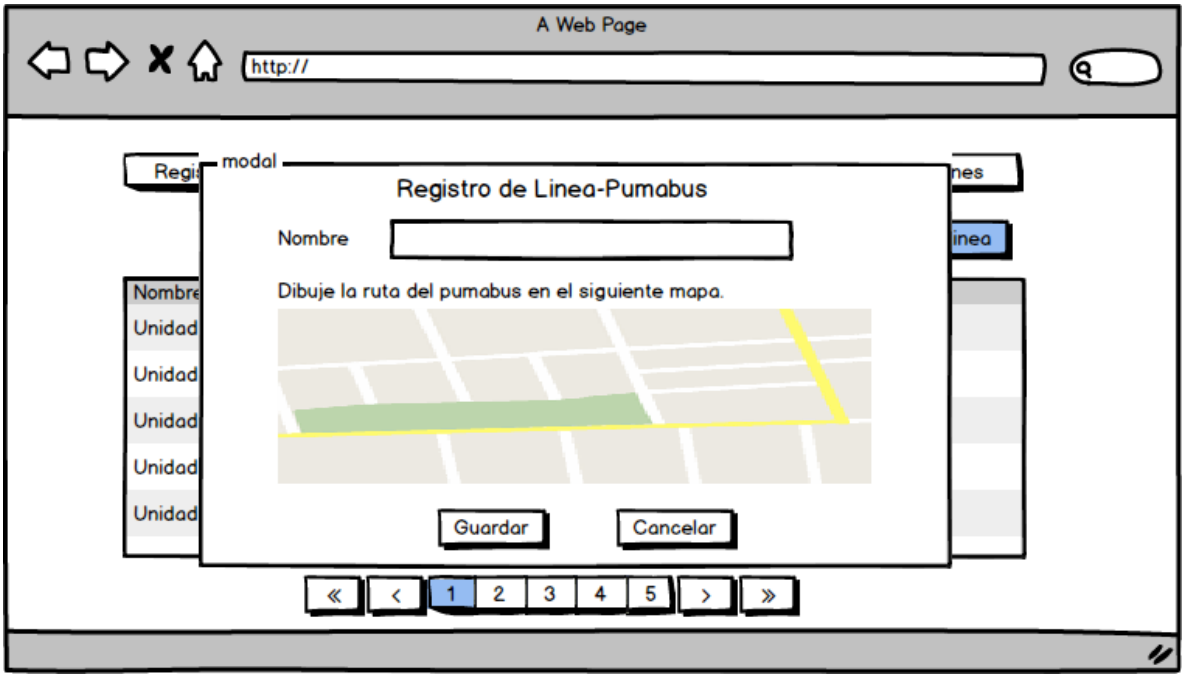


Figura 3.29 Módulo de líneas o rutas del pumabús: Al dar clic en el botón Agregar línea se permite el registro de una nueva línea (Ruta) del pumabús.

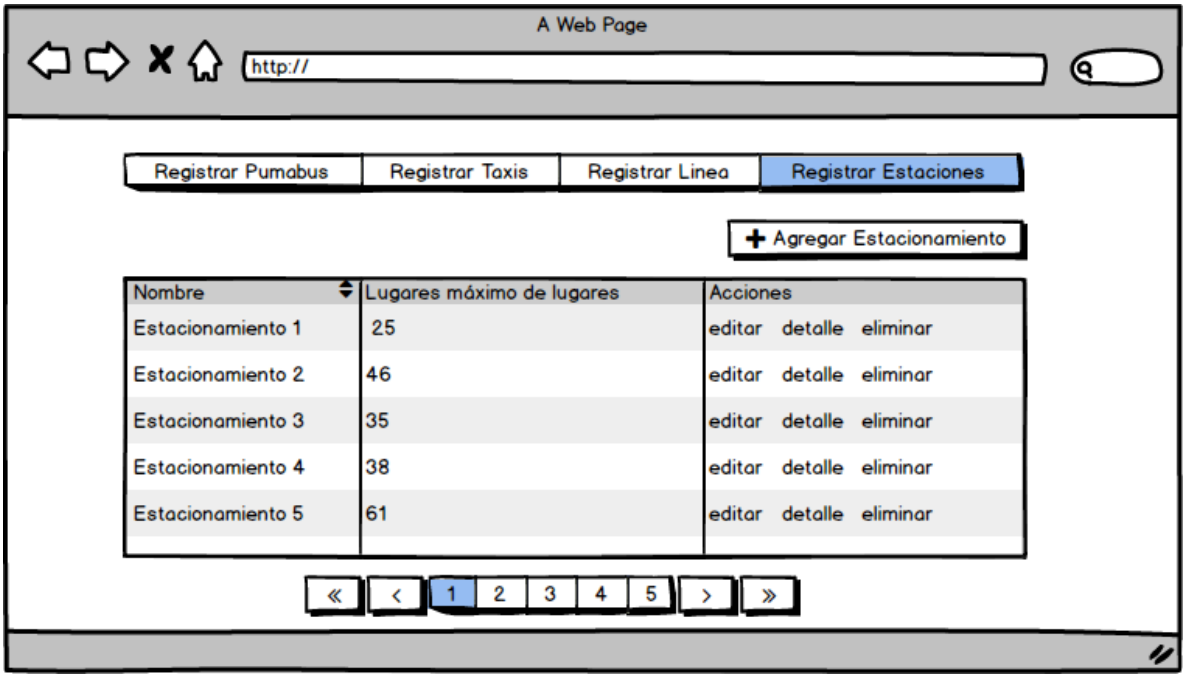


Figura 3.30 Módulo de estacionamientos: Se en listan los estacionamientos existentes.

Figura 3.31 Módulo de estacionamientos: Al dar clic en el botón Agregar Estacionamiento se permite el registro de un nuevo estacionamiento.

3.4 Requerimientos no funcionales

Los requerimientos no funcionales especifican restricciones indirectas que se presentarán en el sistema como pueden ser restricciones de servicios, funciones, tiempo de respuesta, capacidad de almacenamiento y que conllevan a la elección de herramientas para el desarrollo del sistema.

En la figura 3.32 se desglosan los principales tipos de requerimientos no funcionales[15].

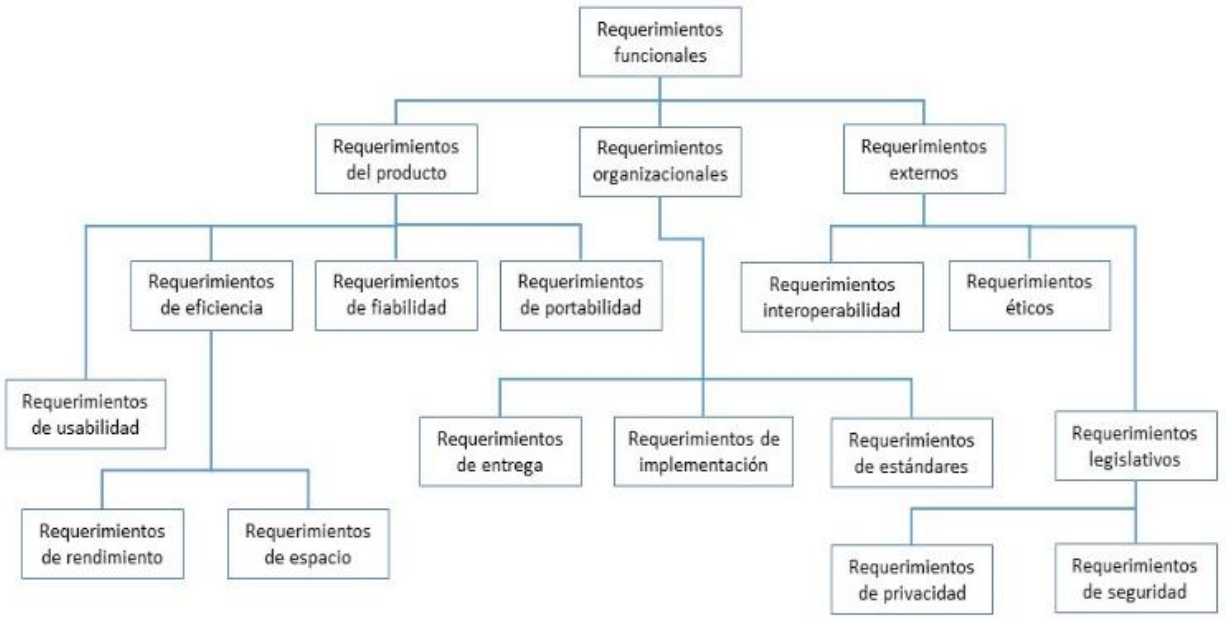


Figura 3.32 Tipos de requerimientos no funcionales (Sommerville)

- Requerimientos del producto.
  Especifican el comportamiento del producto. Por ejemplo rapidez de ejecución del sistema, memoria necesaria, fiabilidad.

- Requerimientos organizacionales.
  Procedentes de las políticas de negocio existentes por parte del cliente. Por ejemplo los estándares que existen al realizar un proceso.

[15] Sommerville,Ian,2005,*Ingenieria Del Software*,pag 112

- Requerimientos extremos

  Derivados de los factores externos del sistema y de su proceso de desarrollo. Por ejemplo, como se llevará a cabo la interacción con sistemas de otras organizaciones.

A diferencia de los requerimientos funcionales, los no funcionales no cuentan con diagramas para su representación, para estos tipos de requerimientos se suelen expresar en oraciones cortas y simples. A continuación (Figura 3.33 y Figura 3.34) se muestra algunas tablas donde se mencionan algunos requerimientos no funcionales para el sistema.

Tipo de requerimiento	Sistema "Movilidad UNAM"
Requerimientos no funcionales de producto	• El sistema será capaz de procesar 100,000 peticiones al sistema por segundo. • Las funcionalidades implementadas en el sistema responderán a los usuarios en menos de 2 segundos. • El sistema será desarrollado aplicando patrones y buenas prácticas de programación. • El sistema estará disponible los 7 días de la semana de un horario de 5 am a 12 pm. • El sistema deberá realizar un respaldo de la información cada 24 horas. • Se deberá contar con un respaldo del sistema por si se llega a tener un fallo, se pueda continuar con el funcionamiento del sistema. • El envió de la información entre los componentes del sistema deberá estar cifrada por el algoritmo. • El tiempo promedio para la duración de una falla en el sistema será de 15 minutos. • El sistema deberá ser desarrollado con "buenas prácticas de programación".
Requerimientos no funcionales organizacionales	• Se deberá de utilizar una combinación entre metodologías tradicionales y metodologías ágiles (Sección 3.1) de desarrollo para la elaboración del sistema. • Se utilizará una herramienta de versionado de código para la elaboración del producto.
Requerimientos no funcionales externos	• La información será confidencial, ningún usuario que no tenga los permisos necesarios no tendrá acceso a los datos de los usuarios.

Figura 3.33 Requerimientos no funcionales para el sistema Movilidad UNAM.

Tipo de requerimiento	Aplicación móvil para usuarios y aplicación móvil para conductores.
Requerimientos no funcionales de producto	• La aplicación obtendrá la información necesaria en menos de 3 segundos. • La información de ubicación de los Pumabus, Bicipuma disponibles y lugares en estacionamientos deberá ser lo más cercano a tiempo real. • La aplicación estará disponible para los Sistemas Operativos Android y IOS. • Los usuarios podrán consultar la ubicación de los Pumabus y la disponibilidad de los estacionamientos los 7 días de la semana con un horario de 6 am a las 10 pm. • Los usuarios podrán consultar las unidades disponibles por estación de las Bicipuma de Lunes a Viernes en un horario 6:30 am a 16:30 pm. • Los conductores solo compartirán su ubicación por medio de la aplicación de lunes a domingo en un horario 6 am a 10 pm siempre y cuando se encuentre dentro de Ciudad Universitaria. • La Base de datos en Firebase deberá estar modularizada para optimizar el tiempo de respuesta. • Las aplicaciones responderán a fallas en un tiempo promedio de 15 minutos.
Requerimientos no funcionales organizacionales	• Se deberá especificar un plan para el lanzamiento de actualizaciones que no afecten a los usuarios. • Se deberá de utilizar una combinación entre metodologías tradicionales y metodologías ágiles (Sección 3.1) de desarrollo para la elaboración de las aplicaciones.
Requerimientos no funcionales externos	• Las aplicaciones serán gratuitas para los usuarios. • Las aplicaciones podrán ser descargadas desde la tienda virtual para Android y IOS.

Figura 3.34 Requerimientos no funcionales para la aplicación móvil de usuarios y conductores.

4. DISEÑO DE ARQUITECTURA

Una de las etapas más importantes en el desarrollo de software es el diseño. El Diseño permite identificar los componentes que proporcionan algún servicio y conforman a un sistema, además, muestra la comunicación que existe entre dichos componentes.

La etapa de diseño debe de ser un proceso iterativo; cuando se realiza por primera vez, el diseño de un software resulta ser de manera general; con el paso del tiempo se puede ir detallando, ya que suelen aparecer errores u omisiones de funcionalidades en etapas previas. Con dicha retroalimentación es posible realizar cambios durante la etapa de diseño.

El diseño de arquitectura establece una organización del sistema con la finalidad de cumplir tanto con los requerimientos funcionales como los requerimientos no funcionales.

4.1 Diagrama de despliegue.

El diagrama de despliegue muestra los elementos del hardware (nodos), así como la relación que tienen entre ellos. Los diagramas de despliegue permiten mostrar elementos de software que se implementan en un elemento de hardware, explicar el procesamiento en tiempo de ejecución y proporciona una vista general de los componentes del sistema.

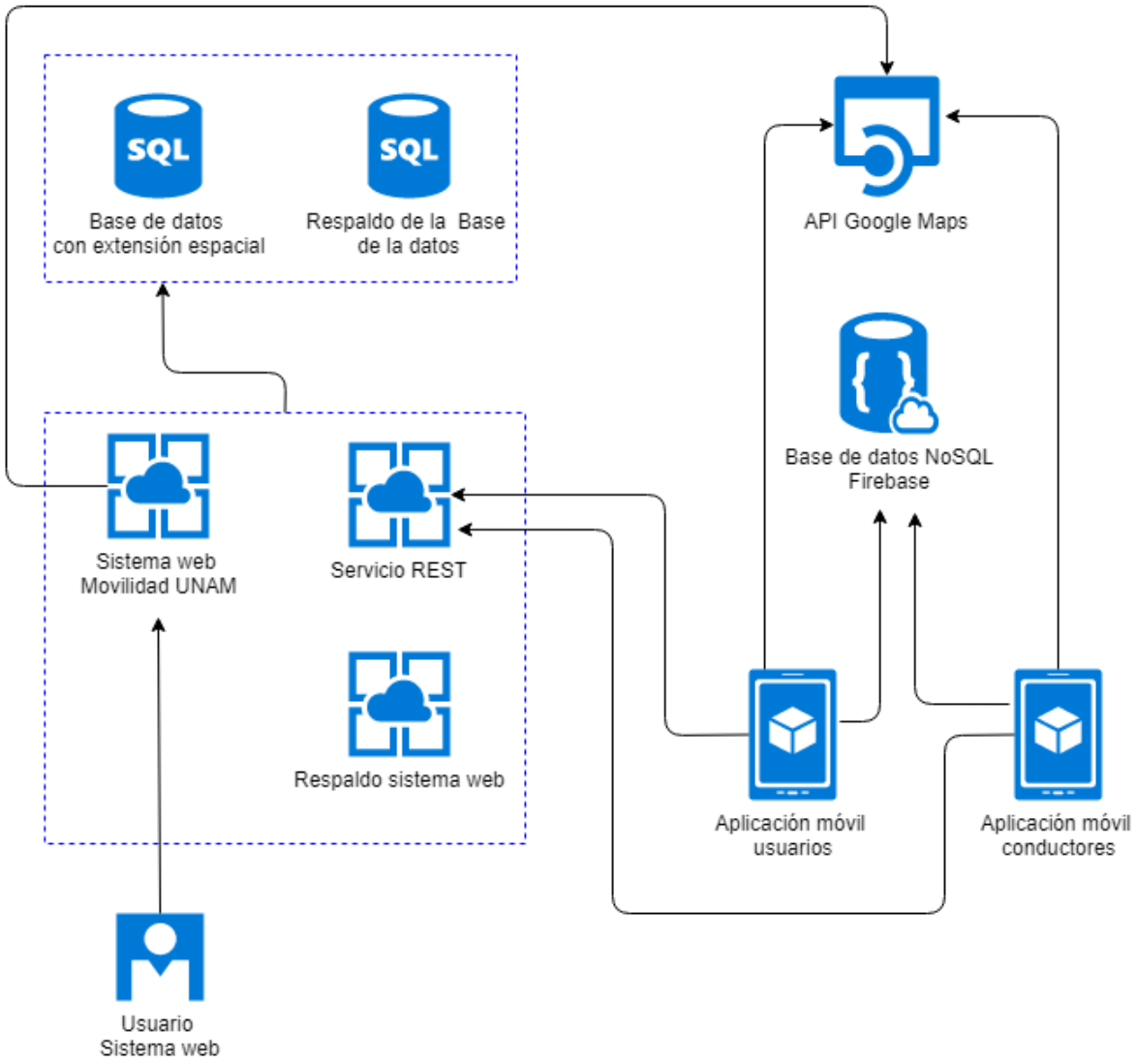


Figura 4.1 Diagrama de despliegue.

En la Figura 4.1 se logra observar la arquitectura del sistema que está formada a su vez por ciertos componentes, dichos componentes interactúan entre sí, intercambiando información, con la finalidad de proveer servicios de manera íntegra y eficaz. A continuación se describe cada componente:

- Base de datos PostGIS: Representa a la extensión espacial de PostgreSQL. Esta base de datos espacial almacenará la información que es procesada por el sistema web Movilidad UNAM. Se guardará la información de usuarios, unidades del Pumabús, rutas del Pumabús, estaciones tanto para el Pumabús, así como también para las Bicipuma, estacionamientos y unidades de taxis. Esta BD tendrá una extensión espacial (para este

caso se ha decidido PostGIS) que permite almacenar geometrías como puntos, líneas y polígonos. Para poder interactuar con la BD de forma gráfica se hará uso de la interfaz gráfica pgAdmin. Así mismo, es posible observar que la BD contará con un esquema de respaldos. Si la BD principal llegara a sufrir un ataque que impidiera su acceso por parte del sistema, el respaldo será utilizado para resolver el problema.

- Movilidad UNAM: Este sistema permitirá el registro de información a la Base de datos PostGIS (punto anterior). Dentro de este componente se hará uso de una herramienta en particular para la representación de las geometrías: API de Google Maps. Así mismo, el sistema web pone a disposición un servicio REST para la consulta de la información almacenada en el sistema. Similar al respaldo de la Base de datos, el sistema contará con su propio respaldo de archivos, si en dado caso sufriera un ataque o tuviera un problema con el acceso al sistema, este respaldo actuaría como el principal, permitiendo el continuo uso del sistema y del servicio REST que provee información a las aplicaciones.

- Firebase Data Base: Herramienta proporcionada por Google que permite almacenar información dinámica. Esta base de datos tendrá la información relacionada con los conductores: Pumabús, Taxis, lugares disponibles en los estacionamientos y bicicletas disponibles por cada estación.

- Aplicación para usuarios: Esta aplicación estará disponible para la comunidad universitaria y personas externas, en ella podrán consultar los diferentes servicios:
    - Consultar en tiempo real la ubicación de las unidades del Pumabús.
    - Consultar en tiempo real las unidades disponibles de bicicletas por estación.
    - Consultar las taxis que se encuentran cerca de su ubicación.
    - Consultar cada ruta y sus estaciones disponibles.

Para la obtención de esta información, es necesario la conexión a Firebase Data Base y al sistema Movilidad UNAM (web service). Para la presentación de la información geométrica como estaciones y rutas se utilizara la API de Google Maps.

- Aplicación para conductores: Esta aplicación solo estará disponible para los usuarios que estén dados de alta en el sistema Movilidad UNAM y que funjan un rol de conductor para las unidades del Pumabus o ya sea si son conductores de un taxi colectivo dentro de ciudad universitaria. La finalidad de esta aplicación será la sobre escritura en la base de datos de Firebase de su ubicación con forme va cambiando. Dicha aplicación también hará uso de la API de Google Maps para el despliegue de información geométrica.

4.2 Sistema web Movilidad UNAM.

Para la construcción del sistema web (Movilidad UNAM) se utilizará una arquitectura multicapa, la cual será explicada a continuación. Se detallará la interacción de los componentes que conforman el sistema web y como está construido e interacción de los elementos.

4.2.1 Sistema multicapa.

Como se da a entender, el sistema multicapa se encarga de dividir un sistema en diversas capas. Cada capa se encarga de realizar un proceso dentro del sistema, permitiendo un diseño modular. Una de las principales características del sistema multicapa es la separación de la lógica de negocios con la lógica de diseño.

Un sistema multicapa cuenta con tres elementos principales:

- Capa de presentación.
  También llamada capa de usuario, esta capa es la encargada de presentar información al usuario así como capturar información proveniente del usuario.
  La capa de presentación únicamente tendrá comunicación con la capa de negocio.
  Además la capa de usuario debe ser entendible y fácil de usar para el usuario.

- Capa de negocio.
  La capa de negocio es la encargada de gestionar la lógica de negocio. Principalmente se manipula y se transforman los datos obtenidos desde la capa de presentación o la capa de acceso a datos.

- Capa de acceso a datos.
  Es la encargada de almacenar o recuperar la información que se encuentra en la Base de datos, dicha capa solo tendrá comunicación con la capa de negocio.

En la figura 4.2 Se ilustran los elementos mencionados anteriormente de un sistema multicapa.

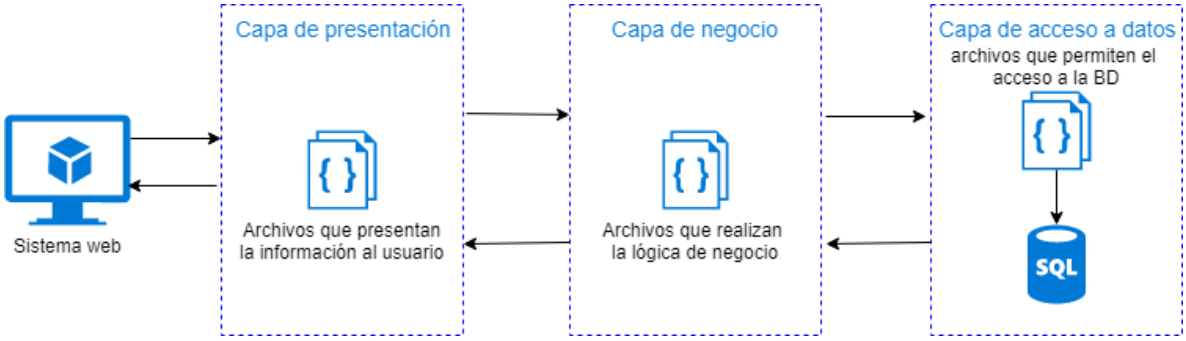


Figura 4.2 Sistema multicapa

Ahora bien, para el caso del sistema web Movilidad UNAM estará estructurado con los elementos de un sistema multicapa: capa de presentación, capa de negocio y la capa de acceso a datos. Aunado a esto, se utilizará el patrón MVC (Modelo Vista Controlador). En la figura 4.3 podemos apreciar la estructura del sistema Movilidad UNAM con el patrón MVC. Dicho patrón será explicado a continuación.

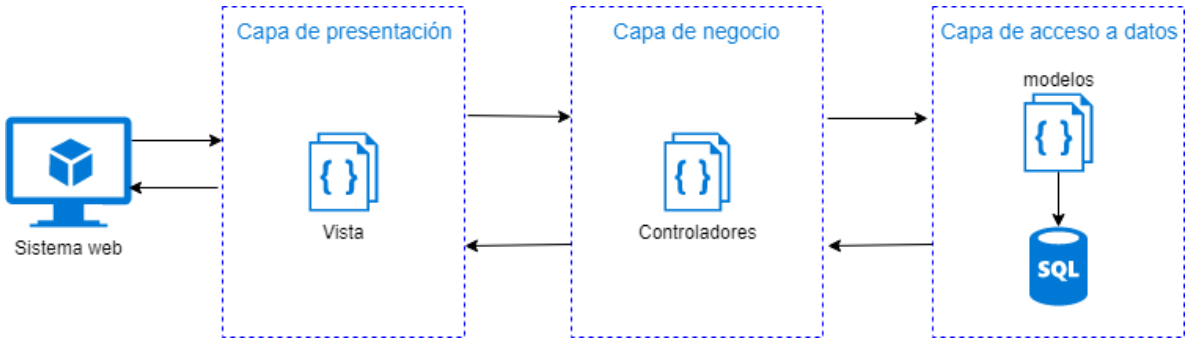


Figura 4.3 Sistema multicapa para el sistema movilidad UNAM.

4.2.2 Modelo Vista Controlador (MVC).

El patrón MVC permite definir la arquitectura de los sistemas, logrando crear sistemas estructurados y robustos, además, de ser escalables. Dicho patrón separa los datos o información, la lógica de negocio y la interfaz mostrada a los usuarios.

El patrón MVC cuenta con tres elementos:

- Modelo: El modelo se encarga de manipular y gestionar los datos. Este componente es el único que tiene acceso para realizar acciones en la BD (agregar, actualizar, eliminar).

- Vista: Este componente se encarga de mostrar la información al usuario final por medio de pantallas, páginas, formularios, entre otros elementos.

- Controlador: Es el encargado de recibir, atender y procesar las instrucciones generadas por el usuario final. Es el intermedio entre la vista y el modelo.

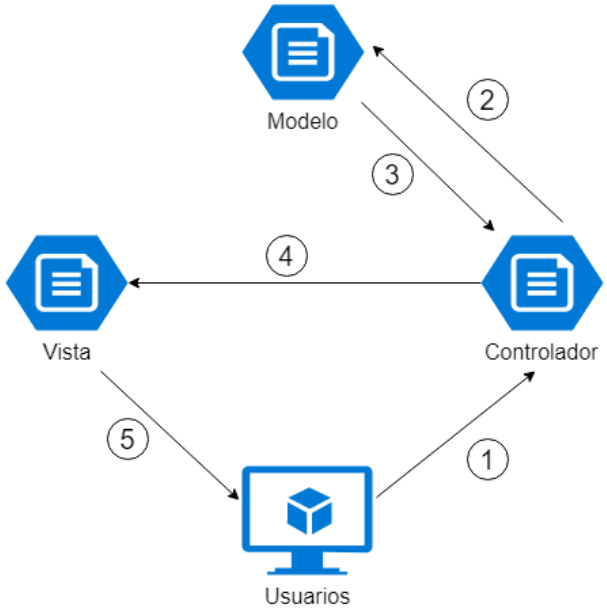


Figura 4.4 Componentes del patrón MVC

En la figura 4.4 se ilustra cómo interactúan los componentes del patrón MVC, a continuación, se explica cada punto:

1. Los usuarios realizan alguna petición al sistema, dicha petición llega al controlador el cual tendrá que procesarla.
2. En dado caso que se necesiten datos almacenados en la BD el controlador realizará una petición al modelo. En dado caso que no se necesite la intervención del modelo se realiza el punto 4.
3. El modelo recibe una petición por parte del controlador, el modelo procesa dicha petición y obtiene información de la BD, los cuales regresa como respuesta de la petición al controlador.
4. Una vez que el controlador recibe la información por parte del modelo, este procesa los datos y los envía a la vista, si en la petición del usuario no es necesario utilizar el modelo (punto 2) el controlador manda directamente a la vista.
5. La vista recibe los datos por parte del controlador y son presentados al usuario final.

4.2.3 Interacción de los componentes MVC en el sistema.

Como se ha mencionado anteriormente, para el desarrollo del sistema movilidad UNAM se utilizará el patrón MVC. Es importante definir como los componentes interactúan entre ellos ya que facilita el entendimiento de la programación en la fase de desarrollo.

A continuación (Figura 4.5) se presenta un diagrama general de la interacción de los componentes:

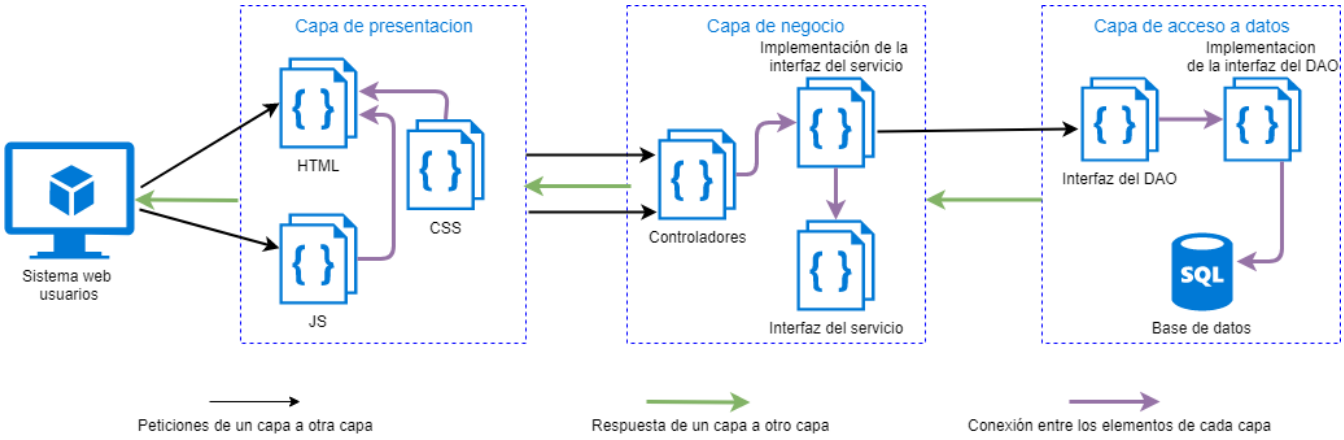


Figura 4.5 Interacción de los componentes en MVC

Como podemos observar en la figura 3.5 se han agregado dos nuevos componentes:

- Servicio: Este componente es un intermedio entre los DAOs (Data Access Object) y los controladores. Los servicios pueden ejecutar algunas reglas de negocio, además, los DAOs serán referenciados desde el servicio permitiendo tener un código más limpio y mejor estructurado.

- DAO: Los DAO (Objeto de acceso a datos) son los componentes que contendrán la lógica para obtención, modificación y gestionamiento de los datos almacenados en la BD.

Por otro lado, observamos que los usuarios pueden tener interacción con elementos visuales presentados en el sistema, esto se debe a que la herramienta JavaScript (incorporada en el sistema) permite asociar eventos a elementos visuales, por ejemplo: un botón, cuando se da clic se habilita un evento que realiza una petición al sistema.

4.3 Aplicaciones móviles (Usuarios y Conductores)

Como se especificó en el Capítulo 2. Marco teórico se utilizará el framework Ionic para la construcción de las aplicaciones móviles. En Ionic se hace uso de tres elementos principales:

- Templates. Vistas mostradas al usuario que utilizará la aplicación. Dicho componente tiene una fuerte interacción con los componentes.

- Component. Los componentes son los elementos básicos para manejar la lógica de alguna vista (template) en específico.

- Services. Los servicios son elementos que tienen alguna característica especial o realizan una tarea particular, su principal función es compartir recursos entre los componentes.

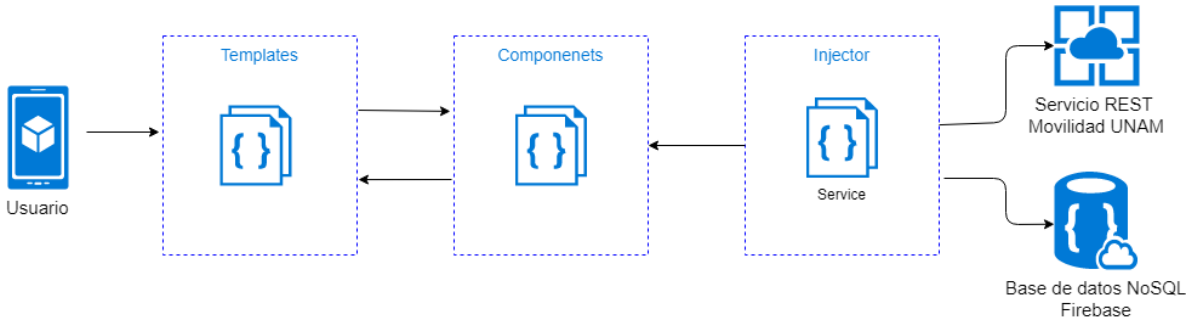


Figura 4.6 Componentes de las aplicaciones móviles para el sistema.

En la figura 4.6 se logra observar los componentes mencionados anteriormente, una característica principal para la construcción de las aplicaciones es que en los servicios se tendrá la conexión a dos elementos fundamentales para su funcionamiento:

- Servicio REST del sistema web Movilidad UNAM.
- Conexión a la BD NoSQL en Firebase.

Dichos servicios permitirán a la aplicación móvil a realizar peticiones a estos dos elementos, por ejemplo, se permitirá la escritura y lectura de información en Firebase, obtener información espacial de la BD PostGIS (rutas, estaciones, etc) para poder ser mostrada al usuario.

Ionic hace uso de la herramienta Angular, y si observamos detenidamente la figura 4.6 podríamos asimilarlo con una Arquitectura MVC, en donde claramente los template serían las vistas, los componentes serían los controladores ya que manejan la lógica, y los servicios serían los modelos, en este último punto si tendríamos una diferencia, ya que los modelos normalmente están encargados de obtener información de la BD, sin embargo, para el caso de las aplicaciones móviles no se obtendrá información de una Base de datos, si no será de servicios que proveen otros elementos.

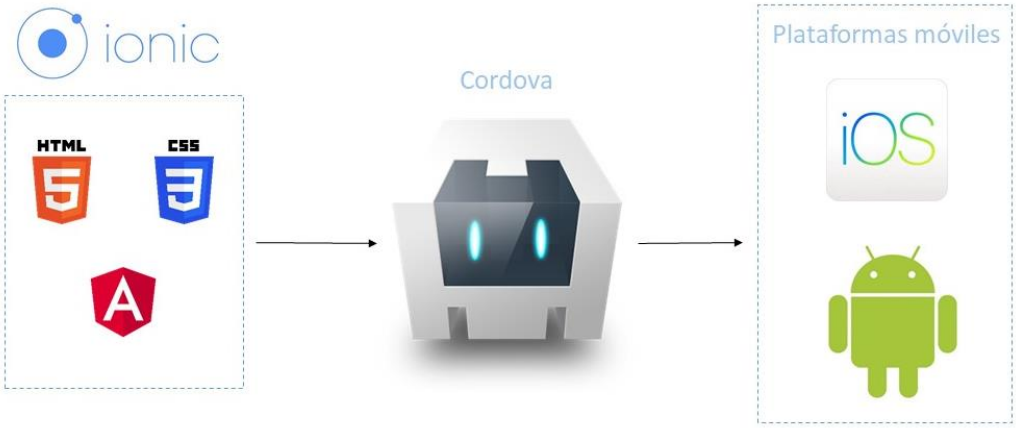


Figura 4.7 Construcción de una aplicación con Ionic

Una vez que se tiene construida la aplicación móvil con Ionic (a base de herramientas como HTML, CSS, Angular) es necesario que sea procesada por un módulo llamado Cordova.

Cordova provee todo lo necesario para que una aplicación hecha en Ionic pueda hacer uso de funciones nativas del teléfono como: Cámara, la agenda, GPS, etc.

Cuando ya se tiene construida la aplicación en Ionic y las funciones nativas proporcionadas por Cordova, Ionic permite crear las aplicaciones para las plataformas iOS y Android de forma nativa.

4.4 Diagramas de clases.

Los diagramas de clases permiten tener un panorama de la estructura que tendrá el sistema, ya que se modelan las clases, atributos, operaciones y relaciones entre los objetos. En otras palabras, los diagramas de clases describen lo que deberá estar presente en el sistema.

Se debe de tomar en cuenta que los diagramas UML se establecieron como un estándar para el entendimiento y descripción de la programación orientada a objetos (POO), por esta razón, los componentes involucrados en los diagramas de clases representan las clases que se programarán en la fase de desarrollo.

Los diagramas de clases muestran relaciones entre las clases definidas y no un flujo de datos entre ellas. Los elementos principales de los modelos de clases son:

- Clase
  Las clases estarán formadas por atributos y métodos (Figura 4.8).

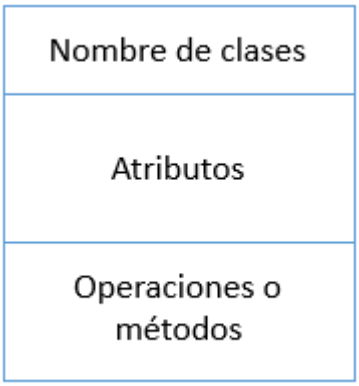


Figura 4.8 Representación de una clase.

- Relaciones:

Símbolo	Nombre de relación	Descripción
	Herencia	Una subclase hereda los atributos y métodos de una súper clase.

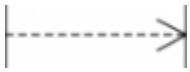
Símbolo	Nombre de relación	Descripción
	Composición	El tiempo de vida de una clase depende de tiempo de vida de otra.
	Agregación	El tiempo de vida es independiente de otro.
	Asociación	Permite una interacción entre los objetos que están relacionados.
	Uso	La creación de un objeto de alguna clase depende de otro objeto.

Figura 4.9 Relaciones en el modelo de clases.

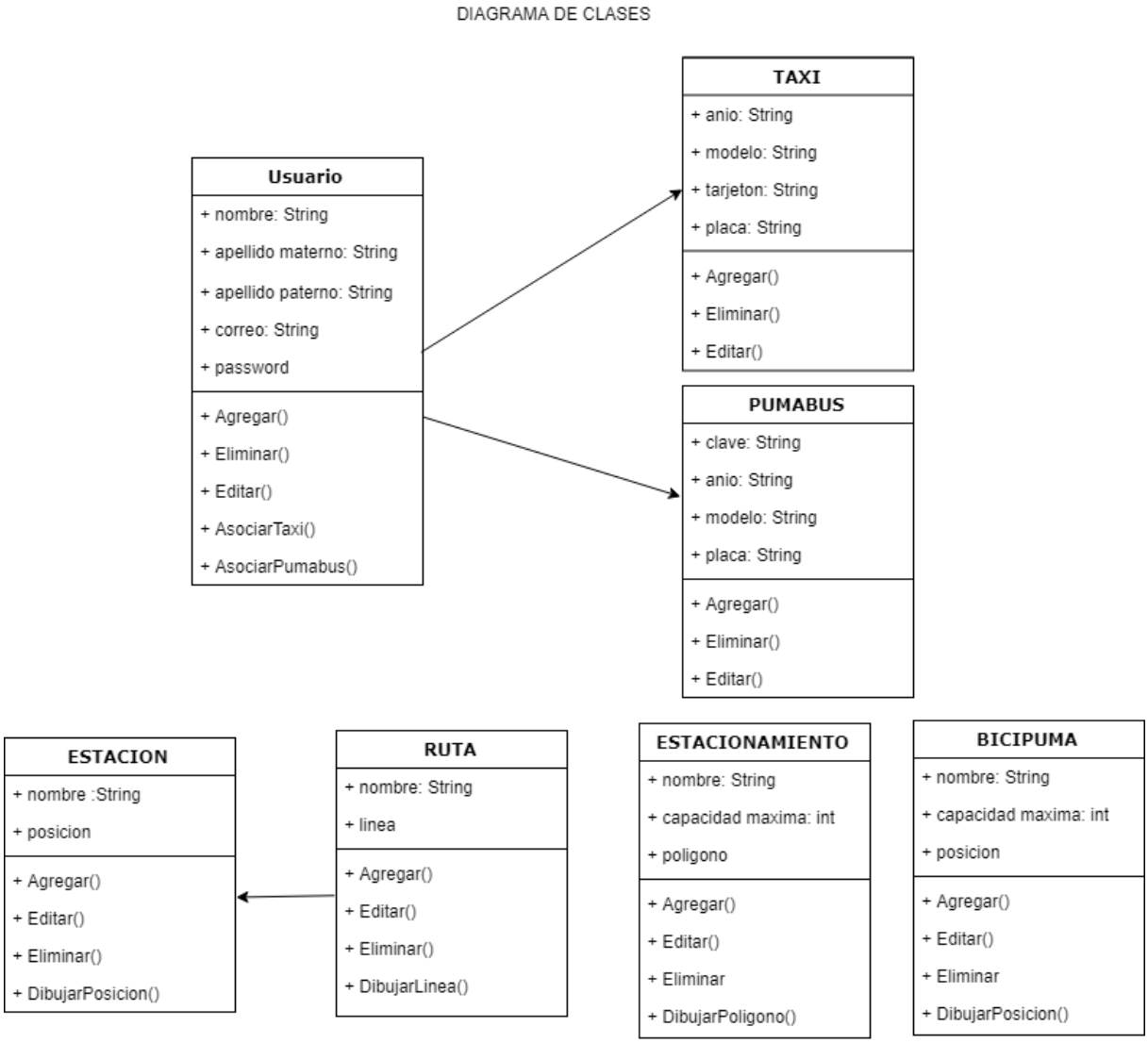


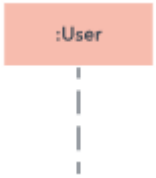
Figura 4.10 Diagrama de clases para el sistema Movilidad UNAM.

En la figura 4.10 se muestra el diagrama de clases para el sistema “Movilidad UNAM” donde se muestra un ejemplo del diseño de clases a utilizar así como la relación que existe entre ellas.

4.5 Diagramas de secuencia

Un diagrama de secuencia permite ilustrar las líneas de vida de los procesos que se llevan a cabo en el sistema (definidos en los casos de uso), así como también, las respuestas que se presentan antes de que acabe el ciclo de vida de los procesos a través del tiempo. Gracias a estos diagramas se logra identificar las interacciones y lazos de comunicación que se establecerán entre los diferentes componentes.

De manera general, los diagramas de secuencia muestran las interacciones entre actores y objetos que permiten la ejecución de un flujo o proceso de negocio.
En la figura 4.11 se presenta la simbología para la elaboración de un diagrama de secuencia.

Símbolo	Nombre	Descripción
	Objeto	Representación de una clase u objeto
	Casilla de activación	Representa el tiempo en que un objeto estará vivo y podrá interactuar con los demás objetos.
	Actor	Elemento que tiene interacción con el sistema, puede tratarse de un elemento externo al sistema.
	Paquete	Elementos interactivos del sistema.
	Línea de vida	Línea discontinua vertical. Representa la interacción de los eventos mientras transcurre el tiempo.
	Bucle de opción	Es utilizado para incorporar sentencias de control.

Símbolo	Nombre	Descripción
	Alternativas	Permite la toma de decisiones en el sistema.
	Mensaje síncrono	Cuando se emite una petición se debe de esperar un mensaje para poder proseguir el flujo del diagrama.
	Mensaje asíncrono	La petición no necesita una respuesta para que se pueda continuar el flujo del diagrama.
	Mensaje de respuesta asíncrono	Respuesta sin mensaje.
	Crear mensaje asíncrono	Se crea un nuevo objeto.
	Mensaje de respuesta	Respuestas a las peticiones.
	Eliminar mensaje	Al finalizar el flujo este mensaje destruye el objeto.

Figura 4.11 Elementos para elaborar un diagrama de secuencia.

Basándose en la especificación de los diagramas de secuencia en el lenguaje unificado de modelado (UML) se realizaron los siguientes diagramas:

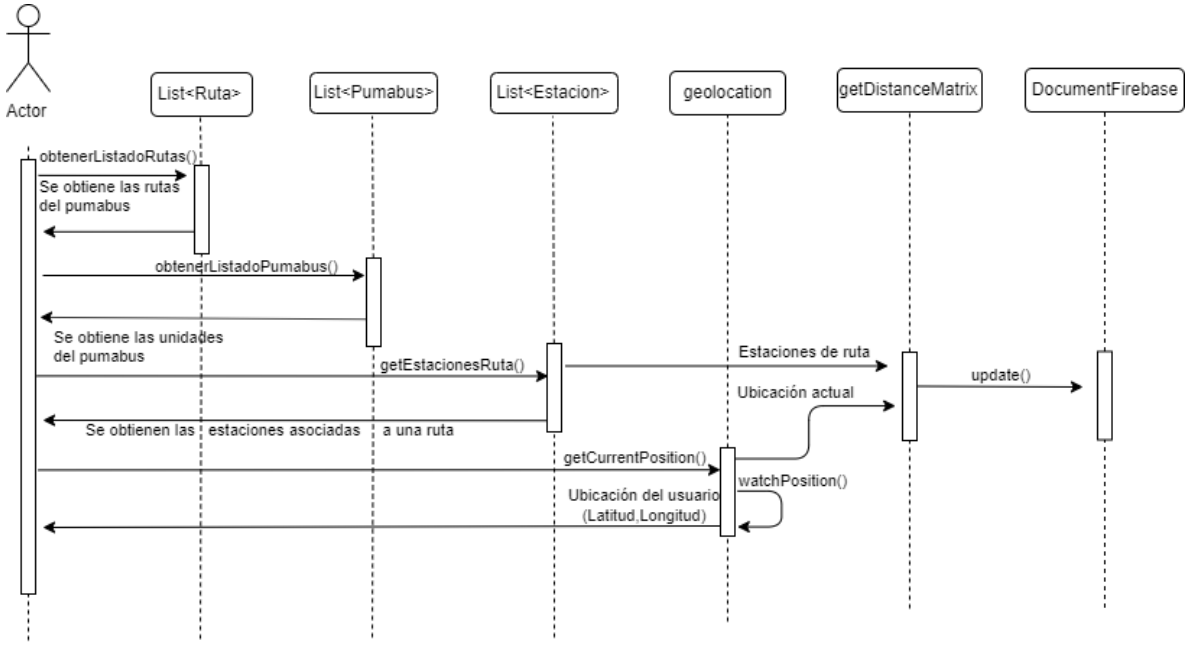


Figura 4.12 Diagrama de secuencia para la aplicación de conductores.

La figura 4.12 ilustra el diagrama de secuencia para la aplicación de los conductores del Pumabús. A continuación, se describen los puntos importantes de dicho diagrama:

- En primer lugar se obtienen un listado tanto de las rutas del Pumabús como de las unidades de transporte. Esta información es devuelta al usuario.
- Una vez que el usuario (conductor) escoge tanto una ruta del Pumabús como una unidad, se obtienen las estaciones de dicha ruta.
- Al mismo tiempo, se inicia la obtención de la ubicación del conductor, al presentar un cambio su ubicación es actualizada. Dicha ubicación es presentada al usuario.
- Con la ubicación del conductor y la estación próxima, se realiza el cálculo del tiempo aproximado de llegada. Para este cálculo se utiliza la API de Google Maps.
- Finalmente, la ubicación del conductor y el tiempo aproximado de llegada es escrito en la Base de datos de Firebase en el nodo respectivo de la ruta seleccionada.

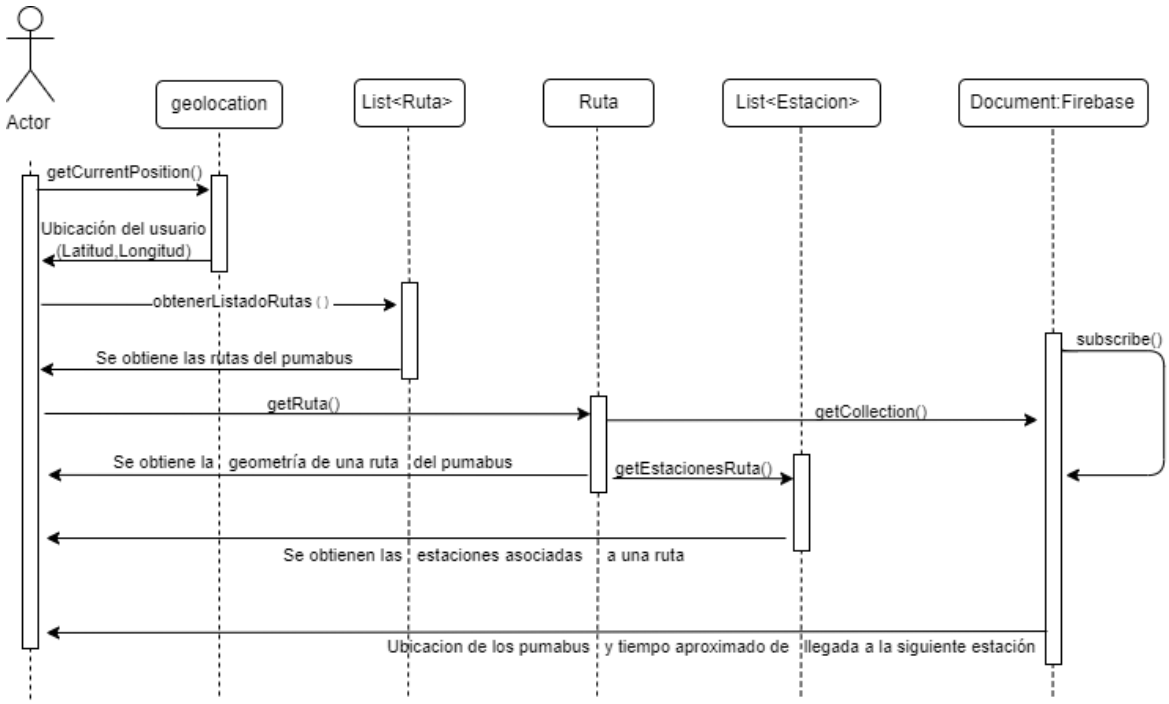


Figura 4.13 Diagrama de secuencia para el módulo de Pumabus en la aplicación de usuarios.

En la figura 4.13 se ilustra el diagrama de secuencia para el módulo de Pumabús en la aplicación de los usuarios, A continuación se describen los puntos de dicho diagrama:

- El diagrama inicia con la obtención de la ubicación actual del usuario, estos datos son presentados al usuario.
- Por otro lado, se obtiene un listado de las rutas del Pumabús y se presentan al usuario.
- Cuando se selecciona una ruta, se obtiene la geometría de la ruta y sus estaciones.
- Con el identificador de la ruta se realiza una consulta a Firebase con la finalidad de obtener los datos de los conductores que están en dicha ruta, se obtendrá la ubicación y el tiempo aproximado de llegada a la siguiente estación.

La implementación de los procesos mencionados anteriormente se explicara a detalle en el siguiente capítulo.

4.6 Modelo Entidad Relación

Un modelo entidad relación permite la ilustración de entidades y sus atributos. Estos componentes se obtienen de los requerimientos planteados por el cliente. Este diagrama es el primer paso para crear la BD de datos del sistema y facilita la realización del diagrama entidad relación (DER).

Un modelo entidad relación se basa en tres componentes principales: Entidades, Atributos y relaciones.

- Entidad: Objeto al cual se le puede obtener información (atributos). La representación de entidades es un cuadrado en el que contiene el nombre de la entidad (Figura 4.14).

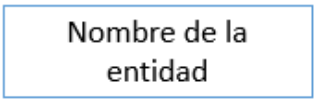
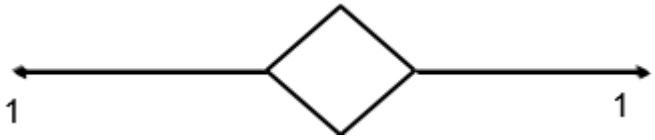
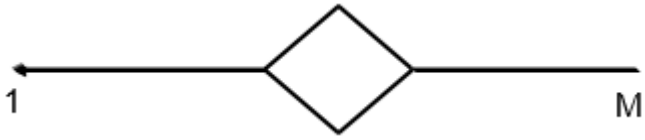
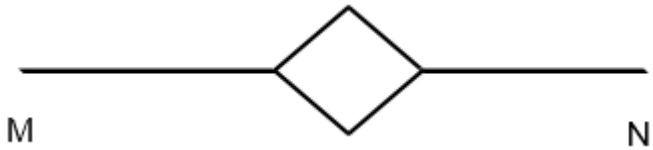


Figura 4.14 Representación de una entidad

- Atributo: Un atributo es una propiedad asociada a una entidad. Los atributos se representan con un círculo u óvalo (Figura 4.15)

Figura 4.15 Representación de un atributo

- Relación: Elemento que permite relacionar entidades. Las relaciones se representan con una línea que conecta a dos o más entidades.

Tipo de relación	Cardinalidad	Representación
Uno a uno (one-to-one)	1:1	
Uno a muchos (one-to-many)	1:M	
Muchos a muchos (Many-to-many)	M:N	

En la figura 4.16 se presenta el modelo entidad relación del sistema Movilidad UNAM.

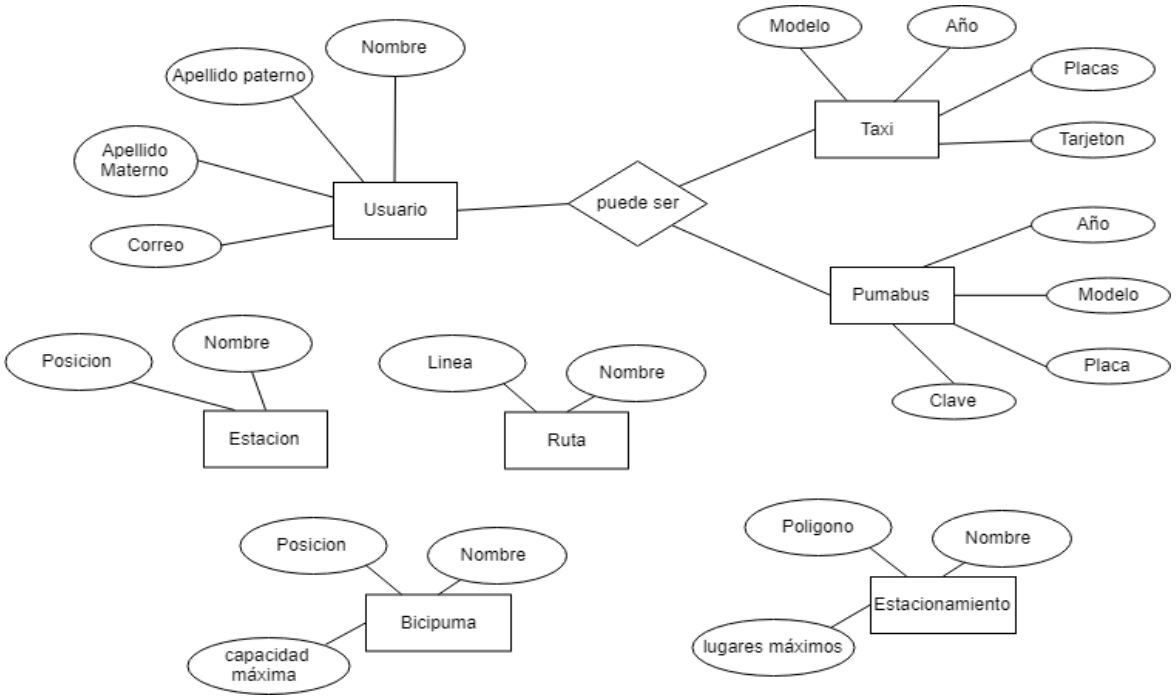


Figura 4.16 Diagrama entidad relación para el sistema integral.

4.7 Diccionario de datos

En un diccionario de datos se presentan las características lógicas de los datos que serán utilizados en el sistema a desarrollar, cuando se realiza el diccionario de datos se especifican los datos de manera precisa y rigurosa, especificando su nombre, una breve descripción, el tipo de dato, su longitud, entre otros aspectos.

Los datos que se presentan en el DD suelen ser obtenidos del modelo de entidad relación. Por cada entidad se cuenta con ciertos atributos que serán presentados en el DD.

A continuación, se presenta un diccionario de datos para las entidades definidas en la Figura 4.16

Nombre de la tabla: usuario							
Descripción: Tabla que almacenará los datos de los usuarios.							
Fecha de creación: 02/10/2018				**Fecha de modificación:** 03/10/2018			
Orden	**Atributo**	**Descripción**	**Llave**	**Tipo**	**Longitud**	**Único**	**Not null**
1	usuario_id	Identificador único y consecutivo del usuario.	PK	Integer		Si	Si
2	usu_password	Contraseña única del usuario.				No	Si
3	usu_correo	Correo electrónico único del usuario.		Varchar	50	Si	Si
4	usu_ap_materno	Apellido materno del usuario.		Varchar	50	No	Si
5	usu_ap_paterno	Apellido paterno del usuario.		Varchar	50	No	Si
6	usu_nombre	Nombre del usuario.		Varchar	50	No	Si

Figura 4.17 Entidad usuario.

Nombre de la tabla: taxi							
Descripción: Tabla para almacenar los datos de los vehículos que serán taxis.							
Fecha de creación: 02/10/2018				**Fecha de modificación:** 03/10/2018			
Orden	**Atributo**	**Descripción**	**Llave**	**Tipo**	**Longitud**	**Único**	**Not null**
1	taxi_id	Identificador único y consecutivo de cada unidad de taxi.	PK	Integer		Si	Si
2	taxi_fecha_registro	Fecha cuando se registró la unidad de taxi.		Date		No	Si
3	taxi_anio	Año del automóvil.		Date		No	Si
4	taxi_modelo	Modelo del automóvil.		Varchar	50	No	Si
5	taxi_tarjeton	Número de tarjetón proporcionado por el gobierno de la CDMX.		Varchar	50	Si	Si
6	taxi_num_placa	Placas del automóvil.		varchar	50	Si	Si

Figura 4.18 Entidad Taxi.

Nombre de la tabla: pumabus							
Descripción: Tabla para almacenar los datos de los vehículos que pertenecen al sistema de transporte pumabús.							
Fecha de creación: 02/10/2018				**Fecha de modificación:** 03/10/2018			
Orden	**Atributo**	**Descripción**	**Llave**	**Tipo**	**Longitud**	**Único**	**Not null**
1	pumabus_id	Identificador único y consecutivo por cada pumabús.	PK	Integer		Si	Si
2	pumabus_clave	Clave única por cada pumabús.		Varchar	50	Si	Si
3	pumabus_fecha_reg	Fecha cuando se registró la unidad de pumabús.		Date		No	Si
4	pumabus_anio	Año del automóvil.		Date		No	Si
5	pumabus_modelo	Modelo del automóvil.		Varchar	50	No	Si
6	pumabus_num_placa	Placas del automóvil.		varchar	50	Si	Si

Figura 4.19 Entidad Pumabus.

Nombre de la tabla: ruta							
Descripción: Tabla para almacenar las rutas disponibles del sistema de transporte pumabús.							
Fecha de creación: 02/10/2018				**Fecha de modificación:** 03/10/2018			
Orden	**Atributo**	**Descripción**	**Llave**	**Tipo**	**Longitud**	**Único**	**Not null**
1	ruta_id	Identificador único y consecutivo para cada ruta.	PK	Integer		Si	Si
2	ruta_nombre	Nombre de la ruta.		Varchar	50	Si	Si
3	geo	Geometría que representa la trayectoria de la ruta(polyline).		geometry		No	Si

Figura 4.20 Entidad Ruta.

Nombre de la tabla: estacion							
Descripción: Tabla para almacenar las estaciones que pertenecen al sistema de transporte pumabús.							
Fecha de creación: 02/10/2018				**Fecha de modificación:** 03/10/2018			
Orden	**Atributo**	**Descripción**	**Llave**	**Tipo**	**Longitud**	**Único**	**Not null**
1	estacion_id	Identificador único y consecutivo para cada estación.	PK	Integer		Si	Si
2	estacion_nombre	Nombre de la estación.		Varchar	50	No	Si
3	geo	Geometría que representa la ubicación de la estación (point).		geometry		No	Si

Figura 4.21 Entidad Estacion.

Nombre de la tabla: estacionamiento							
Descripción: Tabla para almacenar los estacionamientos disponibles dentro de Ciudad Universitaria.							
Fecha de creación: 02/10/2018				**Fecha de modificación:** 03/10/2018			
Orden	**Atributo**	**Descripción**	**Llave**	**Tipo**	**Longitud**	**Único**	**Not null**
1	estacionamiento_id	Identificador único y consecutivo para cada estacionamiento.	PK	Integer		Si	Si
2	estacionamiento_cap_max	Capacidad máxima de vehículos en el estacionamiento.		Integer	50	No	Si
3	estacionamiento_nombre	Nombre del estacionamiento.		Varchar		Si	Si
4	geo	Geometría que representa el área de cada estacionamiento(polygon)		geometry		No	Si

Figura 4.22 Entidad Estacionamiento.

Nombre de la tabla: bicupuma							
Descripción: Tabla para almacenar los centros de préstamo para las bicipuma dentro de Ciudad Universitaria.							
Fecha de creación: 02/10/2018				**Fecha de modificación:** 03/10/2018			
Orden	**Atributo**	**Descripción**	**Llave**	**Tipo**	**Longitud**	**Único**	**Not null**
1	bicipuma_id	Identificador único y consecutivo para cada bicipuma.	PK	Integer		Si	Si
2	bicipuma_cap_max	Capacidad máxima de bicipuma por estación.		Integer		No	Si
3	bicipuma_nombre	Nombre de la estación.		Varchar	50	Si	Si
4	geo	Geometría que representa la ubicación de la estación para las bicipuma(point)		geometry		No	Si

Figura 4.23 Entidad Bicupuma.

Nombre de la tabla: estacion_ruta							
Descripción: Tabla para almacenar la relación entre las estaciones y rutas.							
Fecha de creación: 02/10/2018				**Fecha de modificación:** 03/10/2018			
Orden	**Atributo**	**Descripción**	**Llave**	**Tipo**	**Longitud**	**Único**	**Not null**
1	estacion_ruta_id	Identificador único y consecutivo para cada registro de la tabla.	PK	Integer		Si	Si
2	estacion_id	Identificador de la estación.		Integer		No	Si
3	ruta_id	Identificador de la ruta.		Integer		No	Si

Figura 4.24 Entidad estacion_ruta

4.8 Modelos relacionales.

Los modelos relacionales son utilizados para diseñar base de datos. Muestran la relación que existe entre las entidades y los atributos asociados a cada una de las entidades.

Los modelos relacionales son utilizados para modelar y diseñar bases de datos basados en las reglas de negocio de una manera lógica. Por otro lado, si ya se cuenta con una BD creada para un sistema, los modelos relacionales son utilizados para hallar errores y resolver problemas de lógica o implementación.

Los modelos relacionales cuentan con elementos principales para su elaboración:

- Entidad (Figura 4.25)

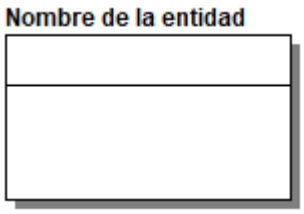


Figura 4.25 Representación de una entidad en el diagrama ER

- Atributo

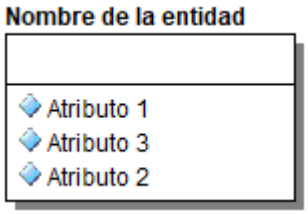


Figura 4.26 Representación de atributos en el diagrama ER

- Relación

Para el modelo relacional existen dos tipos de relaciones: las relaciones no identificativas y las relaciones identificativas.

Las relaciones no identificativas hacen referencia a una entidad que no depende de la existencia de otra entidad, ya que cada entidad tiene su propia clave principal (llave primaria), sin embargo, la llave primaria de una entidad aparecerá como atributo de la segunda entidad. Normalmente son representadas con una línea punteada (Figura 4.27).

Figura 4.27 Representación de una relación no identificativa

En la Figura 4.28 se puede observar un ejemplo de una relación no identificativa: Se tiene dos entidades llamadas `Estacion` y `Estacion_ruta`. La llave primaria (`estacion_id`) de la entidad `Estacion` es un atributo de la entidad `Estacion_ruta`, nótese que la entidad `Estacion_ruta` cuenta con su propia llave primaria (`estacion_ruta_id`).

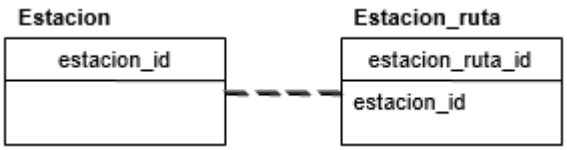


Figura 4.28 Ejemplo de una relación no identificativa

Por otro lado, las relaciones identificativas hacen referencia a una entidad (conocida como entidad débil) que depende de la existencia de otra entidad (conocida como entidad fuerte). La llave primaria de la entidad fuerte se propaga como llave primaria a la entidad débil. Son representadas con una línea continua (Figura 4.29).

Figura 4.29 Representación de una relación identificativa

La figura 4.30 muestra un ejemplo de una relación identificativa en donde las entidades `Alumno` y `Dirección` tienen el mismo atributo como llave primaria (`alumno_id`).

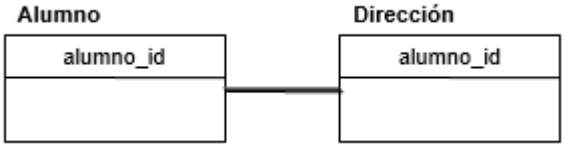


Figura 4.30 Ejemplo de una relación identificativa

Aunado a los tipos de relaciones que se mencionaron anteriormente, existen notaciones para representar dichas relaciones, a continuación se presentan las notaciones Crows Foot e IDEF1X:

Notación Crows Foot (Figura 4.31).

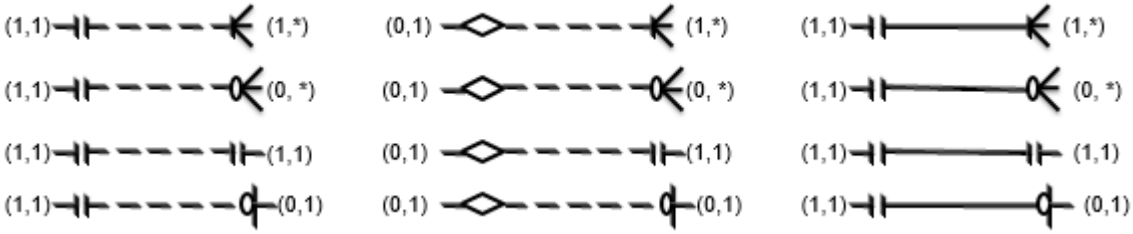


Figura 4.31 Relaciones con la notación Crows Foot

Notación IDEF1X (Figura 4.32)

Figura 4.32 Relaciones con la notación IDEF1X

Los modelos relacionales son independiente del manejador que se defina utilizar para la creación de la base de datos. Para el sistema se eligió utilizar PostgreSQL.

En el capítulo No. 2 (sección 2.1.6) se mencionaron algunas diferencias y características entre el manejador de base de datos PostgreSQL y su extensión espacial PostGIS.

4.9 Modelo relacional sin extensión espacial. (PostgreSQL)

Con base a los requerimientos obtenidos en el capítulo anterior y con los conceptos mencionados en el presente capítulo se presenta a continuación el modelo relacional (Figura 4.33) utilizando el manejador PostgreSQL y los tipos de datos proporcionados por dicho manejador.

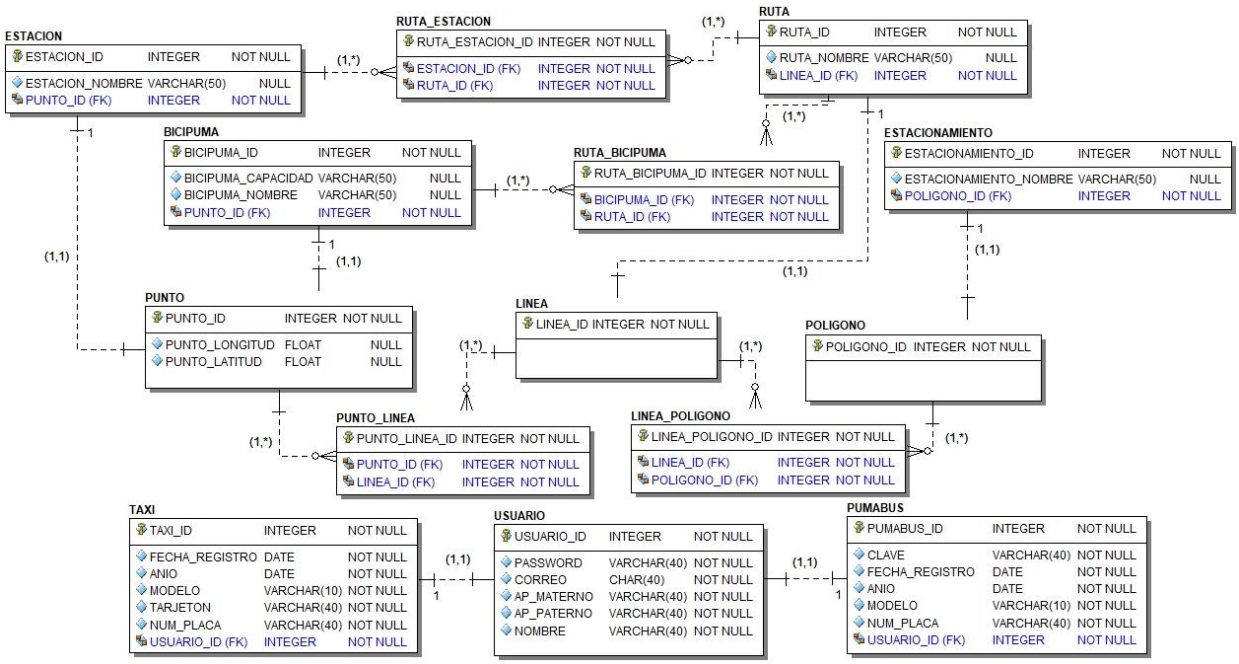


Figura 4.33 Modelo relacional del sistema Movilidad UNAM con PostgreSQL

Como se logra observar el DER de la figura 4.33 es muy complejo, ya que se hace manejo de varias tablas para representar objetos geométricos como son: polígonos, líneas y puntos. Para la representación geográfica del sistema se utilizarían las siguientes tablas:

- Punto
- Linea
- Punto_linea
- Poligono
- Linea_poligono
- Bicipuma
- Estacion
- Ruta
- Ruta_estacion
- Ruta_bicipuma
- Estacionamiento

Para lograr entender la parte geográfica del DER se mencionan algunos puntos para su explicación:

- La geometría básica es el punto (tabla punto), por cada ítem es necesario saber su latitud y longitud.

- Cada punto puede ser una estación del pumabús o en dado caso un centro de bicipuma. Es por eso que se tiene la relación entre estas tablas.

- Una línea puede estar formada a partir de dos puntos o muchos (cabe la posibilidad de infinito), además, cada punto existente puede pertenecer a más de una línea, por esta razón se crea una tabla intermedia para establecer la relación muchos a muchos de las tablas punto y línea.

- Las rutas del pumabús son representadas por una línea continua, es por eso que se asocian las tablas línea y ruta.

- Para la representación de un estacionamiento se utilizarán polígonos, ahora bien, un polígono puede estar formado mínimo por tres líneas o infinitas (se debe de tomar en cuenta que entre más líneas se necesiten para representar un polígono más exacto será la figura o el polígono a formar), es por eso que existe una relación muchos a muchos y la necesidad de crear una tabla intermedia.

- Ahora bien, cada ruta del Pumabús (líneas) cuenta con un número finito de estaciones (puntos), lo mismo ocurre con la ruta de las Bicipuma, para poder saber que estaciones se encuentran dada una ruta del Pumabús se creó la tabla `ruta_estacion`.

4.10 Modelo relacional con extensión espacial (PostGIS)

Para entender un poco mejor el manejo de la información espacial, se hablará un poco de OGC (Open Geospacial Consortium). La OGC es la encargada de proveer estándares para el manejo de información geográfica e intercambio de dicha información entre sistemas.

Algunos de los estándares proporcionados por la OGC se mencionan a continuación:

- Geography Markup Language(GML): Estándar basado en XML para el manejo y guardado de información geográfica.

- Web Mapping Service(WMS): Proporciona operaciones para poder obtener mapas como imágenes y obtener información sobre el mapa.

- Web Feature Service(WFS): Permite acceder a los elementos de un componente geográfico (rio, ciudad, lago, etc.) en forma vectorial.

- Web Coverage Service(WCS): Permite consultar atributos almacenados en cada píxel.

- Keyhole Markup Language(KML): Es el estándar basado en XML para la representación de datos geométricos en tres dimensiones.

Las bases de datos espaciales permiten almacenar y manipular los objetos espaciales como cualquier otro objeto en una base de datos. Los objetos de datos espaciales pueden ser un punto, línea y polígono. Además, los manejadores de BD espaciales proveen una serie de funciones en SQL, que permiten al desarrollador a consultar propiedades y relaciones espaciales.

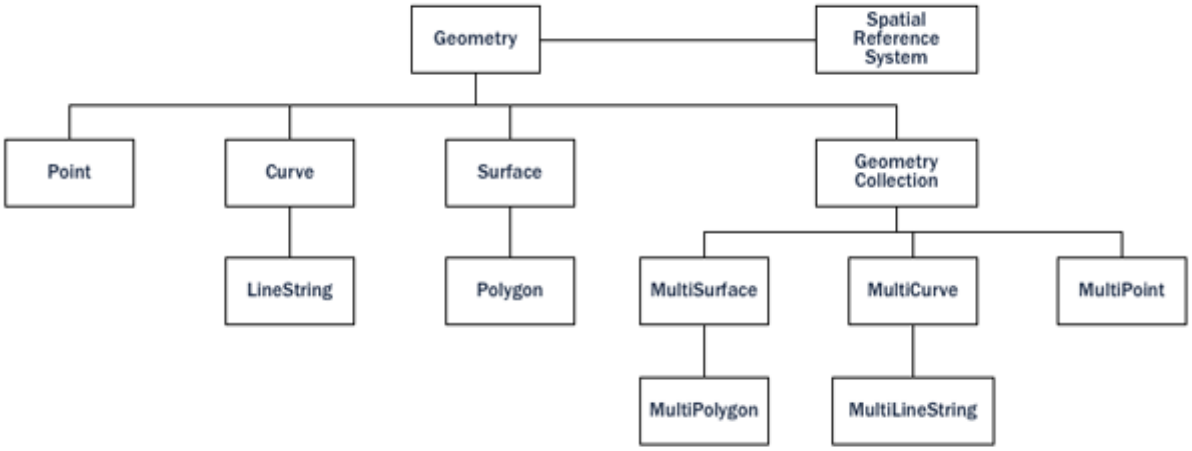


Figura 4.34 Tipos de datos PostGIS

En la figura 4.34 se muestra un diagrama que permite ver los tipos de datos soportados en PostGIS, estos tipos de datos están organizados por una jerarquía. Cada subtipo hereda los atributos y comportamiento del supertipo.

- Point: El tipo de dato Punto representa una ubicación, es representada con coordenadas, por ejemplo: POINT(1 1).

- LineString: El tipo de datos LineString está formada por dos o más objetos Point formando una línea, este tipo de objeto puede ser cerrada: comienza y termina en el mismo punto, o simple: si no se cruza ni se toca, por ejemplo:
  LINESTRING(0 0, 1 1, 2 1,2 2).

- Polygon: El tipo de dato Polygon representan un área, el límite del Polygon es un LineString simple y cerrada. Por ejemplo,
  POLYGON((0 0,1 0,1 1,0 1,0 0)).

Para identificar la gran diferencia que se tendrá al utilizar la extensión espacial PostGIS se presenta el DER (Figura 4.35) considerando el uso de dicha herramienta:

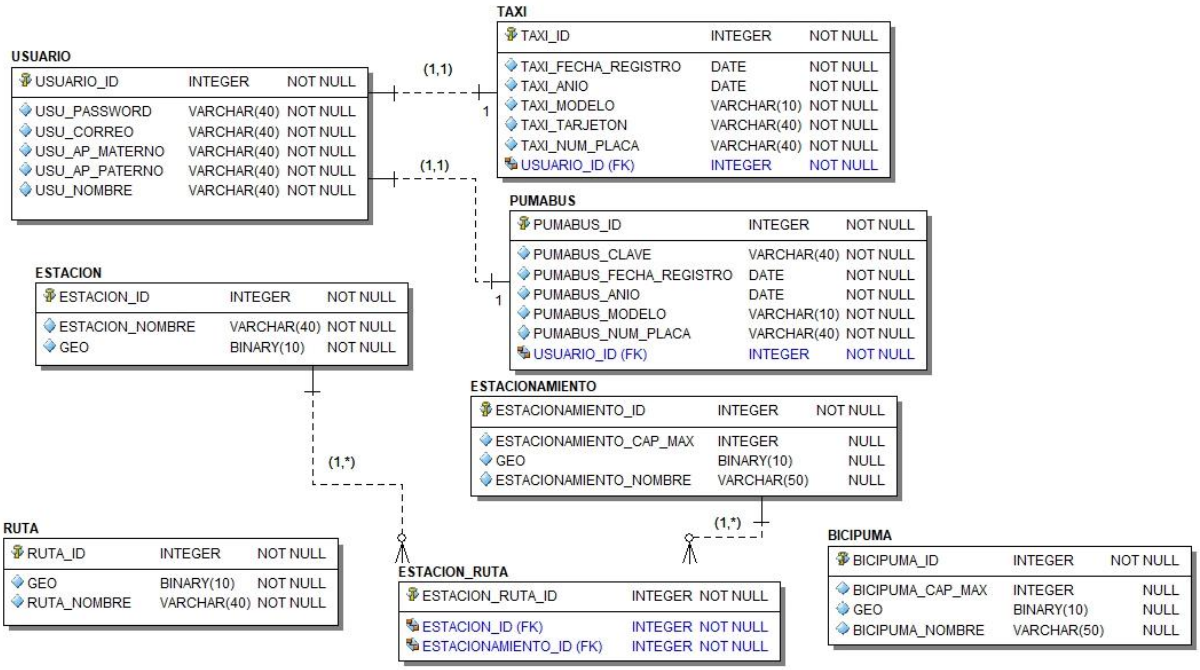


Figura 4.35 Modelo relacional del sistema Movilidad UNAM con PostGIS

4.11 Diferencias entre PostgreSQL y PostGis

Como se logra observar en los diagramas realizados con las herramientas PostgreSQL y PostGIS existe una diferencia en el tamaño y complejidad entre sí mismos. A continuación, se mostrará la diferencia para la elaboración de los elementos geométricos utilizados en la aplicación (estación, ruta, estacionamiento)

- Estación - Punto

  Cuando se realiza el diagrama con PostgreSQL y se pretende representar las estaciones solo basta la utilización de un punto, como podemos observar solo se necesitarán dos tablas:

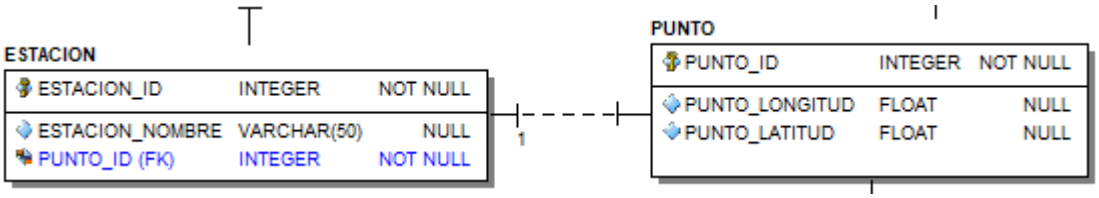


Figura 4.36 Representación de una estación con PostgreSQL

Por el lado contrario, el diagrama con PostGIS solo será necesaria una tabla con la columna GEO en la cual se podrá guardar los datos binarios que representen un punto.

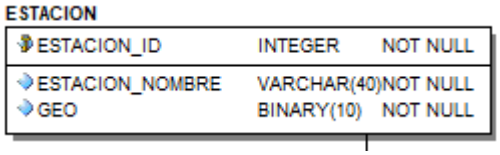


Figura 4.37 Representación de una estación con PostGIS

- Ruta – Línea

Ahora bien, para la representación de las rutas el diagrama con PostgreSQL se vuelve un poco más complejo aunque aún es entendible, como se puede observar en la Figura 4.38.

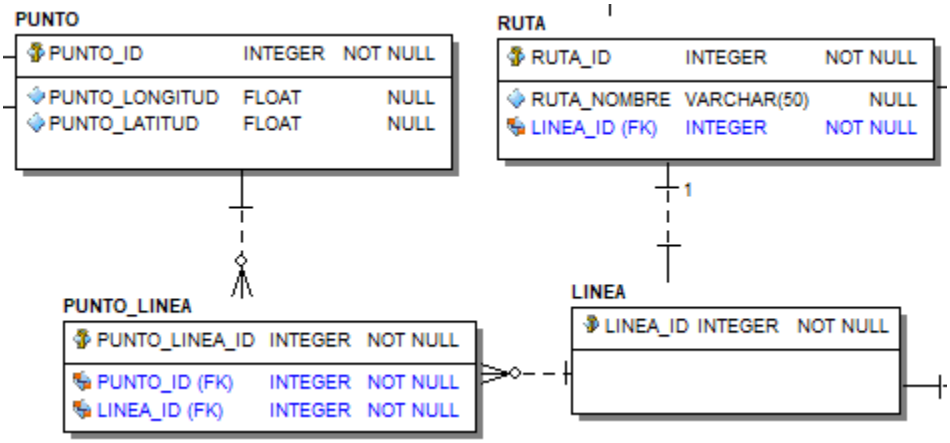


Figura 4.38 Representación de las rutas del Pumabus con PostgreSQL

Como en el caso de la representación del punto con PostGIS, al implementar la ruta solo bastará con una tabla que contenga la columna GEO para representar la geometría como se observa en la Figura 4.39

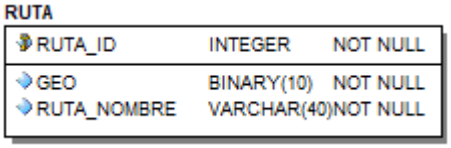


Figura 4.39 Representación de las rutas del Pumabus con PostGIS

- Estacionamiento – Polígono

  Para el caso de los estacionamientos los cuales necesitan ser representados por polígonos el diagrama contiene un número mayor de tablas, además, como podemos observar se necesitan las geometrías básicas (punto, línea) para la representación del polígono.

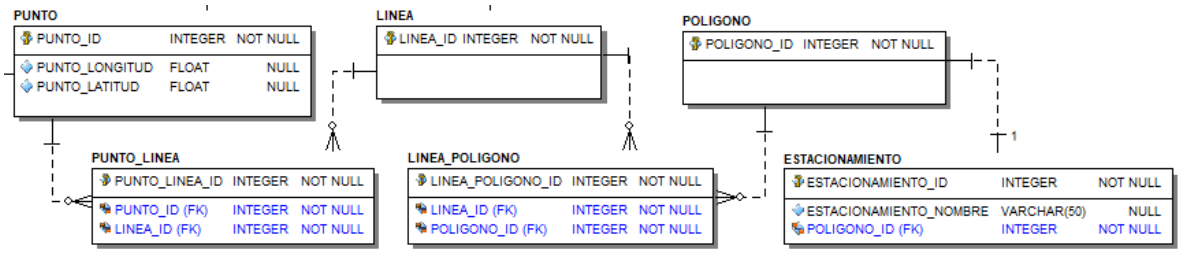


Figura 4.40. Representación de los estacionamientos con PostgreSQL

Al igual que los dos diagramas anteriores realizados con la herramienta PostGIS solo se necesitará de una tabla para el guardado de los datos binarios que representen el polígono.

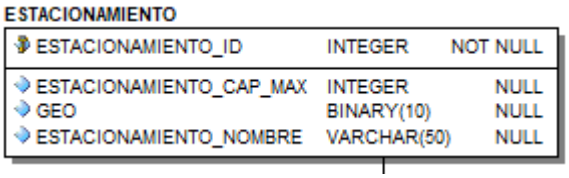


Figura 4.41 Representación de los estacionamientos con PostGIS

Características principales por destacar:

- Se reduce el número de tablas y las relaciones entre ellas.
- No se cuenta con un dato especifico como latitud y longitud, ya que la extensión espacial te brinda la posibilidad de almacenar una geometría (punto, línea, polígono,etc)
- El acceso a los datos es más eficaz y veloz.
- La columna geo es la geometría y es almacenada en datos binarios.
- PostGIS permite el uso de funciones espaciales ya definidas, por el lado contrario, si se utilizara PostgreSQL se tendrían que programar.
- Ahorro de tiempo en la ejecución de consultas gracias a la indexación espacial.
- PostGIS es compatible con OGC (Open Geospatial Consortium).

5. IMPLEMENTACIÓN

La fase de implementación en el desarrollo de software es la encargada de la elaboración del producto con base a las fases de análisis de requerimientos y diseño. La elaboración del producto deberá cumplir con los requerimientos especificados por el cliente.

En la fase de implementación prácticamente se basa en el desarrollo de código para el sistema. Sin embargo, los desarrolladores deben tomar en cuenta ciertos puntos para su desarrollo, por ejemplo:

- Definir convenciones para el nombre de variables y funciones.
- Documentar el código.
- Indentar el código dentro de estructuras de control y funciones.
- Utilizar de manera correcta la herramienta (por ejemplo si, se utiliza el patrón MVC, definir y usar adecuadamente los componentes).
- Simplificar el código.
- No repetir código o funciones que ya estén programadas.
- Probar el código realizado.

Los puntos anteriores son conocidos como “buenas prácticas de programación” las cuales pueden ser encontradas en documentos, foros o libros. Estos puntos ayudan a que el código sea entendible para otros programadores y en un futuro sea escalable.

Antes de iniciar la fase de desarrollo se requiere planear las actividades a realizar por parte del equipo, ya que ciertos elementos dependen de otros, por ejemplo, si el sistema necesita almacenar información, es factible que primero se inicie con la elaboración de la Base de Datos que con el desarrollo del sistema.

En el presente capítulo, se abordará la forma de obtener la información del módulo de Pumabús para la aplicación del usuario y se explicará todo lo necesario para cumplir con este requerimiento. A continuación se en lista la estructura del capítulo:

- Primero se explicarán puntos destacados para la instalación de PostGIS, así mismo, se incluirán algunos scripts para la creación de tablas, inserción de datos y se dará una breve explicación.
- También, se explicará cómo crear una BD NoSQL en Firebase, la estructura que tendrá dicha base para el sistema.
- Posteriormente, se explicará el procedimiento para la obtención de la información del Pumabús y el cálculo de tiempo aproximado de llegada.

5.1 Base de datos SQL.

Como se ha mencionado en capítulos anteriores, se utilizará una base de datos con extensión espacial. Para su instalación partiremos en que tenemos como sistema operativo alguna distribución de Linux.

Como primer paso será descargar las herramientas PostgreSQL(https://www.postgresql.org/ftp/source/) y postGIS (http://postgis.net/source) de preferencia las versiones más recientes.

5.1.1 Instalación de PostgreSQL

Una vez descargados los archivos para la instalación de PostgreSQL se deberá de realizar las siguientes instrucciones:

a. Descomprimir el archivo `tar.gz`

```
tar –xvf postgresql-10.5/pgsql
```

b. Para verificar que nuestro equipo cuente con lo necesario para compilar los archivos se necesitara ejecutar la siguiente instrucción

```
./configure –prefix=/opt/postgresql-10.5/pgsql
```

c. Iniciar la compilación de los archivos

```
make
```

d. Instalación del software

```
sudo make install
```

e. Creación del usuario postgres

```
sudo adduser postgres
sudo passwd postgres
```

f. Iniciar y detener la instancia

```
sudo –l postgres
Pg_ctl –D /opt/postgresql-10.5/pgsql/data –l
/home/postgres/logs/postgreSQL.log start 2>&1 &

Pg_ctl –D /opt/postgresql-10.5/pgsql/data stop
```

Hasta este momento solo se ha instalado PostgreSQL sin su extensión espacial, el siguiente punto procederemos a instalar dicha herramienta.

5.1.2 Instalación de PostGIS

Para la instalación de la extensión espacial de PostGIS, seguir los siguientes pasos:

a. Descomprimir el archivo `tar.gz`

```
tar –xvf
```

b. Cambiarse al directorio generado y ejecutar

```
./configure  --with-
pgconfig=/opt/postgresql/10.5/pgsql/bin/pg_config
```

c. Compilar e instalar

```
make
sudo make install
```

5.1.3 Instrucciones DDL y DML.

Al utilizar una Base de datos dos de los principales conceptos que se manejan son DDL (Lenguaje de Definición y datos) permite definir las estructuras que almacenarán los datos y DML (Lenguaje de Manipulación de Datos) permite consultar o modificar los datos que se encuentran almacenados en la base de datos.

A continuación, se presentan las instrucciones necesarias para la creación de tablas con su extensión espacial e inserciones a la base de datos con información espacial:

```
create table estacion(
   estacion_id numeric(10,0) constraint estacion_pk primary key,
   nombre varchar(100) not null
   );

create table ruta(
   ruta_id numeric(10,0) constraint ruta_pk primary key,
   ruta_nombre varchar(100) not null
   );
```

Para agregar la extensión espacial a nuestra base de datos y por ende a las tablas se deberán ejecutar las siguientes líneas:

```
select addGeometryColumn
('public','estacion','geo',0,'POINT',2);
```

La función `addGeometryColumn` agrega una columna para el uso de las geometrías, dicha función recibe los siguientes parámetros:

- `public`: nombre del esquema de la tabla
- `estacion`: nombre de la tabla
- `geo`: nombre de la columna.
- 0: número entero que hace referencia al SRID (valor entero que se encuentra en la tabla spatial_ref_sys).
- `POINT`: Cadena que corresponde al tipo de geometría (`POINT`, `LINESTRING`, `POLYGON`, etc.).
- 2: dimensión para la geometría.

Una vez ejecutadas las líneas de código, la tabla `estacion` cuenta con una columna llamada `geo` que permite el guardado de datos espaciales, para poder llevar a cabo una inserción de datos se utilizará la función `st_geomFromText` como se muestra a continuación:

```
Insert into estacion (nombre, geo)
values ('Universidad', st_geomFromText ('point (19.3241822 -
99.1748808)'));
```

Una vez que se tiene la estructura de la base de datos, se podrá continuar con la elaboración de los elementos que conforman el sistema. Para este caso, a continuación, se presenta una sección del Sistema Movilidad UNAM.

5.2 Base de datos NoSQL.

Las bases de datos NoSQL surgieron principalmente para el uso empresas que manejan una gran cantidad de información y así poder cubrir ciertas necesidades como escalabilidad, rendimiento y mantenimiento. En la actualidad se pueden encontrar 5 categorías de las bases de datos NoSQL:

- Bases de datos Clave – Valor: Se estructuran almacenando información con tuplas con una clave y un valor. Los clientes o en este caso los sistemas solicitan valores a partir de una clave conocida.
  Alguna de las ventajas al utilizar este tipo de BD NoSQL es que se tendrá una alta escalabilidad y un rendimiento alto para el manejo de la información.

- Bases de datos Documentales: Para este tipo de BD NoSQL un concepto fundamental es el "documento" que se refiere a la unidad de almacenamiento. Dicho documento puede ser almacenado en diferente formato como XML o JSON. Normalmente los documentos son almacenados en colecciones que son proporcionados por el gestor de la BD.

- Base de datos orientada a Grafos: Como su nombre lo menciona, este tipo de BD NoSQL representa la información almacenada como si fuera un grafo. La información almacenada son los nodos y la relación entre los nodos (información) son las aristas.

- Base de datos orientados a Objetos: La información es representada como un objeto (Similar a la Programación Orientada a Objetos) en la cual se definen operaciones que pueden ser ejecutadas en dichos datos.

- Base de datos orientados a Columnas: La información es estructurada en columnas a diferencia de las BD relacionales que se estructura en filas.

Firebase Realtime Database es un gestor de BD NoSQL proporcionado por Google el cual almacena la información en formato JSON. Como se mencionó anteriormente, este tipo de BD es de tipo documental. Una de las principales características de Firebase Realtime Database es la sincronización en tiempo real con los clientes conectados a la BD de datos: cuando un nodo en la BD recibe alguna modificación en su valor automáticamente se realiza una actualización a los clientes interesados en dicho nodo (Figura 5.1).

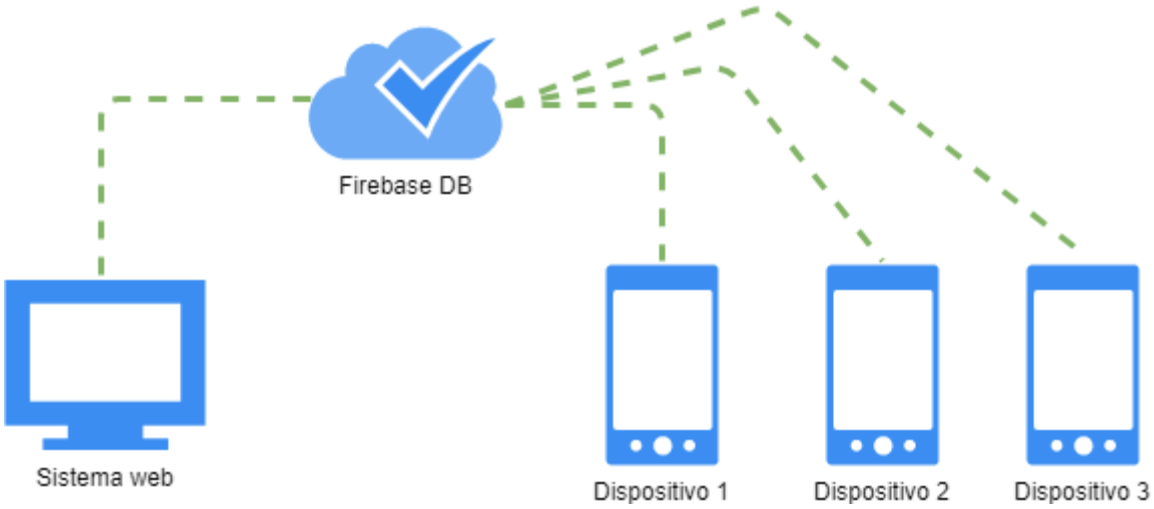


Figura 5.1 Actualización de un valor en Firebase.

Firebase Realtime Database puede ser implementada para aplicaciones móviles ya sea iOS o Android, Aplicaciones WEB, API REST, configuración para el lenguaje C++ y para Unity. En la figura 5.2 podemos identificar cuatro características de esta herramienta.

Funciones	Descripción
Tiempo real	Firebase Realtime Database utiliza la sincronización de datos (Cada vez que cambian los datos, los dispositivos conectados reciben esa actualización en milisegundos).
Sin conexión	Cuando un dispositivo se encuentra sin conexión los datos se almacenan, una vez reestablecida la conexión el dispositivo recibe los cambios que faltaban o que se llevaron a cabo cuando no se contaba con conexión.
Acceso desde dispositivos cliente	No es necesario un servidor de aplicaciones para el manejo de la información ya que se puede realizar desde una aplicación móvil o navegador web.

Funciones	Descripción
Escalamiento en varias bases de datos	Dentro de un mismo proyecto es posible crear múltiples instancias para el manejo de información de algún sistema.

Figura 5.2 Características de Firebase Realtime Database

5.2.1 Estructura de Firebase Realtime DataBase

El establecer la estructura para la BD de Firebase es de suma importancia ya que este punto dependerá del funcionamiento adecuado del sistema, para la construcción de la estructura se pueden considerar tres puntos importantes:

- Evitar la anidación de datos: Firebase permite una anidación de 32 niveles, sin embargo, lo más adecuado es que no se tenga un número alto de anidación, ya que cuando se lee un nodo se obtiene la información de todos los nodos secundarios y eso podría causar un bajo rendimiento en el sistema.

- Compactar las estructuras de datos: Tratar de dividir la información para que su obtención se haga de manera independiente, esto es conocido como normalizar.

- Creación de datos escalables: Esta característica es utilizada cuando se construye una aplicación y que su información es de uso bidireccional, para este caso se permite la duplicidad de datos con la finalidad de obtener de una forma más directa la información y no tener que realizar consultas complejas a la BD.

Como se ha mencionado anteriormente, los datos en Firebase Realtime Database se almacenan en objetos JSON. A diferencia de las BD de datos SQL, en Firebase no existen las tablas.

Normalmente la estructura de Firebase es asimilada a un árbol y cuando se le agregan datos a dicho árbol se agrega un nodo a nuestro objeto JSON con una clave asociada (Figura 5.3).

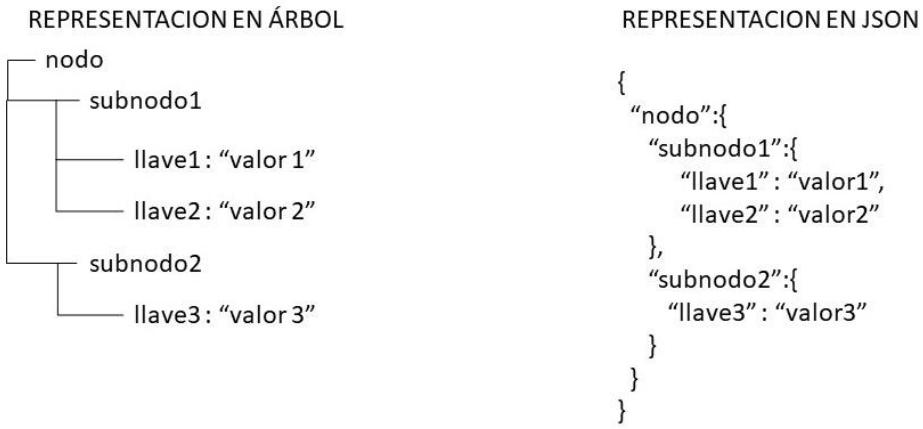


Figura 5.3 Representación de los datos en Firebase

Una vez entendido la estructura que maneja la BD de Firebase, en la Figura 5.4 se define la estructura que se utilizará para guardar la ubicación de los Pumabus.

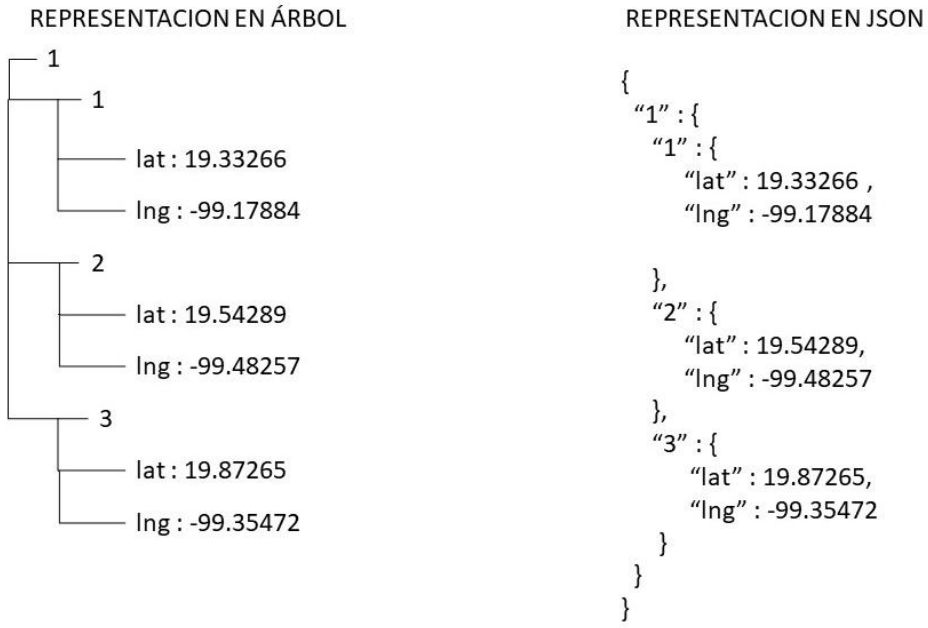


Figura 5.4 Estructura de la BD en Firebase para los Pumabús

El nodo principal con valor de uno, representa la ruta uno del Pumabús, los subnodos con valor de uno, dos y tres, son los id que tiene cada unidad del Pumabús. Dentro de estos submodulos se tienen tuplas con llave lat y lng y sus valores respectivamente.

Cuando un conductor este en la ruta número uno, la aplicación apuntará al nodo principal seguido del id de su unidad del Pumabús, por ejemplo, si el id del Pumabús es tres, apunta al subnodo tres y actualiza los valores de lat y lng cada que la aplicación del conductor detecte un cambio en su localización.

Ahora, cada que Firebase detecte que se realizó cambio en alguno de los valores manda dicho cambio a todos los dispositivos que estén conectados a su nodo, en este caso todos los usuarios que estén revisando la ruta uno recibirán el cambio de la localización del Pumabús con id tres.

5.3 Obtención de la ubicación de los Pumabús a mostrar en la aplicación de los usuarios.

Después de haber construido tanto la BD espacial como la BD NoSQL para la manipulación de la información, es posible empezar con el desarrollo de los componentes que conforman el sistema.

Para este apartado, se explicará cómo se obtienen los datos de los Pumabús para ser mostrados en la aplicación del usuario, aquí se verán aspectos como el uso de los web services y la conexión a la BD NoSQL en Firebase. Para el entendimiento de esta funcionalidad se dividirá en las siguientes secciones:

- Obtención de los nombres de las rutas del Pumabús.
- Obtener y mostrar en la aplicación la ubicación del usuario.
- Obtención de la geometría de la ruta seleccionada.
- Obtención de las estaciones de una ruta del Pumabús.
- Obtención de la ubicación de las unidades del Pumabús que circulan sobre una ruta del Pumabús.
- Obtener el tiempo aproximado de llegada de un Pumabús a la siguiente estación.

5.3.1 Obtención de los nombres de las rutas del Pumabús.

En la figura 5.5 se muestra el menú de la aplicación móvil para los usuarios, como se logra observar se cuentan con los diferentes módulos, entre ellos está el módulo de Pumabús.

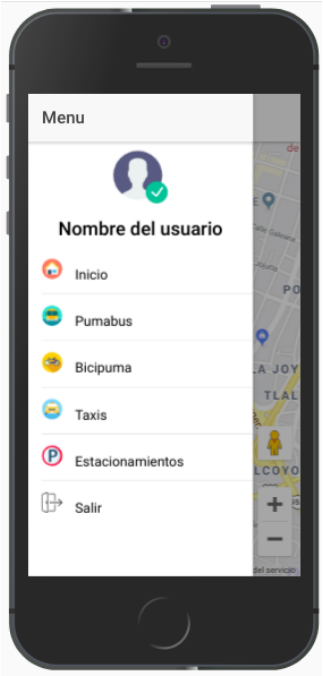


Figura 5.5 Menú de la aplicación móvil para usuarios

Al realizar clic en el apartado "Pumabús" del menú, se ejecuta la función `ionViewDidLoad()`, como se muestra a continuación:

```
ionViewDidLoad() {
  this.opcionMenuSeleccionado(this.opcion);
}
```

Al cargar por completo la vista se ejecuta la función `opcionMenuSeleccionado()` la cual se encarga de procesar la opción del módulo escogido por el usuario (para este caso: Pumabús). Se tienen múltiples casos para los diferentes módulos, en el módulo de Pumabús se define iconos para las unidades del Pumabús como para las estaciones los cuales serán presentados en la aplicación móvil:

```
opcionMenuSeleccionado(elemento){
   switch (elemento) {
```

```
    case INICIO:
      this.icono = RUTA_IMG + IMG_ESTACION_CERCANA;
      this.obtieneEstacionesCercanas();
      break;

    case PUMABUS:
      this.icono = RUTA_IMG + IMG_ESTACION_PUMA;
      this.iconoCamion = RUTA_IMG + IMG_CAMION_PUMABUS;
      this.obtieneRutas();
      break;

    /*Opciones para los otros módulos de la aplicación*/
      }
    }
```

Dentro del apartado Pumabús se ejecuta la función `obtieneRutas()`, el cuerpo de dicha función se muestra a continuación:

```
obtieneRutas(){
  this._ruta.obtenerListadoRutas().subscribe(data => {
    this.rutas = data;
  });
}
```

La variable `this._ruta` es una instancia del servicio `RutaService` del archivo `src/providers/ruta/ruta.ts`, gracias a esto, esta variable puede ejecutar las funciones definidas en el servicio mencionado anteriormente.

La función `obtenerListadoRutas` realiza una petición http al servicio web, como parámetros debe de recibir una URL, la definición de la función

`obtenerListadoRutas ()` se presenta a continuación:

```
obtenerListadoRutas(){
  let url = URL_SERVICIOS + "/ruta/getListadoRutas";
  return this.http.post(url,{} ).map(resp => resp);
}
```

El servicio web está definido en el archivo `RestRutaController.java` del sistema Movilidad UNAM, la función `getListadoRutas()` devolverá el listado de las rutas registradas.

Como se explicó en el capítulo 4. Diseño (Figura 4.5) los controladores se comunican con las interfaces de servicio, y los servicios con la interfaz de los DAO´s para poder realizar una consulta a la BD, para simplificar el proceso no se colocarán las llamadas a las interfaces de los servicios y DAO´s:

```
/**
* Método que obtiene las rutas que seran mostradas en la
* aplicación
* @return listado de rutas
*/
@CrossOrigin(origins = "*")
@RequestMapping(value = "getListadoRutas",
  method = RequestMethod.POST)
public List<Ruta> getListadoRutas() {
  return rutaService.getListadoRutas();
}
```

Como se logra observar en el código mostrado anteriormente dentro de la función `getListadoRutas()` se invoca a la interfaz del servicio `RutaService` En la clase `RutaDaoImpl` se realiza la implementación del DAO para los Pumabús, en donde solo se obtienen el identificador y el nombre de las rutas:

```
private static final String get_listado_rutas_sql =
     "SELECT ruta_id,ruta_nombre FROM ruta
      WHERE ruta_id != 14 ORDER BY ruta_id";

@Override
public List<Ruta> getListadoRutas() {
  List<Ruta> listRuta = getJdbcTemplate().query(
       get_listado_rutas_sql,new RowMapper<Ruta>() {
         @Override
         public Ruta mapRow(ResultSet rs, int rowNum)
         throws SQLException {
            Ruta ruta = new Ruta();
           ruta.setRutaId(rs.getLong("ruta_id"));
           ruta.setRutaNombre(rs.getString("ruta_nombre"));
           return ruta;
```

```
        }
   });
  return listRuta;
}
```

La variable `listRuta` contiene un listado de las rutas del Pumabús que se han registrado, este arreglo se regresa hasta el archivo `RestRutaController.java` y a su vez los datos regresan al archivo `/src/pages/home/home.ts` y son almacenados en la variable `this.rutas` que fue en donde se realizó la petición http.

Ahora bien, los datos ya se encuentran en el archivo de la aplicación móvil, pero hasta este momento no han sido mostrados al usuario. En el archivo `/src/pages/home/home.html` se realiza este proceso:

```
<ion-header>
  <ion-item>
    <ion-label>Rutas</ion-label>
    <ion-select placeholder="Seleccionar"
                (ionChange)="cambioRuta($event)">
      <ion-option *ngFor="let ruta of rutas"
                  value={{ruta.rutaId}}>{{ruta.rutaNombre}}
      </ion-option>
    </ion-select>
  </ion-item>
</ion-header>
```

La figura 5.6 muestra el resultado del proceso explicado anteriormente.

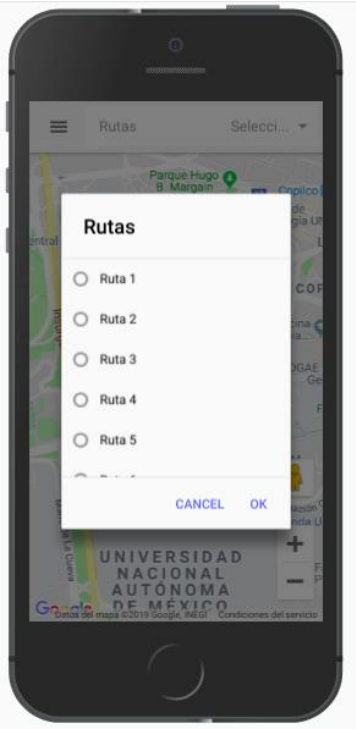


Figura 5.6 Despliegue de las rutas del Pumabus.

5.3.2 Obtener y mostrar en la aplicación la ubicación del usuario.

Un aspecto importante dentro de la aplicación es saber la ubicación del usuario, esto con la finalidad de ser mostrada en la aplicación (dentro de un mapa) y que permita su movilidad de manera rápida y eficaz, así como la toma de decisiones sobre las diferentes opciones de traslado.

En el archivo `src/pages/home/home.ts` se define un constructor. Los constructores permiten establecer valores predeterminados, en este caso el constructor manda a llamar la función `iniciarGeolocalizacion()`:

```
Constructor (private geolocation: Geolocation) {
   this.iniciarGeolocalizacion();
}
```

Como se ha mencionado en capítulos anteriores, para hacer uso de los elementos del hardware, Ionic hace uso de diferentes plugins proporcionados por Cordova, en este caso como se hará uso del GPS, es necesario importar en el constructor el elemento `Geolocation` (proporcionado por Cordova).

La función `iniciarGeolocalizacion` se encarga de obtener la ubicación del usuario, la definición de la función se presenta a continuación:

```
iniciarGeolocalizacion(){
  this.geolocation.getCurrentPosition().then((resp) => {
    this.lat = resp.coords.latitude;
    this.lng = resp.coords.longitude;

    let watch = this.geolocation.watchPosition();
    watch.subscribe((data) => {
      this.lat = data.coords.latitude;
      this.lng = data.coords.longitude;
     });

   }).catch((error) => {
      console.log('Error getting location', error);
   });
}
```

La función `getCurrentPosition` obtiene la ubicación actual del usuario (esto es cuando se inicia la aplicación) y los valores de latitud y longitud son asignados a las variables `lat` y `lng` respectivamente.

Ahora, cuando el usuario ha cambiado su ubicación la función `watchPosition` es la encargada de detectar dichos cambios y actualiza los valores de las variables `lat` y `lng`.

5.3.3 Obtención de la geometría de la ruta seleccionada

En la sección 5.3.1 se explicó la forma en que se obtienen las rutas registradas en el sistema Movilidad UNAM y como son mostradas en la aplicación móvil del usuario, ahora, cuando el usuario selecciona una ruta en específico se ejecutara la función `cambioRuta($event)`, en donde `$event` representa el identificador único de la ruta el cual es obtenido desde la Base de datos. El contenido de la función `cambioRuta` contenido se presenta a continuación:

```
cambioRuta($event){
  this.colorRuta($event);
  this._ruta.getRuta($event).subscribe(data => {
    let puntos = data[0].replace("LINESTRING(","");
    puntos = puntos.replace(")","");
    puntos = puntos.split(",");
    for (let i = 0; i < puntos.length; i++) {
      puntos[i] =puntos[i].split(" ");
    }
    this.generaPolylinea(puntos);
  });
  this._estacion.getEstaciones($event).subscribe(data => {
    this.generaEstaciones(data);
  });
  this.camionesObs =
   this.db.collection('1').valueChanges();
    this.camionesObs.subscribe(res =>{
      this.camiones = res;
    });
}
```

Como podemos observar, dentro de la función se encuentra la variable `this._ruta`, que como habíamos mencionado anteriormente es una instancia del servicio `RutaService` y por ende ejecuta la función `getRuta` que se encuentra definida en el archivo `/src/providers/ruta/ruta.ts`:

```
getRuta(rutaId:number){
  let datos = {"rutaId":rutaId};
  let url = URL_SERVICIOS + "/ruta/getRuta";
  return this.http.post(url,datos).map(resp => resp);
}
```

Esta función se encarga de obtener la geometría de la ruta seleccionada, para esto se realiza una petición http al web service del sistema Movilidad UNAM.
La definición del web service se encuentra definido en el controlador `RestRutaController` y se presenta a continuación:

```
/**
* Método que obtiene la geometría de una ruta para la
* aplicación
* @return geometría
*/
@CrossOrigin(origins = "*")
@RequestMapping(value = "getRuta",
                method = RequestMethod.POST)
public ArrayList<String> getRuta(@RequestBody Ruta ruta) {
  ArrayList<String> rutaGeo = new ArrayList<String>();
  rutaGeo.add(rutaService.getRuta(ruta.getRutaId()));
  return rutaGeo;
}
```

Para obtener la geometría de la ruta se llega hasta el archivo `RutaDAO.java`, para ejecutar la consulta, como se muestra a continuación:

```
@Override
public String getRuta(Long rutaId) {
  Ruta ruta = getJdbcTemplate().queryForObject(
      get_ruta_sql, new Object[] { rutaId },
      new RowMapper<Ruta>() {
        @Override
        public Ruta mapRow(ResultSet rs, int rowNum)
        throws SQLException {
          Ruta ruta = new Ruta();
          ruta.setGeometria(rs.getString("geometria"));
          return ruta;
        }
      });
  return ruta.getGeometria();
}
```

Los datos obtenidos (Figura 5.7) son regresados hasta la variable `this._ruta` de la función `cambioRuta` en el archivo `src/pages/home/home.ts`:

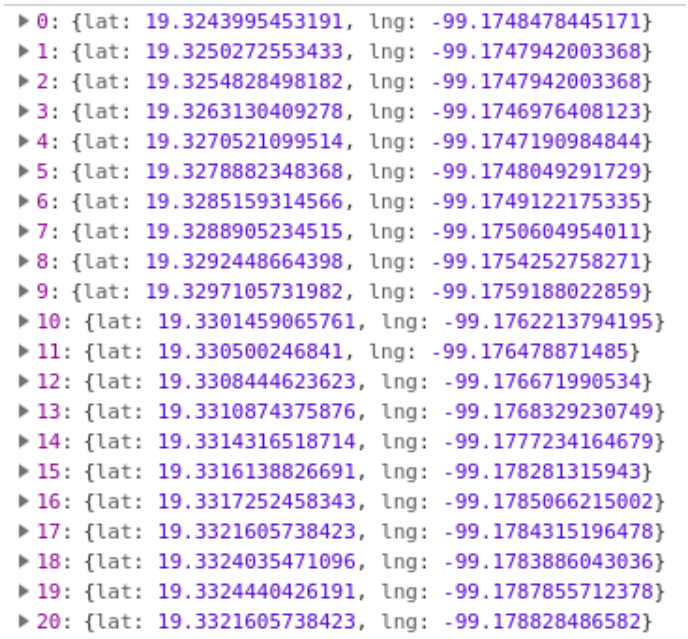


Figura 5.7 Datos obtenidos del servicio web para la geometría de la ruta

Una vez obtenido la geometría de la ruta, el siguiente paso es mostrarla en la aplicación móvil del usuario, en este caso se deberá mostrar en un mapa, la siguiente sección de código realiza esta acción:

```
<agm-polyline-point *ngFor="let geo of geometria"
        [latitude]="geo.lat" [longitude]="geo.lng">
  </agm-polyline-point>
</agm-polyline>
```

La figura 5.8 se muestra como se coloca la ruta del Pumabús en el mapa de la aplicación móvil.

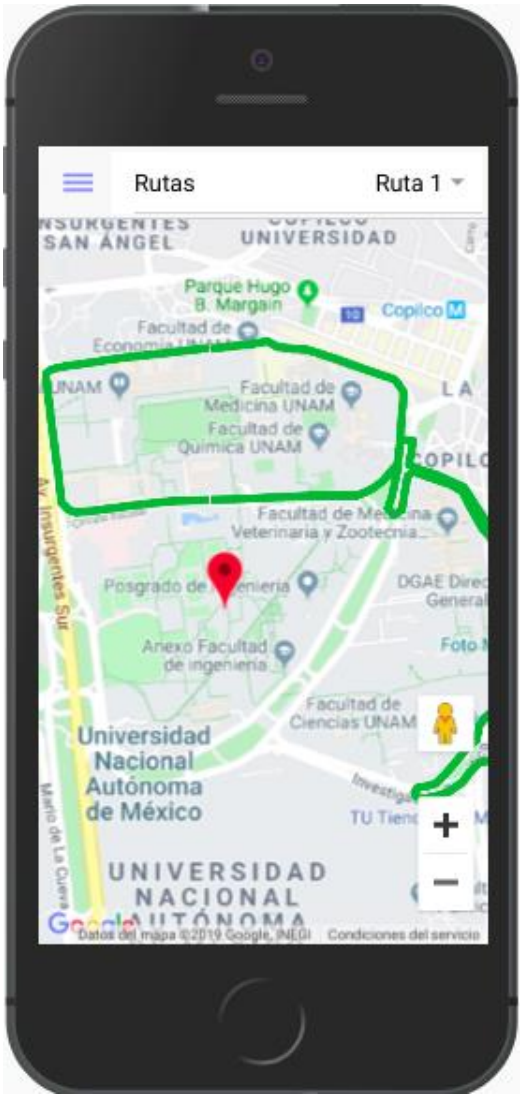


Figura 5.8 Ruta del Pumabús mostrada en la aplicación móvil

5.3.4 Obtención de las estaciones de una ruta del Pumabús

Hasta este punto solo hemos obtenido la geometría de la ruta que ha seleccionado el usuario, sin embargo, también se necesita obtener las estaciones del Pumabús para dicha ruta, la variable `this._estacion` es una instancia del servicio `EstacionService` que permite ejecutar la función `getEstaciones` del archivo `/src/providers/estacion/estacion.ts`:

```
getEstaciones(rutaId:number){
  let datos = {"rutaId":rutaId};
  let url = URL_SERVICIOS + "/estacion/getEstacionRuta";
  return this.http.post(url,datos).map(resp => resp);
}
```

De igual manera se realiza una petición http al web service del sistema Movilidad UNAM, en el archivo `RestEstacionController.java` se encuentra su definición:

```
/**
* Método que obtiene las estaciones de una ruta.
* @param ruta
* @return
*/
@CrossOrigin(origins = "*")
@RequestMapping(value = "getEstacionRuta",
                method = RequestMethod.POST)
public List<Estacion> restGetUsuario(
                      @RequestBody Ruta ruta) {
  List<Estacion> estacionesRuta =
        estacionService.getEstacionesRuta(ruta);
  return estacionesRuta;
}
```

De la misma forma, para realizar la consulta se ejecuta la función `getEstacionesRuta` del archivo `estacionDAOImpl.java`:

```
@Override
public List<Estacion> getEstacionesRuta(Ruta ruta) {
   private static final String get_estaciones_ruta_sql =
     "SELECT e.estacion_id,nombre,ST_X(geo),ST_Y(geo)
     FROM estacion e
     JOIN estacion_ruta er
     on er.estacion_id = e.estacion_id
     where er.ruta_id = ?";

  List<Estacion> listEstaciones = getJdbcTemplate().query(
         get_estaciones_ruta_sql,new Object[] {
         ruta.getRutaId() }, new RowMapper<Estacion>() {
         @Override
         public Estacion mapRow(ResultSet rs, int rowNum)
```

```
            throws SQLException {
              Estacion estacion = new Estacion();
              double[] coords = new double[2];
              estacion.setEstacionId(rs.getLong(1));
              estacion.setNombre(rs.getString(2));
              coords[0] = rs.getDouble("st_x");
              coords[1] = rs.getDouble("st_y");
              estacion.setX(coords[0]);
              estacion.setY(coords[1]);
              return estacion;
            }
          });
      return listEstaciones;
  }
```

Los datos obtenidos de la consulta a la Base de datos son devueltos hasta la variable `this.estacion` en el archivo `src/pages/home/home.ts`, la estructura de los datos se presenta a continuación (figura 5.9).

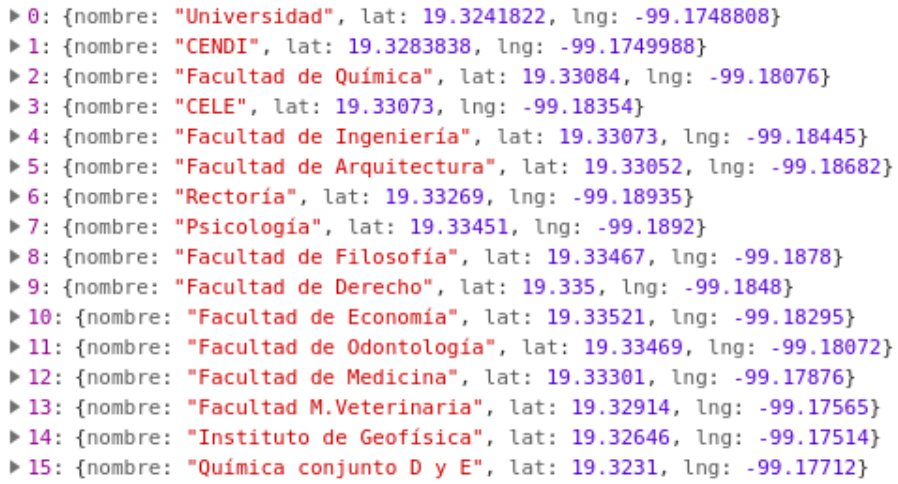


Figura 5.9 Datos obtenidos del servicio web para la geometría de las estaciones de una ruta

Para que los datos sean mostrados en el mapa de la aplicación móvil es necesario colocar el siguiente código:

```
<agm-marker *ngFor="let estacion of estaciones"
                  [latitude]="estacion.lat"
                  [longitude]="estacion.lng"
                  [iconUrl]="icono">
  <agm-info-window>
    <strong>{{estacion.nombre}}</strong><br>
  </agm-info-window>
</agm-marker>
```

La figura 5.10 presenta como se muestran las estaciones en el mapa de la aplicación móvil.

Figura 5.10 Estaciones de una ruta del Pumabús mostradas en la aplicación móvil

5.3.5 Obtención de la ubicación de las unidades del Pumabús que circulan sobre una ruta del Pumabús

Hasta este punto ya se han obtenido la geometría para la ruta y la geometría de las estaciones de dicha ruta, un aspecto importante para la aplicación móvil es mostrar los Pumabús que se encuentran circulando en dicha ruta.

Como se mencionó en capítulos anteriores, la ubicación de las unidades del Pumabús será almacenada en Firebase, para poder obtener su ubicación se ejecuta el siguiente código:

```
this.camionesObs =
   this.db.collection($event).valueChanges();
    this.camionesObs.subscribe(res =>{
      this.camiones = res;
    });
```

La variable `$event` contiene el valor con la que se identifica la ruta del Pumabús, recordando la estructura de la Base de datos en Firebase (Sección 5.2) al estar apuntando a un nodo se obtendrán los nodos subyacentes que contiene los Pumabús y su ubicación.
Los datos obtenidos se muestran en la figura 5.11

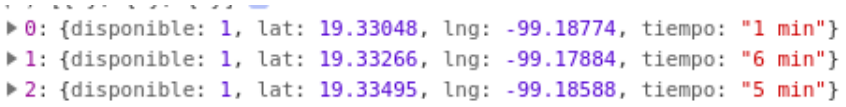


Figura 5.11 Datos obtenidos de Firebase

Como podemos observar en el arreglo de datos obtenidos de Firebase se contienen campos con valores de latitud y longitud los cuales representan la ubicación de los Pumabús, ahora, cada que se detecte un cambio en la ubicación Firebase actualizara los datos y mandara una respuesta con los nuevos valores.

En la figura 5.12 presenta como se muestran las unidades del Pumabús en una ruta específica.

Figura 5.12 Ubicación de las unidades del Pumabús en la aplicación móvil

5.3.6 Cálculo del tiempo aproximado de llegada de un Pumabús.

El cálculo del tiempo de llagada de un Pumabús se realiza en la aplicación del conductor con la finalidad de reducir los servicios y costos de uso de la API de Google Maps. A continuación, se presenta el código necesario:

Como primer punto es obtener la ubicación actual del conductor del Pumabús, para esto se ejecuta la función `inciarGeoLocalizacion` que se encuentra en el archivo `src/providers/ubicacion/ubicacion.ts`, para esto se utiliza el componente `geolocation` que es proporcionado por Cordova, la ubicación se encuentra en valores de latitud y longitud:

```
inciarGeoLocalizacion(estaciones:Array<any>){
  this.finEstacion = estaciones.length;
  this.geolocation.getCurrentPosition().then((resp) => {
    this.conductor.update({
    lat:resp.coords.latitude,
    lng:resp.coords.longitude
});
```

Ahora bien, cada que el conductor cambie su posición es necesario saberlo, la función `watchPosition` detecta dicho cambio, por cada cambio realizado se manda a llamar la función `getAllDistance`.
La función `then` obtiene los resultados de la ejecución de la función `getAllDistance` en este caso obtendrá el tiempo aproximado de llegada del Pumabús a la siguiente estación. Este tiempo de llegada y la ubicación del conductor (latitud, longitud) serán actualizados en la base de datos de Firebase.

```
let watch = this.geolocation.watchPosition();
watch.subscribe((data) => {
  this.ubicacionConductor = data;
 this.getAllDistance([
        [data.coords.latitude,data.coords.longitude]],
        [estaciones[this.inicioEstacion].lat,
         estaciones[this.inicioEstacion].lng])
.then(result => {
  this.conductor.update({
  lat:data.coords.latitude,
```

```
  lng:data.coords.longitude,
  tiempo:result[0]['text']
   });
  });
 });
}
```

La función `getAllDistance` se encarga de crear un arreglo con los resultados de la llamada a una función, para este caso se ejecuta la función `getDistance`.

```
getAllDistance (starts, end) {
  const promisedDistances = starts.map((start) =>
     this.getDistance(start, end));
  return Promise.all(promisedDistances);
}
```

La función `getDistance` es la encargada de realizar una petición al servicio `Distance Matrix` de la API de Google Maps, `Distance Matrix` permite realizar el cálculo de tiempo de llegada entre dos puntos (origen y destino), esto es basado con la información mapeada por Google.

```
getDistance (start, end) {

const origin = new google.maps.LatLng(start[0], start[1]);
const final = new google.maps.LatLng(end[0], end[1]);
const service = new google.maps.DistanceMatrixService();
return new Promise((resolve, reject) => {
       service.getDistanceMatrix({
       origins: [origin],
       destinations: [final],
       travelMode: 'DRIVING'
       },
      (response, status) => {
         if(status === 'OK') {
           resolve({
     distance:response.rows[0].elements[0].duration.value,
     text:response.rows[0].elements[0].duration.text });
        } else {
          reject(new Error('Not OK'));
```

```
            }
          }
      );
  });
}
```

Como podemos observar en el código proporcionado anteriormente, para el uso del servicio `Distance Matrix` es necesario definir parámetros, los cuales se explicarán a continuación:

- `Origins`: Punto de origen, puede estar definido en coordenadas (latitud,longitud) o una cadena con la dirección de origen.
- `Destinations`: Punto de destino, puede estar definido en coordenadas (latitud, longitud) o una cadena con la dirección de destino.
- `travelMode`: Modo en el que se llegará entre el punto de origen y de llegada. Para este caso de define el tipo `'DRIVING'`.

Con los parámetros definidos es posible realizar la petición al servicio `Distance Matrix`, la figura 5.13 muestra un ejemplo de la respuesta del servicio `Distance Matrix` en donde se puede observar que regresa la dirección de origen, dirección de llegada, distancia entre ambos puntos y el tiempo aproximado de llegada.

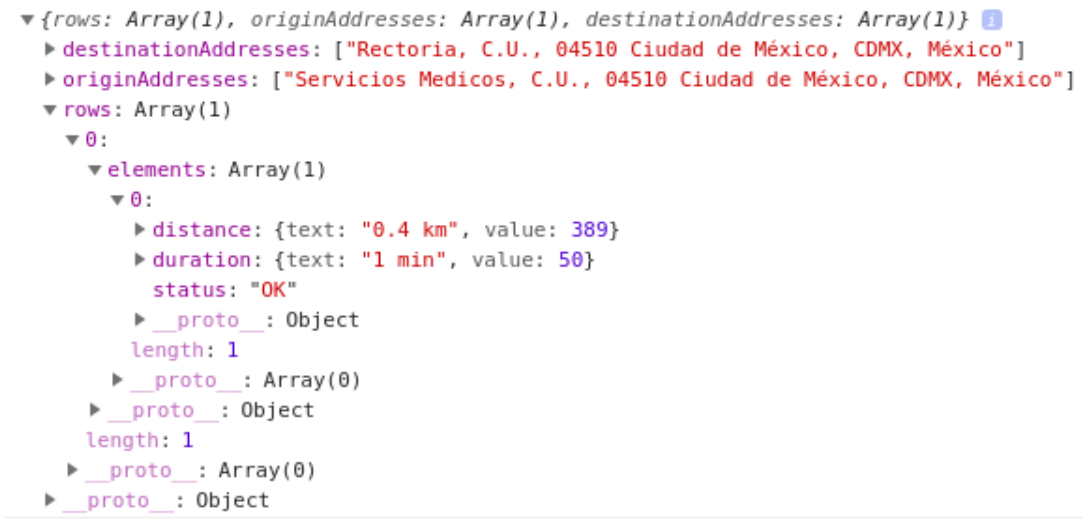


Figura 5.13 Respuesta del servicio Distance Matrix.

La figura 5.14 representa el resultado final de los procesos mencionados anteriormente en el módulo de Pumabús en la aplicación móvil de usuarios, como se puede observar, se muestra en el mapa la ruta, las estaciones asociadas a la ruta 1, los Pumabús que se encuentran circulando en dicha ruta, y el tiempo aproximado de llegada.

Figura 5.14 Resultado final en la aplicación de usuarios

A lo largo del presente capítulo se habló acerca del proceso de desarrollo para la aplicación móvil de los usuarios, específicamente en el módulo del Pumabús. Con las herramientas seleccionadas: Base de datos PostGIS, FireBase y el framework Ionic se logró el desarrollo de este módulo, además, es posible identificar la forma que interactúan dichos componentes para la obtención y muestreo de la información. Por otro lado, podemos darnos cuenta que se cumplieron con los requerimientos definidos en el capítulo 3.

Si bien, la funcionalidad de este módulo ya está realizada, el proceso de desarrollo de software no termina en este punto, ya que ahora es turno del equipo de pruebas para corroborar el funcionamiento adecuado del módulo, esto se verá en el siguiente capítulo.

6. PRUEBAS

Durante el presente documento se han explicado diferentes fases para el desarrollo de software como son: la definición del problema, la especificación de requerimientos tanto funcionales como no funcionales, el diseño del sistema y como llevar la implementación (desarrollo).

Finalmente, una de las últimas etapas de este proceso es el de pruebas. La etapa de pruebas es la encargada de revisar, validar y verificar que el sistema desarrollado cumpla con la funcionalidad esperada por el usuario final y que se tenga un nivel de calidad adecuado para su óptimo funcionamiento.

Al desarrollar un producto de software existe la posibilidad de cometer errores los cuales provocan defectos en su funcionalidad. Estos errores pueden ser causados por diferentes razones, por ejemplo: falta de entendimiento en los requisitos, presión al momento de la programación, requerimientos complejos, etc. Estos errores pueden causar problemas que afecten a ambas partes, tanto la empresa o personas que desarrollaron el software como al usuario final: pérdidas económicas, pérdida de credibilidad en la empresa, daños personales, etc. Por esta razón es de suma importancia contar con una etapa de pruebas para evitar consecuencias perjudiciales.

Algunos de los objetivos en la fase de pruebas son:

- Encontrar errores, mejor conocidos como bugs, con la finalidad de resolverlos.
- Aumentar la calidad del software.
- Asegurar al usuario final que el software funciona correctamente.
- Corroborar que el software debe de hacer según lo especificado en etapas anteriores.
- Detectar funcionalidades que el software realice y no estén especificadas en la documentación.
- Facilitar información para la toma de decisiones.
- Evitar la aparición de errores.

Durante esta fase, los miembros del equipo realizan diferentes tipos de pruebas. En caso de encontrar errores o inconvenientes con el software deberán ser notificadas al equipo de desarrollo para que sean resueltas en el menor tiempo posible. Este proceso puede ser iterativo hasta que el software presente el menor número de errores.

Prácticamente, el equipo encargado de realizar las pruebas será el encargado de notificar si el software está listo para ser puesto en producción y el usuario final pueda utilizarlo. Por el lado contrario, el software no estará en producción si el equipo de pruebas no aprueba el funcionamiento del software, asegurando al cliente un software de calidad.

Para que la etapa de pruebas sea efectiva y se logre alcanzar los objetivos, es necesario planificar un proceso que permitirá definir el flujo de trabajo, así como, acciones a realizar. Dicho proceso de pruebas puede depender de las características del equipo de trabajo.

6.1 Proceso de pruebas

Para la etapa de pruebas es importante tener definido el proceso que se llevará a cabo, esto, con la finalidad de tener una planificación y control de todo el proceso, ya que no basta la propia ejecución sin orden, ni planeación alguna. Recordemos que la etapa de pruebas puede ser iterativa hasta la reducción mínima de errores. A continuación, se presentan las principales actividades que integran el proceso de pruebas.

- *Planificación y control*. Se definen los objetivos de la etapa de pruebas, así como las actividades a realizar para alcanzar dichos objetivos.

- *Análisis y diseño*. En este punto del proceso de pruebas, los objetivos se convierten en condiciones de pruebas que permiten identificar los elementos que deben ser verificados. Dichas condiciones deben ser priorizadas y es necesario definir los datos necesarios para su ejecución.

- *Implementación y ejecución*. Proceso en el que se llevan a cabo las pruebas en un orden determinado, dichas pruebas pueden llevarse a cabo de manera manual o con ayuda de herramientas; mejor conocidas como pruebas automatizadas. En la actualidad, las empresas han optado por llevar a cabo este tipo de pruebas ya que hay una mayor efectividad en tiempo y costo.

  Durante el proceso de Implementación y ejecución, se comparan los datos reales con los resultados esperados. En este punto es en donde el equipo de pruebas puede encontrar errores los cuales serán reportados como incidencias.

- *Evaluación de resultados*. En este punto se evalúan los resultados obtenidos en la etapa de pruebas con base a los objetivos planteados.

- *Cierre de pruebas*. Se deben de cerrar las incidencias reportadas, verificando que se haya solucionado el problema o en su defecto ver por qué no han sido atendidas

Como se ha mencionado, las pruebas son actividades que se llevan a cabo con la principal finalidad de encontrar fallos en la implementación de un software. Hay que tomar en cuenta que existen diferentes tipos de pruebas, los cuales permiten verificar y corroborar características o funcionalidades específicas del software.

Dentro del área de pruebas podemos encontrar las pruebas estáticas y dinámicas: el primer tipo de pruebas se enfoca en el análisis y revisión del código sin ser ejecutado, mientras que las pruebas dinámicas necesitan la ejecución del código.

Las pruebas dinámicas están subdivididas en pruebas funcionales y no funcionales, dentro de estas clasificaciones podemos encontrar una serie de categorías de pruebas: componentes, integración, aceptación, rendimiento, seguridad, usabilidad, mantenibilidad, eficiencia, entre otras. A continuación de detallará los principales tipos de pruebas.

6.2 Pruebas estáticas.

Las pruebas estáticas se caracterizan en que no es necesario ejecutar el código elaborado para su realización. Estas pruebas se basan en la revisión manual y análisis automatizado del código o hasta de la misma documentación.

En la mayoría de los casos, las pruebas estáticas son realizadas antes que las pruebas dinámicas, esto es porque, si se encuentra algún defecto en la revisión estática el costo de su eliminación será menor a que si se realiza la eliminación en una prueba dinámica.

Existen diferentes elementos que pueden ser considerados en las pruebas estáticas: especificación de requerimientos, especificaciones del diseño, la estructura y flujo del código, el plan y especificación de las pruebas, guías de usuario, etc. Gracias a las pruebas estáticas se puede detectar y corregir de manera temprana los errores, se logra la reducción de tiempo en el desarrollo, reducción de tiempo en la etapa de pruebas, reducir cierto número de errores.

Actualmente, para la realización de las pruebas estáticas se utilizan herramientas, por ejemplo, PMD en sus siglas en inglés Programming Mistake Detector, el cual se encarga de analizar código fuente y permite obtener una evaluación del código basado en reglas predefinidas o personalizadas:

- Posibles defectos, por ejemplo, switch vacíos.
- Variables, parámetros y métodos no utilizados.
- Código no óptimo.
- Sentencias If innecesarias.
- Código duplicado.

A continuación, se mencionan ciertos puntos para la importancia de realizar pruebas estáticas:

- Detección temprana de errores.
- Detección de código sospechoso que pueda dar origen a un error.
- Identificar errores que en las pruebas dinámicas no es posible encontrar.
- Mejorar la estructura del código y del diseño.
- Prevención de errores.

6.3 Pruebas dinámicas.

6.3.1 Pruebas funcionales

En etapas previas, la funcionalidad del sistema se define con ayuda de la especificación de requerimientos, casos de uso, diagramas de secuencia, entre otros componentes. Las pruebas funcionales se encargan de verificar lo mencionado anteriormente, es decir, se encargan de verificar que la funcionalidad del sistema concuerde con lo estipulado en la documentación; prácticamente se enfoca en verificar que las acciones realizadas por el sistema sean correctas.

Dentro de las pruebas funcionales existen diferentes subtipos. A continuación, se mencionan tres de ellas:

- Pruebas de componente o unitarias.

  Las pruebas de componente o también conocidas como pruebas unitarias consisten en aislar una parte del código del software para verificar que funcione adecuadamente y en dado caso encontrar errores. Estas pruebas validan la lógica del código, además, el elemento a verificar en las pruebas unitarias no debe tener dependencia con otros elementos o componentes del sistema.

  Para realizar este tipo de pruebas se debe de tomar en cuenta la documentación realizada en etapas previas, por ejemplo, se suele utilizar los casos de uso para corroborar su funcionamiento.

- Pruebas de integración.

Las pruebas de integración se realizan a las diferentes interfaces de los componentes que conforman el sistema, verifican la interacción entre los componentes de software, así mismo, como la interacción entre el software y el hardware. Por ejemplo, si en un sistema se tienen dos módulos A y B, en las pruebas de interacción se deberá de verificar que la comunicación entre estos dos módulos se realice de manera correcta y no verificar la funcionalidad de cada módulo ya que eso lo realizan las pruebas de componente.

Para que se lleven a cabo de manera correcta las pruebas de integración, los miembros del equipo de pruebas deberán de entender la arquitectura del sistema.

- Pruebas de aceptación.

Las pruebas de aceptación evalúan la disponibilidad del sistema para su uso e interacción con usuarios. A diferencia de los tipos de pruebas mencionados anteriormente, las pruebas de aceptación no tienen como objetivo encontrar errores dentro del sistema. En ocasiones las pruebas de aceptación son asociadas al usuario final.

Las pruebas de aceptación pueden realizarse una vez que el software haya sido instalado para que el usuario final pueda hacer uso de él, también pueden llevarse a cabo durante las pruebas de componentes

6.3.2 Pruebas no funcionales

- Pruebas de rendimiento.

Las pruebas de rendimiento tienen como finalidad evaluar al software en aspectos como velocidad, fiabilidad y estabilidad. Al realizar este tipo de pruebas se pueden identificar cuellos de botella al momento de su uso y también, identificar y localizar problemas de rendimiento dentro del software.

Dentro de las pruebas de rendimiento podemos encontrar las pruebas de carga; enfocadas a realizar el número de peticiones concurrentes que soportará el sistema y que fue especificado en los requerimientos, también están las pruebas de estrés; en donde se exceden el número de peticiones concurrentes especificadas para ver la forma en que reacciona el sistema, y finalmente las pruebas de estabilidad; en donde por un tiempo prolongado se utiliza el sistema con un número de peticiones

concurrentes aceptables para determinar si el sistema funciona correctamente durante este periodo de prueba.

- Pruebas de usabilidad.

La usabilidad se refiere al proceso en el que un sistema es entendido, aprendido y utilizado por usuarios. Esta característica debe de ser tomada en cuenta cuando se inicia con el proyecto ya que si pasa por desapercibido, al final del proyecto puede causar consecuencias y provocar grandes cambios dentro del sistema.

6.4 Automatización de pruebas.

La automatización en la etapa de pruebas ha ido tomando un papel relevante, consiste en disminuir la intervención humana para verificar el funcionamiento correcto de un software. Prácticamente las pruebas automatizadas realizan los pasos que una persona ejecutaría para verificar las salidas de un cierto proceso y compararlos con los resultados esperados.

La automatización de pruebas no elimina del todo la participación humana, debe de considerarse como un apoyo a las actividades del equipo de pruebas; de todos los procesos deben de identificarse cuales son automatizables para poder realizar las pruebas con herramientas, así el equipo de pruebas tendrá más tiempo y podrán enfocar su esfuerzo en resolver aspectos más complejos que la automatización de pruebas no pudo detectar. Además, las pruebas automatizadas son reutilizables durante la etapa de pruebas.

Algunos de los beneficios se presentan a continuación:

- Rapidez.
- Fiabilidad
- Repetición
- Programable
- Reusabilidad

Existen ciertos aspectos que deben de ser considerados al tratar de implementar la automatización en la etapa de pruebas, por ejemplo: un proceso automatizado difícilmente evaluará la usabilidad del sistema. Para generar las pruebas automatizadas es necesario tener conocimiento de programación y su mantenimiento puede llegar a ser costoso. Sin embargo, el costo puede ser rentable comparado con el beneficio obtenido. Hay que tomar en cuenta

que este tipo de pruebas pueden ser reutilizadas, y representa una ventaja para las empresas que deciden hacer uso de este tipo de pruebas.

6.5. Reportar errores.

El reporte de errores o mejor conocidas como incidencias o bugs dentro de un sistema es una parte fundamental en la etapa de pruebas, ya que en las incidencias se especifica detalladamente el problema encontrado, dichas incidencias deberán ser reportadas con un lenguaje claro e intuitivo para el equipo de desarrollo con la finalidad de ser resuelta.

Al reportar una incidencia normalmente se incluye a siguiente información:

- Proyecto: Toda incidencia debe estar relacionada a un proyecto de software.
- Módulo: En dado caso que el proyecto tenga módulos, se deberá especificar el módulo en donde fue encontrada la incidencia.
- Título: Una descripción muy breve de lo que consiste la incidencia.
- Descripción: Explicación detallada del problema encontrado.
- Reproducibilidad: Indica si el problema encontrado se reproduce de manera frecuente, a veces. Aleatoria o no reproducible.
- Severidad: Impacto que tiene el problema sobre el proyecto.
- Prioridad: Indica la urgencia que tiene el problema por ser resuelto. Puede tomar los siguientes valores; baja, media, alta.
- Estado: Las incidencias tienen un ciclo de vida en las que pueden pasar por diferentes estados, por ejemplo: Aceptada, Confirmada, Asignada, Resuelta, Cerrada, Reabierta, etc.
- Asignación: Se indica al miembro del equipo que tiene asignada la incidencia con la finalidad de analizar el problema y resolverla.
- Evidencias: Capturas de pantalla que corroboren la presencia del error dentro del sistema.

En la fase de pruebas normalmente se utilizan herramientas para la gestión y manipulación de los errores encontrados, estas herramientas permiten tener un seguimiento más específico de los errores, como, por ejemplo: en que parte se encontró el error, que pasos se siguieron para provocar dicho error, personal del equipo de desarrollo que atenderá dicho error, entre otra información.

6.5.1 Mantis Bug Tracker

Mantis es una de las herramientas para el manejo de incidencias dentro de un equipo de trabajo. Dicha herramienta permite el reporte de incidencias para un proyecto de desarrollo de software, de hecho, se pueden registrar un número indeterminado de incidencias, a las cuales permite asignarle diferentes estados, por ejemplo: abierta, cerrada, reportada, en desarrollo, entre muchos otros.

Además, permite la creación de perfiles para ser asignados a los miembros del equipo de desarrollo, por ejemplo: programador, tester, administrador, etc. Gracias a esta herramienta es posible llevar el control completo de las incidencias encontradas en un software, así como tener un historial de todas ellas.

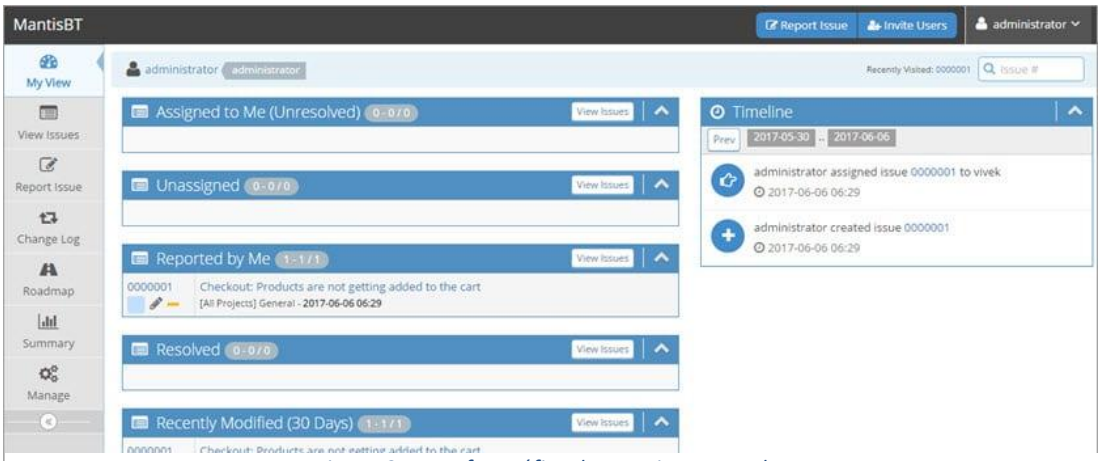


Figura 6.1 Interfaz gráfica de Mantis Bug Tracker

6.5.2 Jira

Jira es una herramienta que permite la administración y gestión de proyectos de software, entre una de sus funciones que provee a los clientes se encuentra el reporte de incidencias o bugs. Jira está enfocado a metodologías ágiles, de hecho, permite la implementación de sprints, manejo de tableros para tener un mayor control de las actividades a realizar por el equipo, permite la creación del Backlog, entre otras funcionalidades.

Algunas de las características de Jira son:

- Permite la creación de varios proyectos.
- Permite la planificación de sprints, creación de historias de usuario, reporte de incidencias.

- Permite priorizar las actividades.
- Es adaptable a las metodologías ágiles como Scrum, Kanban o una metodología propia.
- Permite la creación de reportes.
- Permite la sincronización con el versionador de código Bitbucket.

A diferencia de Mantis Bug Tracker, Jira si tiene un costo ya que ofrece múltiples funcionalidades.

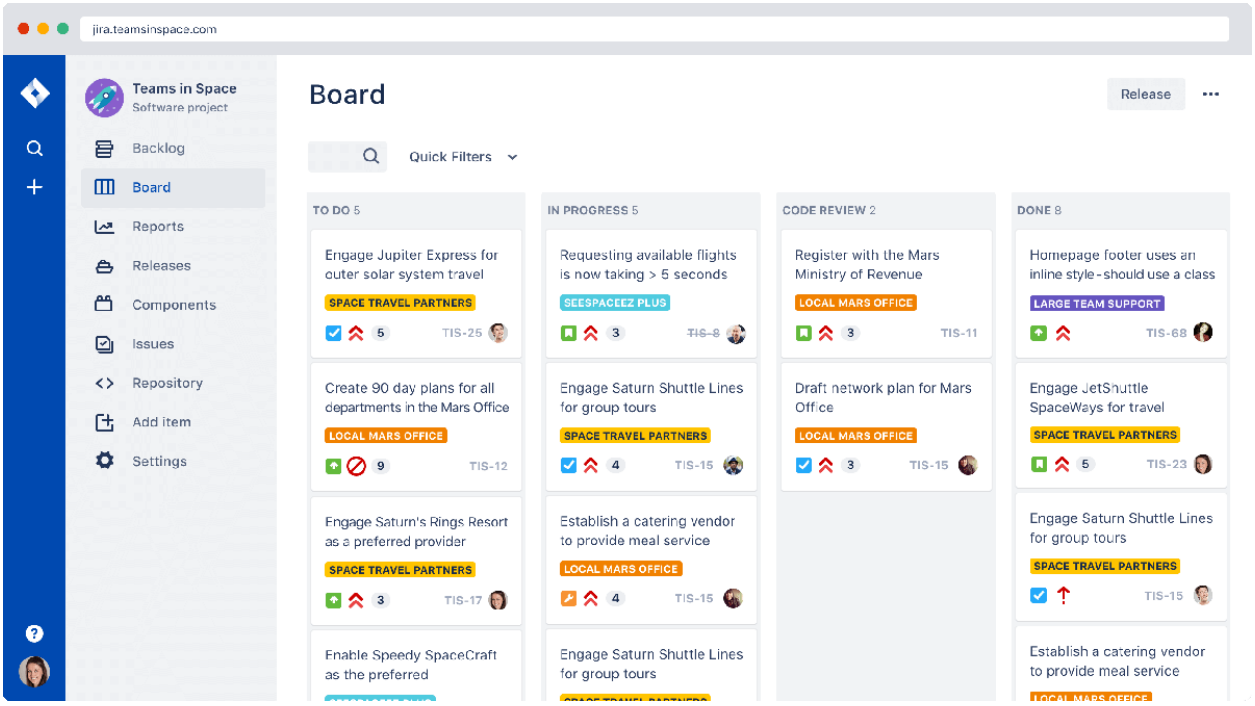


Figura 6.2 Interfaz gráfica de Jira.

6.6 Pruebas en el sistema

Como se ha mencionado anteriormente, el proceso de pruebas es importante para identificar inconsistencias en el software, en este apartado se ilustraran algunas pruebas realizadas en el sistema Movilidad UNAM.

Para realizar pruebas unitarias en java se utilizará el framework Junit. Dicho framework provee una serie de herramientas, clases y métodos que permiten asegurar la funcionalidad correcta de un sistema dado.

6.6.1 Pruebas unitarias en el sistema

Usualmente, cuando se realizan pruebas unitarias en un sistema, se prueban métodos que reciben datos por medio de parámetros, son procesados y se obtiene una respuesta, sin embargo, hay que tomar en cuenta que no todos los métodos dependerán únicamente de los datos de entrada, por ejemplo, pueden existir métodos que consulten información a la Base de datos y dependiendo de la información obtenida se regresa una respuesta.

Un punto a tomar en cuenta, es que si realizamos una prueba en donde se haga una consulta a la Base de datos, dicha prueba dejaría de ser una prueba unitaria (o de componente) ya que la base de datos es considerada como una dependencia externa al sistema.

Existe una técnica ampliamente utilizada en la práctica basada en los llamados Mock Objects. Un Mock object permite resolver las dependencias que un método, función o bloque de código pudiera tener. Estas dependencias pueden ser representadas a través de un Mock Object para simular su funcionamiento.

Lo anterior permite la posibilidad de realizar pruebas sobre una pieza de código sin tener que preocuparse por resolver sus dependencias. Cabe destacar que el objetivo de la prueba no es verificar el correcto funcionamiento de sus dependencias.

Un ejemplo común es la verificación y validación de los métodos de un servicio que implementan lógica de negocio. Dicho servicio depende de un componente DAO para acceder a la base de datos. En este caso, se escribe un Mock Object que simule al DAO de tal forma que la prueba se concentra en verificar la lógica del servicio sin preocuparse en este caso, de verificar el correcto funcionamiento del DAO. Al tener respuestas simuladas, se elimina la complejidad de resolver dependencias entre los componentes del sistema.

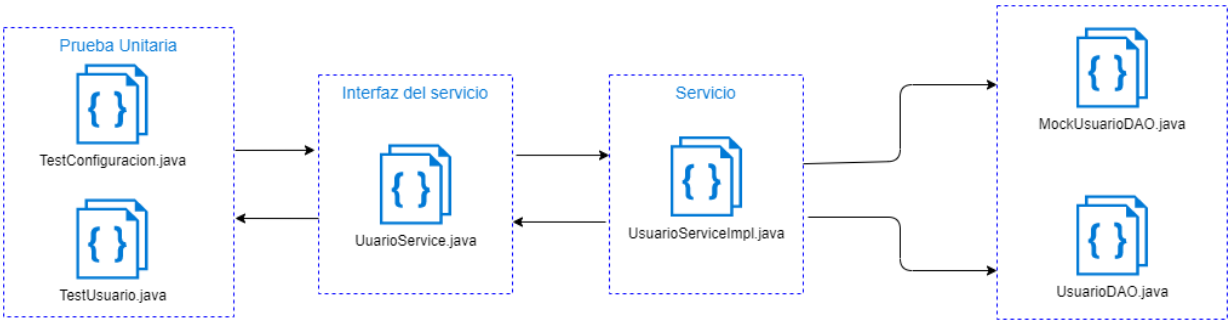


Figura 6.3 Prueba utilizando objeto Mock

La figura 6.3 muestra la relación de los archivos involucrados en una prueba con uso de los objetos Mock. A continuación, se describe dicho proceso.

Primeramente, se debe de crear la configuración para detectar y realizar una prueba dentro de Spring, la configuración se tomará en cuenta solo cuando se esté trabajando con el modo de prueba activo. Además, con esto se garantiza que Spring simule el objeto Mock y no el DAO.

Ahora bien, una vez establecida la configuración se procede a realizar la prueba unitaria, en la Figura 5.3 el archivo es llamado `TestUsuario.java`, en la prueba unitaria se habilita el modo prueba y por ende se activa la configuración definida anteriormente. Spring detecta esta configuración y crea una instancia de los archivos `UsuarioService.java` y `UsuarioServiceImpl.java` pero una instancia simulada de la clase que se encuentra en el archivo `MockUsuarioDAO.java`, con este proceso se evita la ejecución del archivo UsuarioDAO.java y la realización de alguna consulta a la Base de datos para así poder probar la lógica del servicio.

6.6.2 Pruebas de integración en el sistema

En el punto anterior, se mostró el proceso necesario para realizar una prueba con el uso del objeto Mock. Para este ejemplo, al tratarse de una prueba de integración es permitido acceder a la información de la Base de datos.

Se probará la función `getUsuario()` que se encuentra en el archivo DAO `UsuarioDAOImpl.java`. Esta función recibe un valor entero que representa el id del usuario

y regresa información del usuario como: nombre, apellido paterno, apellido materno y correo electrónico.

El primer paso es crear y escribir la prueba unitaria, en este caso, las pruebas se crean en la ubicación `/src/test/java`.

A continuación, se presenta el código de la prueba unitaria del archivo `/src/test/java/UsuarioDAOTest.java`

```
package mx.unam.fi.tesis.movilidad.tests.logging;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;

import mx.unam.fi.tesis.movilidad.dao.UsuarioDAOImpl;

/**
 * Prueba unitaria
 */

public class UsuarioDAOTest {

  @Test
  void verificarUsuario() {
    UsuarioDAOImpl usuarioDaoImpl = new UsuarioDAOImpl();
    Usuario usuario = new Usuario();
    usuario = usuarioDaoImpl.getUsuario(1);

    Assertions.assertEquals("Cristian", usuario.getUsuNombre(),
     "Error al consultar el nombre");
    Assertions.assertEquals("Vargas",
     usuario.getUsuPrimerApellido(),
     "Error al consultar el primer apellido");
    Assertions.assertEquals("Flores",
     usuario.getUsuSegundoApellido(),
     "Error al consultar el segundo apellido");
    Assertions.assertEquals("criz.2410@gmail.com",
     usuario.getUsuCorreo(),
     "Error al consultar el correo electrónico");
  }
}
```

Como se logra observar, la función `verificarUsuario()` tiene la notación `@Test`. Dicha notación le indica a JUnit que se trata de un método Test y que deberá ser ejecutado en la prueba.

Siguiendo el código fuente de la prueba, se ejecuta la función `getUsuario(1)` del DAO definido, si se ejecuta la siguiente consulta en la BD de datos:

```
select * from usuario where usuario_id = 1;
```

Se obtiene la siguiente información (Figura 6.4):

	usuario_id numeric (10)	usu_nombre character varying (30)	usu_primer_apellido character varying (30)	usu_segundo_apellido character varying (30)	usu_correo character varying (50)	usu_contrasena character varying (20)
1	1	Cristian	Vargas	Flores	criz.241095@gmail.com	[null]

Figura 6.4 Registro de usuario en la Base de datos

La información presentada anteriormente es la misma que regresa la función `getUsuario(1)` del DAO.

Una vez obtenido la información del DAO, dentro de la prueba unitaria se utiliza el método `Assertions.assertEquals`. Este método es proporcionado por JUnit y permite comparar si dos objetos son iguales o no.

Para poder ejecutar la prueba en JUnit se necesita dar clic derecho sobre el archivo Test que se creó y dar clic en `Run as > JUnit Test`.

En dado caso que los objetos sean iguales se obtendrá siguiente la salida (Figura 6.5):

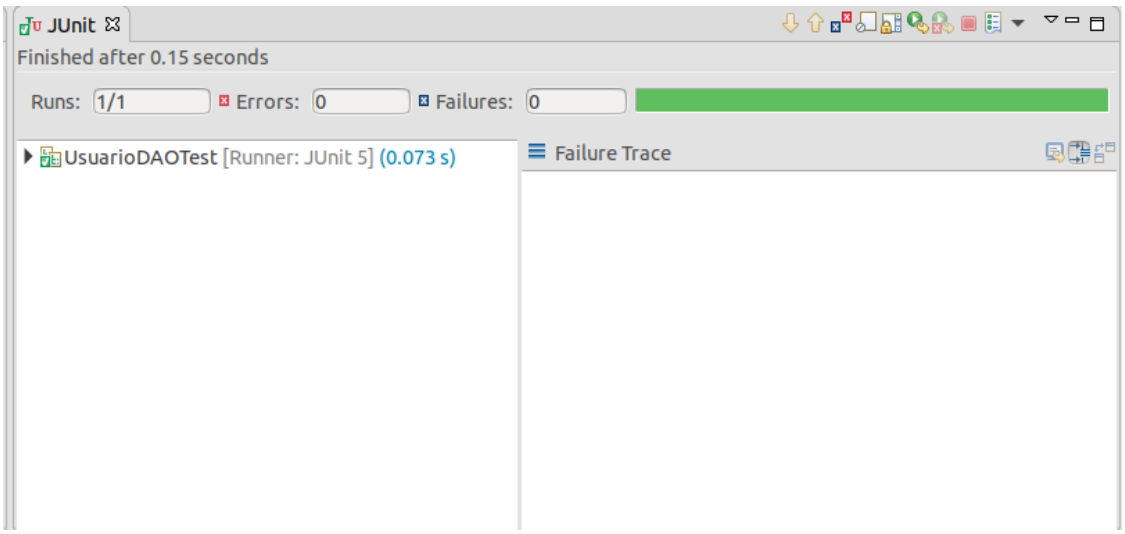


Figura 6.5 Respuesta al ejecutar una prueba con JUnit

Ahora bien, si cambiamos uno de los valores esperados en la función `Assertions.assertEquals` se producirá un error ya que el valor esperado y el valor del usuario no es el mismo, por ejemplo, si cambiamos lo siguiente dentro del método `verificarUsuario()`:

```
Assertions.assertEquals("Christian", usuario.getUsuNombre(),
     "Error al consultar el nombre");
```

Como podemos ver la cadena del valor esperado `"Christian"` es diferente a la cadena que nos regresa la Base de datos `"Cristian"`, si ejecutamos la prueba unitaria nos saldrá lo siguiente (Figura 6.6):

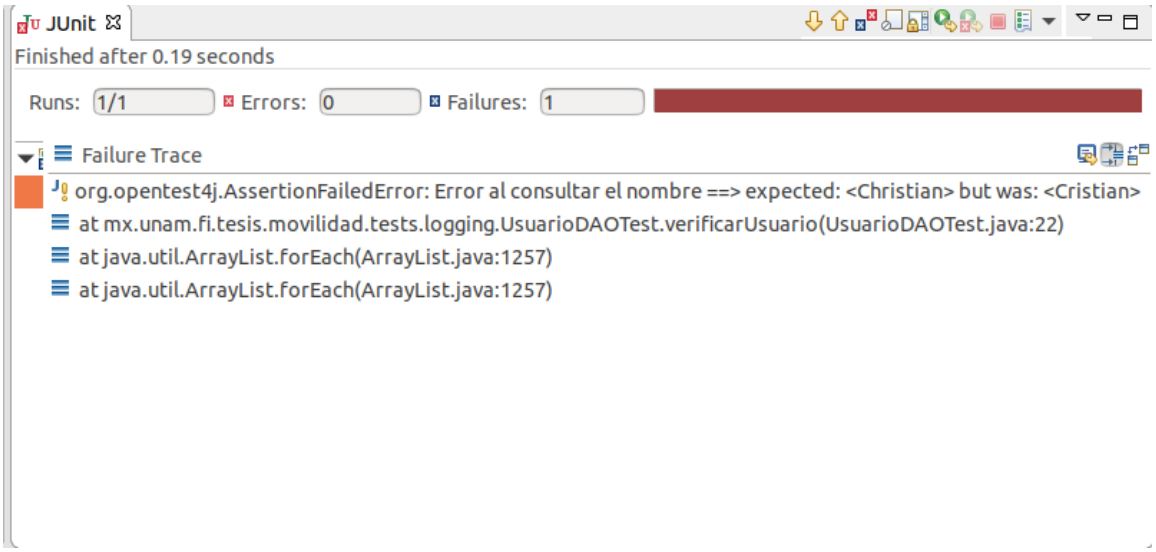


Figura 6.6 Respuesta errónea al ejecutar una prueba con JUnit

Como se ha demostrado anteriormente las pruebas permiten la verificación del funcionamiento adecuado del software, además, nos permiten identificar errores en la lógica de su funcionamiento.

Existen diferentes tipos de pruebas con una finalidad específica para cada una de ellas. En este apartado se ilustraron las pruebas unitarias y de integración de manera representativa en el sistema Movilidad UNAM. Cabe mencionar, que las pruebas automatizadas han tomado un papel muy importante en la actualidad, y en un futuro podrían aplicarse en el sistema desarrollado.

6.7 Áreas de oportunidad.

Una vez cumplido con los requerimientos y el proceso que conlleva el desarrollo de software es importante identificar áreas de mejora dentro del software para brindar una mejor experiencia.

Para el proyecto del presente documento es importante mencionar puntos a mejorar en el futuro. Uno de estos puntos obviamente es seguir tanto con la documentación, así como con el desarrollo de los módulos faltantes: estacionamientos, taxis, etc. Además, evaluar la posibilidad de que el sistema y la aplicación móvil no solo estén enfocadas al campus de Ciudad Universitaria, como bien sabemos, existen otros campus que a su vez cuentan con su propio medio de transporte, estos medios de transporte podrían complementar más el sistema permitiendo abarcar un mayor porcentaje de la comunidad estudiantil.

Por otro lado, un punto que ayudaría de manera significativa tanto en el desarrollo del software y pruebas de este mismo, seria colocar el sistema web en un servidor y habilitar las aplicaciones móviles en las tiendas virtuales de los respectivos sistemas operativos.

El servidor nos permitiría identificar como el sistema actuaría en un caso más real, ya que en la fase de desarrollo todos los procesos se corrían de manera local, además, al tener el sistema en un servidor los dispositivos móviles podrían realizar las peticiones al sistema desde cualquier ubicación, tomando en cuenta que cuando el sistema se prueba de manera local, tanto el sistema como la aplicación móvil deben de estar en la misma red para que exista una comunicación correcta. Aunado a esto, un servidor nos brinda ciertas características: mayor memoria, mayor ancho de banda, menor tiempo de respuesta en las peticiones, mayor número de procesos ejecutados, etc.

Se mencionó anteriormente la posibilidad de habilitar las aplicaciones móviles en sus respectivas tiendas virtuales, con este punto se podría aumentar la muestra de dispositivos móviles físicos en los que se podrían realizar las pruebas, y percatarnos de cómo influyen las características de los dispositivos móviles físicos en el uso de las aplicaciones.

Así mismo, es importante realizar pruebas funcionales tanto en el sistema web como en las aplicaciones móviles, principalmente para identificar como respondería los componentes cuando reciben valores de entradas no esperados y ver si hacen el manejo adecuado de estos valores.

CONCLUSIONES.

El problema principal que se abordó en el presente documento fue la movilidad dentro del Campus de Ciudad Universitaria. Existen ciertos factores que están fuera de nuestro alcance, por ejemplo la sobrepoblación de alumnos. A causa de estos factores es probable que no se pueda erradicar el problema de movilidad dentro de ciudad universitaria, sin embargo, se pretende brindar una solución o herramienta que permita solventar dicha problemática. En este caso, se brinda una herramienta que hace uso de tecnologías para aplicaciones móviles para que las personas puedan tomar la mejor decisión en cuanto al medio de transporte a utilizar dentro de Ciudad Universitaria, o bien identificar los estacionamientos con lugares disponibles y permitirles llegar a tiempo a su destino.

Una vez establecido el problema, fue importante llevar a cabo un análisis más afondo para identificar sus causas y así identificar el impacto en la comunidad académica y estudiantil en Ciudad Universitaria. Lo anterior, permitió tener un panorama más general de la situación e identificación de la solución.

Para implementar la solución planteada, se realizó una investigación de las herramientas a utilizar, permitiendo identificar las características de cada una de ellas y determinar si era viable su uso o no. Para el caso del framework Ionic, en los últimos años esta solución ha tomado un papel muy importante en el desarrollo de aplicaciones móviles por la rapidez que conlleva su implementación. Además, hace uso de tecnologías web empleadas ampliamente en la industria del desarrollo de software.

Por otro lado, la incorporación del framework de Spring, en particular el módulo de Spring MVC, nos permitió generar una aplicación organizada en sus componentes mejorando diversos aspectos como son la eliminación de código innecesario a través de mecanismos como la inyección de dependencias, entre otros.

Hasta este punto se ha definido el problema, la solución y las herramientas que nos permitirían desarrollarla, sin embargo, faltaba definir las acciones del sistema, puntos a considerar en su desarrollo, flujos de las acciones, entre otros aspectos. Esto fue cubierto en la fase de análisis de requerimientos en la cual se describen detalladamente las funcionalidades y características principales del sistema.

A lo largo de esta fase se incorporaron diferentes herramientas como los diagramas UML. Gracias a los casos de uso se definieron los actores del sistema, así como también se permitió

identificar sus acciones a realizar. Por otro lado, los diagramas de actividades permitieron ilustrar y comprender el flujo de las actividades o funcionalidades definidas previamente. En este punto se logra visualizar la lógica e identificar si un proceso está mal definido o se está omitiendo.
Esta fase es de suma importancia durante el desarrollo de cualquier sistema de software ya que permite identificar las principales reglas de negocio, identificar errores en etapas tempranas del ciclo de vida del software reduciendo la posibilidad de fallas e inconsistencias en etapas futuras.

Gracias a la etapa de diseño se logró establecer y comprender la interacción de los diferentes componentes que conforman al sistema. El proceso de diseño permitió tener una idea mucho más clara de su arquitectura y la forma en la que los componentes interactúan entre sí.

Con lo definido anteriormente, se llegó a la fase de implementación. Las tecnologías y herramientas definidas al principio del documento, principalmente los frameworks de Spring MVC y IONIC permitieron el desarrollo tanto del sistema como de las aplicaciones móviles con un funcionamiento correcto, cumpliendo con los requerimientos definidos en su respectivo capítulo. Aunado a esto, la interacción de los diferentes componentes como la Base de datos en PostGIS, el sistema Movilidad UNAM, las aplicaciones móviles y la Base de datos en Firebase la solución fue desarrollada de manera satisfactoria.

La incorporación de la base de datos espacial PostGIS permitió almacenar la información geométrica requerida: los polígonos de los estacionamientos, líneas de las rutas y puntos de las estaciones, además, permitió hacer el uso de las funciones que provee el manejador para las peticiones que se realizaban desde la aplicación móvil, por ejemplo: obtener las estaciones cercanas a la ubicación del usuario.

Por otro lado, la base de datos en Firebase nos permitió almacenar y actualizar de manera inmediata la información dinámica del sistema: ubicación de los conductores, disponibilidad de los estacionamientos y estaciones de las bicipuma. Una de las ventajas proporcionadas por Firebase es la sincronización en tiempo real con los dispositivos móviles, ya que, gracias a esto, se permitió mostrar el movimiento de traslado que generaban las unidades del Pumabús. Aun nado a esto, es importante mencionar que su uso es sencillo permitiendo realizar operaciones directas y fáciles de implementar.

Finalmente, la fase de pruebas permitió conocer conceptos acerca de esta área y realizar algunos ejercicios prácticos e ilustrativos en el software desarrollado para verificar y validar el

funcionamiento correcto del sistema. Se hizo énfasis en la importancia de realizar pruebas automatizadas y comprobar los beneficios que estas ofrecen.

Personalmente este proyecto me ha dejado demasiado aprendizaje: primero para darme cuenta de la importancia del ciclo de vida del desarrollo de software y que cada fase es necesaria para cumplir con los objetivos planteados. Anteriormente, no estaba tan familiarizado con la parte de documentación: especificación de requerimientos y diseño del sistema, gracias a este proyecto he adquirido conocimientos desde cómo realizar la especificación de requerimientos, las características de los diagramas de UML, la información se presenta en los diferentes diagramas, el proceso para diseñar la arquitectura y también acerca de herramientas muy interesantes como Spring, IONIC y Firebase que me permitieron ampliar mis conocimientos técnicos y que servirán en un futuro.

Otros puntos que quisiera mencionar y que me han dejado aprendizaje después de desarrollar el presente proyecto es la organización y la persistencia que tuve a lo largo de su elaboración, estos dos puntos se deben de aplicar en cualquier aspecto de la vida para poder cumplir con objetivos que nos planteemos.

BIBLIOGRAFIA.

Alfredo Weitzenfeld Ridel, S. G. (s.f.). Ingeniería de software: el proceso para el desarrollo de software.

Andrés Vignaga, D. P. (2019). *ARQUITECTURAS Y TECNOLOGÍAS PARA EL DESARROLLO DE APLICACIONES WEB*. Obtenido de https://moodle2.unid.edu.mx/dts_cursos_mdl/pos/TI/LP/AM/01/Arquitecturas_y_tecnologias_para_el_desarrollo_de_aplicaciones_web.pdf

ARQUITECTURA DE SOFTWARE Y DISEÑO DE DATOS TRANSACCIONALES Y TRANSFORMACIONALES. (2019). Obtenido de https://virtual.itca.edu.sv/Mediadores/stis/34___arquitectura_de_software_y_diseo_de_datos_transaccionales_y_transformacionales.html

Atlassian. (2019). *Jira Software*. Obtenido de https://www.atlassian.com/es/software/jira

Barry Boehm, A. E. (2019). *Using the WinWin Spiral Model: A Case Study*. Obtenido de http://www.nyu.edu/classes/jcf/g22.3033-007_sp04/handouts/UsingTheSpiralModel.pdf

DISEÑO A NIVEL DE COMPONENTES. (2019). Obtenido de https://virtual.itca.edu.sv/Mediadores/stis/37___diseo_a_nivel_de_componentes.html

EDB Posrgres. (2018). *PostGIS*. Obtenido de https://www.enterprisedb.com/es/enterprise-postgres/postgis

Excentia. (2020). *Automatización de pruebas*. Obtenido de https://www.excentia.es/automatizacion-de-pruebas

Firabase. (2019). *Límites de Realtime Database*. Obtenido de https://firebase.google.com/docs/database/usage/limits?hl=es-419

Firebase. (2019). *Escala con varias bases de datos*. Obtenido de https://firebase.google.com/docs/database/usage/sharding?hl=es-419

Firebase. (2019). *Firebase Realtime Database*. Obtenido de https://firebase.google.com/docs/database/?hl=es-419

Firebase. (2019). *Optimiza el rendimiento de la base de datos*. Obtenido de https://firebase.google.com/docs/database/usage/optimize?hl=es-419

Framework Spring. (2018). *Spring*. Obtenido de https://docs.spring.io/spring/docs/2.5.x/spring-reference.pdf

Francisco José García Peñalvo, A. G. (2019). *Fundamentos de la vista de casos de uso*. Obtenido de https://repositorio.grial.eu/bitstream/grial/1155/1/UML%20-%20Casos%20de%20uso.pdf

Fuentes, D. d. (2011). *Notas del curso: Análisis de requerimientos.*

Geeks, J. C. (2019). *Spring Framework Cookbook.* Java Code Geeks.

IBM. (2018). *Trabajar con diagramas de actividad*. Obtenido de https://www.ibm.com/support/knowledgecenter/es/SSBSK5_7.5.0/com.ibm.rmc.help.doc/topics/c_activity_diagrams.html

IBM. (2019). *IBM Knowledge Center*. Obtenido de https://www.ibm.com/support/knowledgecenter/es/SSZLC2_8.0.0/com.ibm.commerce.developer.doc/concepts/csdmvcdespat.htm

Ionic. (2018). *Ionic Framework*. Obtenido de https://ionicframework.com/docs

Ivar Jacobson, I. S. (2013). *Casos de uso 2.0.*

Jabif, D. (2019). *Ionic Themes*. Obtenido de https://ionicthemes.com/tutorials/about/ionic-application-structure

Lucidchart. (2018). *Tutorial de diagrama de clases UML*. Obtenido de https://www.lucidchart.com/pages/es/tutorial-de-diagrama-de-clases-uml

Lucidchart. (2019). *Tutorial de diagrama de actividades UML*. Obtenido de https://www.lucidchart.com/pages/es/tutorial-diagrama-de-actividades-uml?a=1

Martin, M. M. (2019). *Manual PostGIS*. Obtenido de http://postgis.refractions.net/documentation/postgis-spanish.pdf

Microsoft. (2019). *UML Sequence Diagrams: Reference*. Obtenido de https://docs.microsoft.com/en-us/visualstudio/modeling/uml-sequence-diagrams-reference?view=vs-2015&redirectedfrom=MSDN

Oracle. (2019). *Arquitectura en capas aplicada a los componentes del sistema*. Obtenido de https://docs.oracle.com/cd/E19528-01/820-0888/aaubd/index.html

PMD. (2020). *PMD Source Code Analyzer*. Obtenido de https://pmd.github.io/

PostGIS. (2018). *About PostGIS*. Obtenido de https://postgis.net/

PostGIS. (2018). *Documentation*. Obtenido de https://postgis.net/documentation/

PostGIS. (2018). *Introduction to PostGIS*. Obtenido de http://postgis.net/workshops/postgis-intro/geometries.html

PostGIS. (2018). *PostGIS*. Obtenido de https://postgis.net/features/

S.Pressman, R. (2010). *Ingenierií del Software.* Mc Graw Hill.

Spring. (2018). *Spring Framework Documentation*. Obtenido de https://docs.spring.io/spring/docs/current/spring-framework-reference/

Spring Framework. (2018). *Building REST services with Spring*. Obtenido de https://spring.io/guides/tutorials/rest/

Spring Framework. (2018). *Introduction to the Spring Framework*. Obtenido de https://docs.spring.io/spring/docs/4.2.x/spring-framework-reference/html/overview.html

Subversion. (2019). *Subversion*. Obtenido de https://subversion.apache.org/

Tinoco Gómez, O., Rosales López, P. P., & Salas Bacalla, J. (2010). Criterios de selección de metodologías de desarrollo de software. *Industrial Data*, 70-74.

Universidad de Alicante. (2020). *Casos de prueba: JUnit*. Obtenido de http://www.jtech.ua.es/j2ee/publico/lja-2012-13/sesion04-apuntes.html

Universidad Interamericana para el desarrollo. (2019). *Herrammientas de software*. Obtenido de https://moodle2.unid.edu.mx/dts_cursos_mdl/lic/IEL/HS/S04/HS04_Lectura.pdf