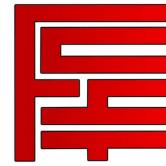




UNIVERSIDAD MAYOR DE SAN SIMÓN
FACULTAD DE CIENCIAS Y TECNOLOGÍA
CARRERA DE INGENIERÍA DE SISTEMAS



DESARROLLO DE UNA APLICACIÓN WEB MÓVIL PARA VISITAS DENTRO EL CAMPUS DE LA UMSS USANDO GEOLOCALIZACIÓN

PROYECTO DE GRADO, PRESENTADO PARA OPTAR
AL DIPLOMA ACADÉMICO DE LICENCIATURA
EN INGENIERÍA DE SISTEMAS.

PRESENTADO POR: Edmundo Figueroa Herbas

TUTOR: Ing. Carlos Alberto Gomez Ormachea

COCHABAMBA - BOLIVIA
DICIEMBRE, 2017

*Para mi Familia, por todo su apoyo
y que siempre estuvo a mi lado.*

*A mi Novia por recordarme,
que no me olvide de seguir avanzando.*

Agradecimientos

Me gustaría agradecer profusamente a mi familia, mis padres, Edwin y Noemí, y mis hermanas, Juana y Carolina, que en todo momento mostraron su interés en verme terminando el proyecto, con la mejor de las ilusiones me ayudaron a que las amanecidas fueran amenas y sin dudar ofrecieron su apoyo, es indudable que sin su ayuda no hubiera concluido este proyecto, muchas gracias.

También un especial agradecimiento a mi tutor, Ing. Alberto Gomez, y a mis colegas, Gimena Mariaca y Álvaro Avila que son sin lugar a dudas, muy buenos amigos que sin pensarlo dos veces ofrecieron su ayuda guiando y muchas veces subrayaron los errores que necesitaban ser corregidos.

Agradezco a los docentes, compañeros de clases y a la Universidad en general, por todas las horas que pasé en esta institución, estudiando, conversando, incluso jugando. La Universidad me enseñó muchas cosas aparte del conocimiento que aprendí en las aulas y de los amigos que forje durante todo el tiempo que pase en la Universidad.

Y finalmente y no menos importante, me gustaría agradecer a mi novia, Silvia, tu ayuda y constante apoyo fueron muy importantes para terminar este proyecto, el cual no fue sencillo pero sin tus palabras de aliento hubiera sido imposible, gracias.

¡Muchas gracias a todos!

Ficha Resumen

En el presente proyecto de grado se trabajó sobre el problema que presenta actualmente la Universidad Mayor de San Simón en relación a la deficiente localización de los lugares que se encuentran dentro del campus Universitario “Las Cuadras”, esto se debe a varios factores, entre los cuales podemos mencionar los escasos mapas con la ubicación geográfica de las aulas u oficinas que se encuentran dentro las distintas facultades o la reestructuración que cada cierto tiempo se lleva a cabo, pero estos mapas no son actualizados.

Para mitigar este problema se planteó la implementación de una aplicación web móvil que pueda localizar un lugar dentro del campus Universitario, mostrando la posición geo-localizada del lugar en un mapa y además la posición actual del usuario, para así poder mostrar la ruta óptima que el usuario puede seguir para poder llegar a su destino, aprovechando las características de geolocalización con que cuentan los dispositivos móviles actuales.

Durante la ejecución del proyecto se analizaron y utilizaron diversas herramientas, como por ejemplo: *EmberJS*, el cual es un entorno de desarrollo *JavaScript*, especializado en el *frontend* o la capa de presentación de la aplicación, *PostgreSQL* como base de datos con *PostGIS* para el procesamiento de datos Geoespaciales, *ExpressJS* para implementar un API o *backend* que maneje la lógica del negocio.

En una primera etapa se construyó un mapa de las rutas internas dentro el campus Universitario, utilizando un dispositivo GPS y recorriendo los caminos peatonales, este mapa resultante tiene las características de un *grafo no dirigido*, sobre el cual se aplica el algoritmo de Dijkstra para obtener el camino más corto entre 2 nodos del grafo, los cuales corresponden al lugar de interés que se está buscando y a la posición del usuario respectivamente.

Posteriormente se implementó la lógica necesaria para poder mostrar al usuario de forma gráfica los lugares disponibles dentro del Campus Universitario, porque al final los datos geoespaciales de los lugares o de la ruta óptima son fácilmente apreciados de forma visual sobre un mapa.

La información obtenida por el uso de la aplicación es bastante valiosa, ya que refleja la relación de personas que desean localizar algún lugar en específico, con la consiguiente posibilidad de tomar decisiones según la cantidad de personas que se mueven por los distintos puntos de interés con los que cuenta la Universidad Mayor de San Simón.

Índice general

Dedicatoria	III
Agradecimientos	V
Ficha Resumen	VII
1. Introducción	1
1.1. Antecedentes	1
1.2. Descripción del problema	2
1.3. Objetivo general	3
1.4. Objetivos Específicos	3
1.5. Justificación	3
1.6. Alcance	3
1.6.1. Alcance Práctico	3
1.6.2. Alcance Metodológico	4
1.6.3. Alcance Teórico	4
2. Marco Teórico	5
2.1. Geolocalización	5
2.1.1. Sistema de Coordenadas para datos Geográficos	6
2.1.2. Coordenadas Geocéntricas (X,Y,Z)	6
2.1.3. Coordenadas Geográficas	7
2.1.4. Coordenadas Proyectadas	7
2.2. Ruta Óptima	9
2.2.1. Grafos	10
2.2.2. Matriz de adyacencias de un Grafo	10
2.2.3. El Problema de la ruta mas corta	11

2.3.	Aplicaciones Móviles	12
2.3.1.	Aplicaciones Nativas	12
2.3.2.	Aplicaciones Web	12
2.3.3.	Aplicaciones Híbridas	13
2.4.	Aplicación de Página Única	13
2.4.1.	MVC	15
2.4.2.	Arquitectura de una Aplicación de Página Única	15
2.5.	REST API	16
2.5.1.	Servicio Web	17
2.5.2.	REpresentational State Transfer (REST)	17
2.6.	NodeJS	19
2.7.	ExpressJS	20
2.7.1.	Middleware	20
2.7.2.	Routing	20
2.8.	Ember JS	20
2.9.	Base de Datos	22
2.9.1.	PostgreSQL	22
2.9.2.	PostGIS	23
3.	Metodología de Desarrollo	25
3.1.	Introducción	25
3.2.	Metodologías Ágiles	25
3.3.	Programación Extrema	26
3.3.1.	Las historias de usuario	27
3.3.2.	Proceso de desarrollo	28
3.3.3.	Roles de la programación extrema	28
3.4.	El Proceso XP	29
3.4.1.	Exploración	29
3.4.2.	Planificación de la Entrega	30
3.4.3.	Iteraciones	30
3.4.4.	Producción	32
4.	Campus Universitario	33
4.1.	Introducción	33

4.2.	Facultad de Arquitectura y Ciencias del Hábitat	35
4.3.	Facultad de Ciencias Jurídicas y Políticas	36
4.4.	Facultad de Ciencias Económicas	38
4.5.	Facultad de Humanidades y Ciencias de la Educación	40
4.6.	Facultad de Ciencias y Tecnología	42
4.7.	Multiacademico	44
4.8.	Complejo Deportivo	46
4.9.	Facultades fuera del campus Universitario	47
5.	Implementación del Proyecto	49
5.1.	Exploración	49
5.1.1.	Identificación de Usuarios	49
5.1.2.	Requerimientos Funcionales	49
5.1.3.	Historias de Usuario	50
5.2.	Planificación de la Entrega	51
5.2.1.	Plan de Entregas	51
5.3.	Iteración 1	53
5.3.1.	Planificación	53
5.3.2.	Diseño	55
5.3.3.	Implementación	58
5.3.4.	Pruebas de Aceptación	63
5.4.	Iteración 2	66
5.4.1.	Planificación	66
5.4.2.	Diseño	69
5.4.3.	Implementación	71
5.4.4.	Pruebas de Aceptación	81
5.5.	Iteración 3	84
5.5.1.	Planificación	84
5.5.2.	Diseño	87
5.5.3.	Implementación	88
5.5.4.	Pruebas de Aceptación	92
5.6.	Iteración 4	95
5.6.1.	Planificación	95
5.6.2.	Diseño	99

5.6.3. Implementación	102
5.6.4. Pruebas de Aceptación	105
5.7. Producción	108
6. Conclusiones y Recomendaciones	111
6.1. Conclusiones	111
6.2. Recomendaciones	112
Bibliografía	113
Anexos	114
A. Manual Instalación	117
A.1. Instalación de la base de datos	117
A.2. Configuración de la base de datos	118
A.2.1. Añadir las funciones de PostGIS y pgRouting a la base de datos	118
A.2.2. Habilitar el acceso a la base de datos	119
A.3. Configuration de la aplicación UBIKATE UMSS	119
A.3.1. Instalación de las dependencias	119
A.3.2. Copiar los archivos del proyecto UBIKATE UMSS	120
A.3.3. Configurar el mapa topológico del campus Universitario	120
A.3.4. Instalar las dependencias del proyecto UBIKATE UMSS	120
A.3.5. Iniciar el servidor <i>ExpressJS</i>	121
B. Manual Usuario	123
B.1. Página de Inicio	123
B.2. Menu de la Aplicación	124
B.3. Lista de Lugares	124
B.4. Formulario de registro de un Lugar	126
B.5. Información del Lugar	126
B.6. Formulario de edición de un Lugar	128
B.7. Formulario de registro de un Usuario	128
B.8. Formulario de ingreso al sistema	129
B.9. Reporte	130

Índice de Figuras **131**

Índice de Tablas **135**

Capítulo 1

Introducción

El presente proyecto consiste en el desarrollo de una aplicación móvil que permita ubicar y encontrar una locación dentro del campus de la Universidad Mayor de San Simón, la aplicación deberá localizar la ubicación actual del usuario y permitir especificar un punto de destino, mostrando a continuación el camino más corto para llegar a destino.

El campus universitario abarca más de $214,000\ m^2$ y encierra varias facultades y oficinas administrativas, para estudiantes nuevos y antiguos o personas que necesitan hacer trámites administrativos, incluso si solo se quiere conocer el campus, es necesario contar con un mapa donde ubicarse.

Las aplicaciones móviles tienen una gran demanda por parte de la población ya que la gran mayoría posee un *smartphone* o teléfono inteligente con capacidad de ejecutar aplicaciones muy fácilmente, los *smartphones* cuentan también con GPS, el cual se usa para conocer la ubicación del usuario con un margen de error de 3 metros, usando puntos de referencia geo-localizados se puede determinar la ruta óptima para llegar a destino. Es una desventaja para nuestra Universidad que no exista información confiable de fácil acceso para poder desplazarse por el campus.

1.1. Antecedentes

Actualmente *Google Maps* ofrece una solución al problema de encontrar una ruta entre 2 puntos geolocalizados, ya que sugiere posibles rutas si se usara movilidad, bicicleta o para ir caminando, para lograr esto se toman en cuenta los distintos tipos de calles que existen y la dirección en el caso de movilidades, *Google Maps* toma en cuenta la descripción de una locación o la referencia cartográfica en latitud y longitud de los puntos, y el cómo se va a desplazar entre los 2 puntos para dibujar con una línea roja la ruta a seguir.

Así como también existen Blogs o Aplicaciones con información de los lugares turísticos o de interés para visitar en la ciudad, como ser TripAdvisor, la información que provee esta aplicación generalmente incluye la locación del lugar referenciada sobre un mapa estático, este tipo de aplicaciones usa el API de *Google Maps* para lograr encontrar una ruta hacia el lugar de interés.

En el caso del campus de la Universidad Mayor de San Simón, Google Maps no cuenta con

la información para lograr este objetivo, de encontrar una ruta entre 2 puntos geo-referenciados, ya que se necesita de un mapa de los caminos internos del campus Universitario e información de las aulas, kioscos, fotocopiadoras, oficinas, etc. Esta información no está disponible o es de difícil acceso lo cual genera malestar cuando se busca una locación dentro del campus Universitario.

1.2. Descripción del problema

La Universidad Mayor de San Simón no cuenta con un mapa interactivo que muestre la ubicación de los puntos o lugares que se encuentran dentro del campus universitario y como llegar hasta su ubicación, la falta de señalización obstaculiza el desplazamiento de los estudiantes o personas que requieren encontrar alguna oficina para, por ejemplo, realizar trámites administrativos, encontrar aulas o auditorios, etc. como resultado se pierde tiempo al tratar de encontrarlos por lo que un mapa con estas características sería de gran ayuda para el desplazamiento dentro del campus universitario. En la figura 1.1, se puede apreciar al árbol de problemas.

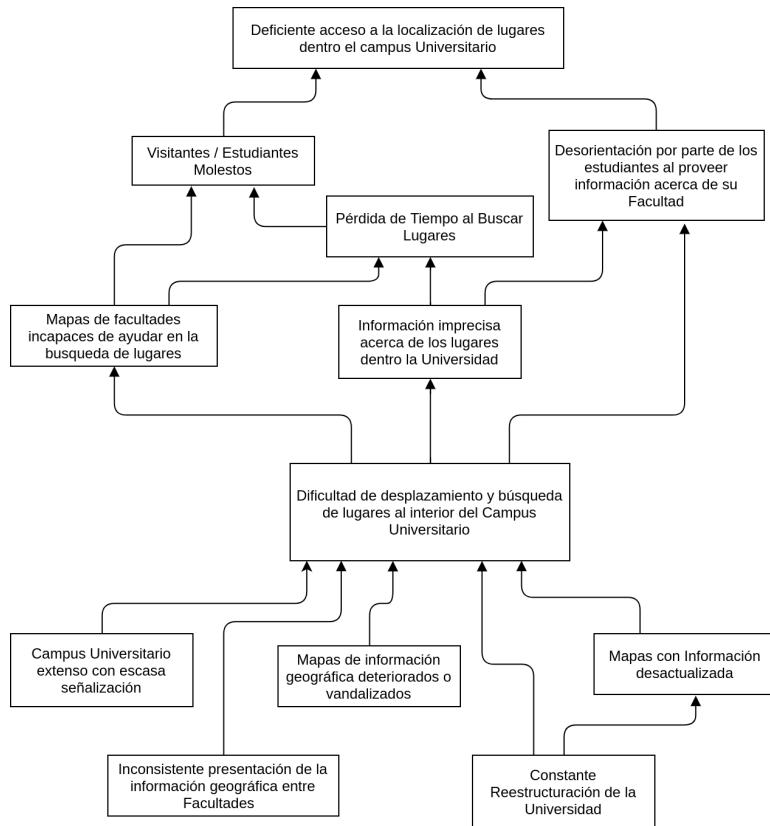


Figura 1.1: Diagrama Árbol de Problemas

Fuente: Elaboración propia

1.3. Objetivo general

Desarrollar una aplicación web móvil *responsiva* para optimizar la búsqueda de lugares y el desplazamiento al interior del Campus Universitario de la UMSS.

1.4. Objetivos Específicos

- Generar un mapa con información geográfica de las rutas dentro del campus Universitario.
- Administrar lugares geolocalizados dentro del campus Universitario.
- Mostrar en la aplicación los lugares geolocalizados desplegando la ruta óptima desde mi posición hasta el punto destino.
- Administrar usuarios en el sistema.
- Registrar las búsquedas sobre rutas realizadas por los usuarios en el sistema.

1.5. Justificación

El Campus Universitario es bastante extenso y se encuentra en constante reestructuración, debido a que las aulas se incrementan, las oficinas son reubicadas, etc. gracias a esto es que los mapas con los que cuenta cada facultad, que son escasos y están impresos sobre banners estáticos, son también difíciles de actualizar. Este hecho genera malestar en estudiantes que llegan tarde a sus clases o necesitan llegar a algún Auditorio o personas/visitantes en proceso de realizar trámites administrativos, no encuentran con facilidad las oficinas a las que necesitan llegar.

Una aplicación que permita localizar o encontrar locaciones y además proveer la ruta óptima dentro del campus de la Universidad Mayor de San Simón es de gran importancia para brindar apoyo a cualquier persona que necesite desplazarse por el campus Universitario.

Las Aplicaciones móviles y/o web demostraron ser el futuro del desarrollo de software y la gran mayoría de los países en el mundo consumen estas soluciones y es necesario apuntar a esta tendencia.

1.6. Alcance

1.6.1. Alcance Práctico

Una aplicación web móvil puede llegar a ser muy compleja y manejar información sensible, y ya que el servidor está expuesto al acceso público de los usuarios, es susceptible de ataques maliciosos y malintencionados para acceder y robar información privada que los usuarios podrían tener almacenados en la aplicación, en el caso de la presente aplicación, el sistema no manejará información sensible del usuario, como ser tarjetas de crédito pero la aplicación

manejará información de lugares, información que podría ser corrompida por usuarios malintencionados. La seguridad es muy importante para una aplicación web, por lo cual el presente proyecto implementara medidas de seguridad para asegurar la identidad del usuario que está solicitando el ingreso al sistema pero no incluirá protección a ataques Phishing, DoS ya que los objetivos específicos no los contempla.

El *look and feel* de una aplicación web es un tema muy importante para cualquier aplicación a desarrollar, para lograr que la aplicación se muestre de manera consistente en la pantalla de un smartphone se usarán herramientas de terceros pero no se extenderá el uso de la misma para la pantalla de un ordenador de escritorio que posee una resolución de pantalla muy superior al de un celular.

1.6.2. Alcance Metodológico

Para la conclusión exitosa del presente proyecto se implementará la metodología Programación Extrema (XP) y cada iteración del proceso tiene como meta el desarrollo conjunto de diferentes módulos, historias de usuario y la documentación relacionada.

1.6.3. Alcance Teórico

La investigación se limita a las estructuras, herramientas y estándares actuales sugeridos en la documentación y bibliografía consultada para la construcción de una aplicación web móvil.

Capítulo 2

Marco Teórico

2.1. Geolocalización

La Geolocalización o Georreferenciación es un término bastante nuevo, de hecho no aparece en el diccionario de la Real Academia Española, no obstante se lo puede definir como:

El posicionamiento en el que se define la localización de un objeto espacial (representado mediante un punto, vector, área, volumen) en un sistema de coordenadas y datum determinado. Este proceso es utilizado frecuentemente en los Sistemas de Información Geográfica. (*Realidad Aumentada Georeferenciada*, 2013)

La Georreferenciación antiguamente era bastante usada en el ámbito científico, y se necesitaba de instrumental y personal cualificado para su manejo, pero en la actualidad la cantidad de dispositivos con capacidad para geolocalizar un objeto sobre la tierra es bastante común, de hecho todos los smartphones actuales (en general los que se consideran gama media o alta) traen integrados receptores GPS (Global Position System), y sumados a la explosión de aplicaciones que integran mapas con localización, ya que se puede tener una base de datos con coordenadas, descripciones, etc., que individualmente no aporta mucho valor pero al obtener datos de una gran cantidad de usuarios puede llegar a ser información valiosa ya que sirve para tomar decisiones a nivel de negocio, pero interpretar estos datos sería muy difícil sin la ayuda de los *Sistemas de Información Geográfica* o *SIG*, un SIG es una herramienta que permite integrar, analizar, mostrar, interpretar y entender las relaciones, patrones y tendencias de la información geográficamente referenciada . (*What is GIS?*, 2017)

Por estas razones es que actualmente existe una explosión de estas aplicaciones donde empresas particulares y hasta organismos gubernamentales están haciendo uso de estas tecnologías. Y las posibilidades son diversas, por ejemplo, si se quisiera planificar la construcción de un colegio se podría integrar los datos del censo con un mapa, identificando los sectores con mayor porcentaje de niños y localizando los sectores más propicios para realizar la construcción del inmueble. En el caso de una catástrofe natural, el tener las rutas de evacuación geolocalizadas y disponibles en un mapa de manera eficiente, ayudaría en la evacuación de las personas del lugar.

La geolocalización es actualmente una tecnología y una herramienta usada en gran medida

por una gran cantidad de aplicaciones web, añadiendo búsquedas y resultados personalizados a nivel país, ciudad, barrio y calle, resultando en una gran variedad de servicios y que actualmente es de gran ayuda en diferentes escenarios. La geolocalización ayuda a moverse por una ciudad, encontrar restaurantes, cines, transporte, etc. actualmente es una de las herramientas mas usadas y desarrolladas a nivel de industria, comercio, turismo, etc. y vale la pena estudiarla y entenderla.

En la aplicación desarrollada se requerirá trabajar con datos espaciales, y para ello es necesario entender algunos conceptos envueltos en el manejo de la información geográfica.

- **Coordenada:** Es una secuencia de n-números que designa la posición de un punto en un espacio n-dimensional.
- **Sistema de coordenadas:** Un sistema de coordenadas es un conjunto de reglas matemáticas que especifican cómo las coordenadas son asignadas a cada punto.
- **Punto:** Es la representación de una posición, topológicamente 0-dimensional (no tiene volumen, área, longitud o cualquier otra unidad multi-dimensional).

Estas definiciones están desarrolladas en la especificación *Simple Feature Access*, la cual es mantenida por la OGC (Open Geospatial Consortium). Esta especificación define el conjunto de tipos de datos (puntos, línea, polígono, etc) y las operaciones o métodos necesarios para manejar estos datos. (Herring, 2011, p. 8–11)

2.1.1. Sistema de Coordenadas para datos Geográficos

Se podría pensar en un sistema de coordenadas como la forma de dar sentido a un *par de coordenadas*, por ejemplo `POINT(-66.1457475 -17.3937285)`, pero cómo se interpretan estos números?. Podría ser la latitud y longitud del campus de la UMSS, o algún punto en el Océano Pacífico. Es gracias al sistema de coordenadas que se puede ubicar este punto en el Universo.

Una aplicación que maneja datos geográficos tiene que trabajar con sistemas de coordenadas relacionadas con la superficie terrestre, conocidas como coordenadas espaciales o coordenadas globales, que permiten representar la tierra en *3-Dimensiones* (3D), ya que esta es una Esfera, más específicamente un esferoide oblato¹, o en una representación de la superficie terrestre en *2-Dimensiones* (2D), los sistemas de coordenadas se clasifican en: Coordenadas Geocéntricas, Coordenadas Geográficas y Coordenadas Proyectadas. (Azuma, 2011)

2.1.2. Coordenadas Geocéntricas (X,Y,Z)

También conocido como *Coordenadas Cartesianas 3D*, Este sistema tiene como origen el centro de la Tierra, con el *eje X* y el *eje Y* en el plano del ecuador. El *eje X* pasa a través del meridiano de Greenwich, y el *eje Z* coincide con el eje de rotación de la Tierra. como se puede ver en la figura 2.1.

¹Un *esferoide oblato* (o elipsoide oblato) es un elipsoide de revolución obtenido por rotación de una elipse alrededor de su eje más corto.

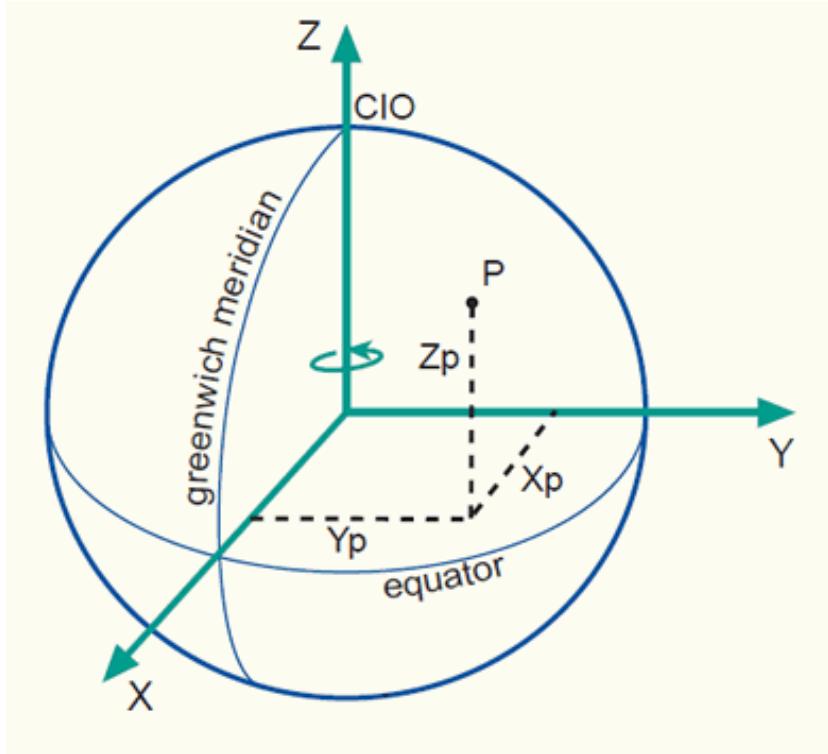


Figura 2.1: Sistema de coordenadas Geocéntricas

Fuente: (Knippers, 2009)

Este sistema de coordenadas no es muy usado en la representación de datos, pero a veces se lo requiere para análisis de algoritmos y geometría computacional.

2.1.3. Coordenadas Geográficas

El sistema de coordenadas Geográficas, ver figura 2.2, utiliza las coordenadas angulares latitud (ϕ o θ) y longitud (λ). Este sistema de coordenadas se expresa en grados, se lo puede representar con la forma *grados:minutos:segundos* ($17^{\circ} 23' 37.4226''$ S, $66^{\circ} 8' 44.691''$ W), o de la forma más común *grados decimales* (-66.1457475 S, -17.3937285 W).

Dentro de ese sistema de coordenadas, está el que es el más ampliamente usado y el que usan por defecto los sistemas *GPS*, es el denominado “WGS84” (*World Geodetic System 1984*) y es el que generalmente la mayoría de las aplicaciones usan para el manejo de mapas. Es el sistema que maneja los más ampliamente conocidos “latitud y longitud”.

2.1.4. Coordenadas Proyectadas

Un sistema de coordenadas proyectadas es una representación plana y bidimensional de la tierra. Se basa en un sistema de coordenadas *geográficas esféricas*, pero utiliza unidades de

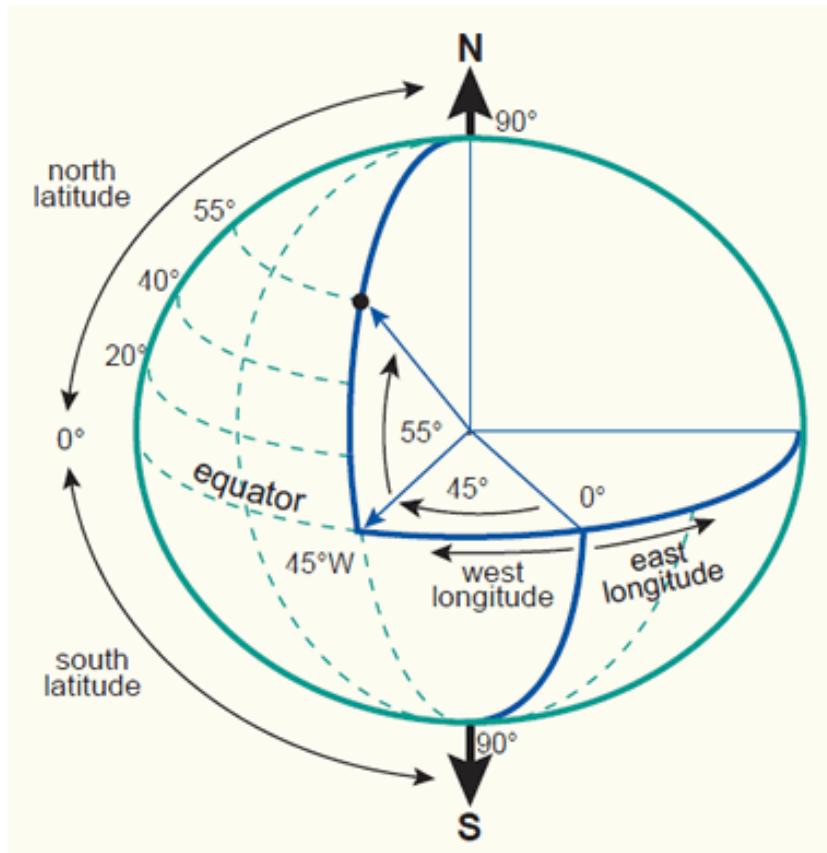


Figura 2.2: Sistema de coordenadas Geográficos

Fuente: (Knoppers, 2009)

medida lineales para las coordenadas, de forma que los cálculos de distancia y área se pueden realizar en términos de esas mismas unidades. (Végső, 2010)

Un sistema de coordenadas proyectadas requiere tomar la superficie esférica de la tierra y “aplanarla”, este procedimiento se lo realiza con la finalidad de tener un mapa representable en una hoja de papel así como en la pantalla de la computadora. Sin embargo este procedimiento introduce diversos tipos de distorsión por lo que existen diferentes clases de proyecciones que varían según la región de la Tierra que se quiere representar.

La proyección que usan por ejemplo, *Google Maps* y *Open Street Maps* es la *Mercator Projection*, esta proyección está diseñada para preservar los ángulos y las formas de las líneas en forma recta, pero distorsiona los tamaños y las distancias mientras más lejos se encuentran de la línea del Ecuador. Esta proyección se puede apreciar en la figura 2.3. (Topf y Hormann, 2015)

Tal como se puede apreciar en la figura 2.3, la distorsión de esta proyección se hace evidente si se observa la zona de Groenlandia ya que parecería tan grande como África o América del Sur, cosa que no es cierta, ya que Groenlandia es casi 14 veces más pequeño que África. A pesar de esta distorsión tan marcada, la *Proyección de Mercator* es una de las más usadas.



Figura 2.3: Sistema de coordenadas Proyectadas

Fuente: (Knippers, 2009)

2.2. Ruta Óptima

En general el mejor camino o el más óptimo para ir de un punto a otro es aquel que toma menos tiempo en ser recorrido, pero para definir que un camino es óptimo hay que tomar en cuenta las características de este, por ejemplo, si se va en coche hay que tomar en cuenta la dirección de las calles, los cruces, etc. si se va caminando hay que ver las características del terreno, caminos cortados, distancias, etc.

Encontrar la ruta óptima entre 2 puntos es un problema al cual se enfrentan las personas diariamente, por ejemplo, las empresas de transporte, de correo, etc., necesitan mejorar la eficiencia del trayecto y a la vez reducir el consumo de combustible, para mejorar la atención al cliente y a la vez reducir costos de operación.

Dentro el campus Universitario es generalmente prioritario optimizar el tiempo de búsqueda de algun lugar o punto de interés al cual se quiera llegar, ya sea como estudiante o visitante externo.

Si se analiza el terreno que se va a cubrir con la aplicación, el campus “Las Cuadras” de la Universidad Mayor de San Simón ubicado entre las calles Oquendo, Sucre, Belzu y M. U. López, se tiene que el camino óptimo es siempre el más corto o de menor longitud, el método de desplazamiento que se tomara en cuenta será *caminando*, el terreno es llano y la única restricción es respetar las rutas peatonales.

El presente problema se lo podría definir como, encontrar la ruta más corta de un punto a otro punto, donde los puntos están interconectados por una red de caminos. El problema descrito se lo puede resolver y describir como un caso específico de la teoría de grafos.

2.2.1. Grafos

Inicialmente es necesario aclararar algunos términos usados en la teoría de grafos.

- **Grafo:** Un *grafo* G consiste en un conjunto de vértices V y un conjunto de aristas A , y se lo representa con $G(V, A)$.
- **Vértice:** El *vértice* v es adyacente a u , o a un vecino de u , si y sólo si $(u, v) \in A$. Los vértices también son llamados nodos.
- **Arista:** Cada *arista* o arco es representada por un par de elementos (u, v) , donde los elementos $u, v \in V$, son los nodos que une la arista.
- **Grafo no Dirigido:** En un *grafo no dirigido* G , dos vértices u y v se dice que están conectados si hay un camino en G de u a v (y cómo G no es dirigido, también hay un camino de v a u). Un grafo se denomina completo si para todos los pares $u, v \in V$ existe una arista $(u, v) \in A$.
- **Grafo Dirigido:** En cambio en un *grafo dirigido* las aristas (u, v) y (v, u) representan dos diferentes aristas. También se puede anotar un tercer componente, llamado peso o costo, en ese caso estaríamos hablando de un *grafo ponderado*.

Existen diversas formas de representar un grafo sea dirigido o no-dirigido, pero entre las mas usadas están la matriz de adyacencias y la lista de adyacencias.

2.2.2. Matriz de adyacencias de un Grafo

Sea $G = (V, A)$ un grafo de n vértices. La matriz de adyacencias M para G es una matriz $M_{n \times n}$ de valores booleanos, donde $M(i, j)$ es verdad si y sólo si existe un arco desde el nodo i al nodo j .

$$M(i, j) = \begin{cases} 1, & \text{si existe la arista } (i, j) \\ 0, & \text{en caso contrario} \end{cases}$$

Las filas y las columnas de la matriz representan los nodos del grafo. Cuando el grafo es no-dirigido la matriz de adyacencias es simétrica, como se puede ver en el grafo de la figura 2.4 y su correspondiente matriz de adyacencias.

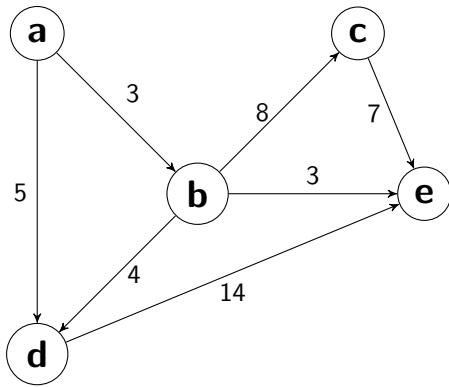


Figura 2.4: Grafo ponderado no-dirigido

Fuente: Elaboración propia

$$M(i, j) = \begin{pmatrix} & a & b & c & d & e \\ a & 0 & 3 & 0 & 5 & 0 \\ b & 3 & 0 & 8 & 4 & 3 \\ c & 0 & 8 & 0 & 0 & 7 \\ d & 5 & 4 & 0 & 0 & 14 \\ e & 0 & 3 & 7 & 14 & 0 \end{pmatrix}$$

Matriz de adyacencias del grafo de la figura 2.4

2.2.3. El Problema de la ruta más corta

Dados los vértices v_i y v_j de un grafo $G = (V, A)$ se llama trayectoria mínima o camino mínimo de v_i a v_j al número de aristas del camino de longitud mínima que va desde v_i a v_j y se representa por $d(v_i, v_j)$.

Cuando en el grafo no exista un camino de v_i a v_j se dice que el camino mínimo es $d(v_i, v_j) = \infty$

Para determinar el camino mínimo que va desde un único vértice a cualquier otro vértice se puede usar el algoritmo de Dijkstra.

2.2.3.1. Algoritmo de Dijkstra

El algoritmo de Dijkstra fue descrito en 1959 por *Edsger Dijkstra*, y permite encontrar la trayectoria más corta entre dos nodos específicos, cuando los valores de los arcos son todos positivos

El algoritmo asigna un etiqueta a cada nodo en el grafo. Esta etiqueta es la distancia que

hay desde el nodo s escogido como origen a lo largo de la trayectoria más corta encontrada, hasta el nodo que se está etiquetando.

La etiqueta de cada nodo puede estar en 2 estados:

- a. Puede ser permanente; en este caso la distancia encontrada es a lo largo de la trayectoria, la más corta de todas las encontradas.
- b. Puede ser temporal; cuando hay incertidumbre de que la trayectoria encontrada sea la más corta de todas.

A medida que el método trabaja se cambian gradualmente las etiquetas temporales por etiquetas permanentes. Al comienzo se tiene un conjunto de nodos con etiquetas temporales y el objetivo es hacer que esas etiquetas disminuyan, encontrando trayectorias a esos nodos usando trayectorias a nodos etiquetados permanentemente. Cuando esto se ha logrado, se selecciona el nodo con la etiqueta temporal más pequeña y esta etiqueta se convierte en permanente. El proceso se repite hasta que al nodo terminal t se le haya asignado una etiqueta permanente, pero esto puede ocurrir eventualmente, ya que cada vez que el algoritmo es usado, una de las etiquetas es omitida y así el número de nodos con etiquetas temporales decrece a cero. (Barrero, de Garcia, y Parra, 2010)

2.3. Aplicaciones Móviles

El desarrollo de aplicaciones web se divide principalmente en 3 grupos o enfoques de desarrollo Aplicaciones Nativas, Aplicaciones Web y Aplicaciones Híbridas.

2.3.1. Aplicaciones Nativas

Las aplicaciones nativas se caracterizan de poder acceder directamente al sistema operativo móvil sin ningún intermediario ni contenedor.

La aplicación nativa puede acceder libremente a todas las APIs (*Application Program Interface* es un conjunto de herramientas, protocolos y rutinas que son usados para desarrollar aplicaciones, un API específica como tienen que interactuar los componentes de un sistema.) que el proveedor del Sistema Operativo (SO) ponga a disposición y en muchos casos, tiene características y funciones únicas que son típicas del SO móvil en particular.

Este tipo de aplicaciones se adapta al 100 % con las funcionalidades y características del dispositivo obteniendo así una mejor experiencia de uso.

2.3.2. Aplicaciones Web

Los dispositivos móviles modernos pueden ejecutar navegadores con capacidad de ejecutar HTML5 (*Hiper Text Markup Language* el cual es el lenguaje para escribir páginas Web), la cual es la Versión de HTML publicado en Octubre 2014, es la más moderna y en la que se escriben

todas las aplicaciones web actuales, así como también incorporan un motor JavaScript que permite ejecutar código para lograr una página Web dinámica. Algunos ejemplos del potencial de HTML5 son: componentes IU avanzados, acceso a múltiples tipos de medios, servicios de geoposicionamiento y disponibilidad offline. Al emplear estas características se puede crear aplicaciones avanzadas usando únicamente tecnologías basadas en la Web.

Se debe distinguir entre las aplicaciones Web, las aplicaciones Web diseñadas para dispositivos móviles ya que estas últimas reconocen cuando se accede a través de un smartphone y despliegan una página HTML que fue diseñada para brindar una experiencia táctil y cómoda en una pantalla pequeña, a este diseño de aplicación se le conoce como aplicación web responsive, esto mejora la experiencia del usuario creando un sitio Web móvil que se parezca a una aplicación nativa.

2.3.3. Aplicaciones Híbridas

El enfoque híbrido combina desarrollo nativo con tecnología Web. Usando este enfoque, se escribe gran parte de la aplicación usando tecnologías Web y se mantienen el acceso directo a APIs nativas cuando se necesita. La porción nativa de la aplicación emplea APIs del sistema operativo para crear un motor de búsqueda HTML incorporado que funciona como un puente entre el navegador y las APIs del dispositivo. (WebSphere, 2012)

Esto permite que la aplicación híbrida aproveche todas las características que ofrecen los smartphones modernos. Para lograr esto existen bibliotecas tal como Apache Cordova (antiguamente conocido como *PhoneGap*, es una de las herramientas más populares para crear aplicaciones híbridas.) que provee una interfaz JavaScript con funcionalidad para conectarse con los dispositivos seleccionados y lograr manejar el API propio del *smartphone*.

La porción Web de la aplicación puede ser una página Web que resida en un servidor o bien un conjunto de archivos HTML, JavaScript, CSS y contenido multimedia, incorporados en el código de la aplicación y almacenados localmente en el dispositivo. (WebSphere, 2012)

2.4. Aplicación de Página Única

Una aplicación de página única, en inglés *single-page application* o SPA es una aplicación que cabe en una sola página con el propósito de dar una experiencia más fluida a los usuarios lo más cercano posible a una aplicación de escritorio. (*Single-page application*, 2017)

Una Aplicación de Página Única es básicamente la combinación de AJAX y HTML5 para crear una aplicación web fluida que no necesite recargar la página, esto significa que el trabajo sucede en lado del cliente, es decir en el browser, por lo tanto es necesario de un gran conocimiento de JavaScript, pero actualmente existen Frameworks diseñados para resolver los problemas de manejar JavaScript en diferentes tipos y versiones de navegadores. *Ember.js* es

un framework con el cual se puede construir Aplicaciones de Página Única.

En una aplicación web tradicional cada vez que la aplicación hace una petición al servidor, este renderiza una nueva página HTML la cual es desplegada en el navegador.

En una Aplicación de Página Única, después que la primera página es cargada en el navegador, todas las demás interacciones del usuario con la aplicación generan peticiones AJAX al servidor, las cuales retornan los datos requeridos, generalmente en formato JSON, de esta forma se actualiza solamente porciones de la página con la información obtenida del servidor, sin necesidad de recargar la página entera, como se puede ver en la figura 2.5.

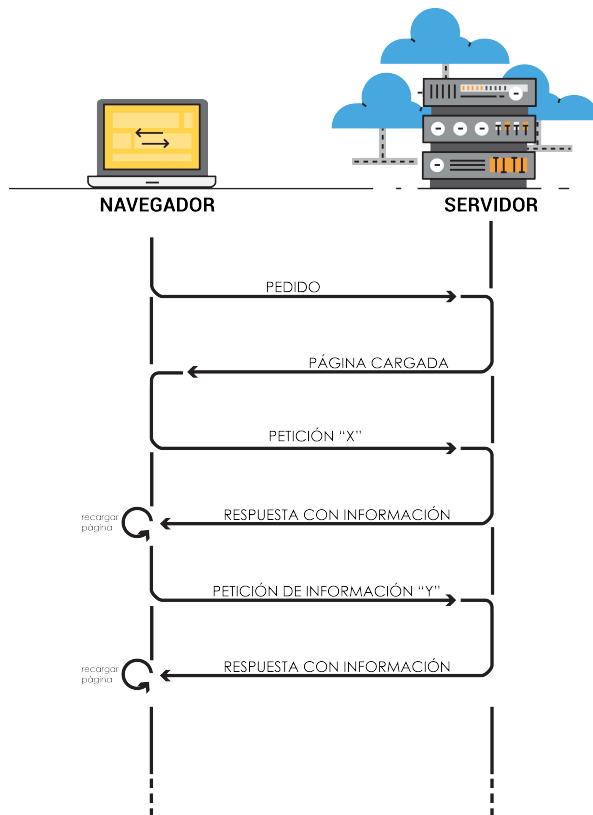


Figura 2.5: Flujo de una Aplicación de Página Única.

Fuente: Elaboración propia

Esto obliga que la lógica de la aplicación con la que el usuario interactúa se cargue en la primera y única vez que la aplicación es cargada, evitando con esto la molestia de ver como se recarga la página con cada interacción del usuario con la aplicación, la cual se siente fluida y lo más parecida a una aplicación de escritorio de lo que una aplicación web puede llegar a ser.

En una aplicación de página única, las interacciones del usuario con la aplicación se desarrollan en el navegador y el servidor se encarga de manejar la información, en esta arquitectura

se puede observar una separación lógica entre la capa de la presentación de la aplicación y el modelo de negocio o la lógica de la aplicación. Por lo tanto se puede apreciar que es básicamente una implementación del Patrón de Diseño *MVC*.

2.4.1. MVC

MVC (Modelo Vista Controlador) es un patrón arquitectónico que separa los datos de la aplicación en la interfaz del usuario y la lógica del negocio, en tres partes cada uno especializado para su tarea, la vista maneja lo que es la interfaz del usuario, puede ser gráficamente o solo texto, el controlador interpreta las entradas del teclado, mouse, o los cambios de la vista de la mejor forma posible y finalmente el modelo maneja el comportamiento de los datos de la aplicación. (Burbeck, 1992)

Este concepto se desarrolló en 1979 por Trygve Reenskaug el cual da una solución al problema de separar la lógica del negocio de la lógica de la presentación.

- **Modelo:** Representa la información o los datos y contiene las reglas o métodos para manipular estos datos. El Modelo es el principal encargado de manejar la *lógica del negocio*.
- **Vista:** Representa la interfaz de la aplicación.
- **Controlador:** Es el encargado de interactuar entre el modelo y la vista, procesando los datos enviados en el request del navegador web, llamando a métodos del modelo para conseguir *información* de la base de datos, posteriormente el controlador envía esta *información* a la vista. El Controlador y la Vista son los encargados de manejar la *lógica de la presentación*.

2.4.2. Arquitectura de una Aplicación de Página Única

La separación de la logica de las capas de la aplicación, implica que su implementación sea mas entendible, por lo que en una aplicación de página única bien diseñada es posible cambiar o rediseñar la vista o la presentación de la aplicación sin tocar la implementación de la lógica de la aplicación y/o viceversa.

La arquitectura de un SPA, como se puede ver en la figura 2.6, donde el *DOM* (*Document Object Model*, es la estructura básica de la página desplegada en el navegador) es de *solo escritura*, ya que no tiene estado o datos almacenados, cuando la página contiene estado o comúnmente la sesión del usuario, es necesario contener y propagar esta *sesión* por las páginas de la aplicación, donde generalmente son usados los *cookies* pero cuando la aplicación se complica y contiene diferentes niveles de permisos, el manejo de esta información se complica, por lo que en vez de guardar la información en objetos aleatorios dentro del DOM es necesario una capa de la aplicación donde estén definidos los objetos que contienen la información y el estado de la aplicación, el cual es el *Modelo*, esta capa también es la encargada guardar y/o extraer

los datos del *Storage*, que dependiendo de la aplicación puede ser una base de datos o el mismo navegador, *localStorage* es una implementación de HTML5 que permite almacenar información en objetos JSON en el Navegador, la *Vista* o *View* se encarga de observar los cambios ocurridos en el *Modelo* y reflejar estos cambios en el *DOM*, el cual es una renderización de un *Template*.

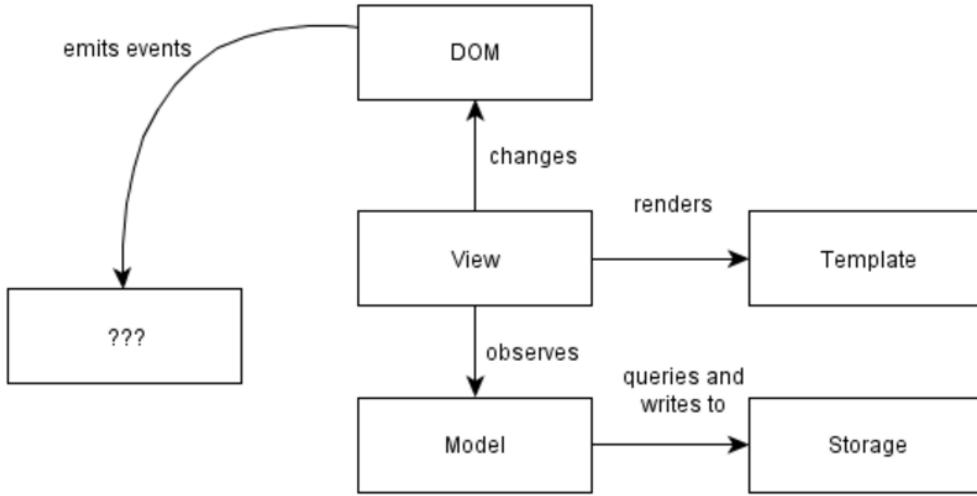


Figura 2.6: Arquitectura de una Aplicación Web Moderna

Fuente: (Beck, 1999)

Como se puede observar en la figura 2.6, la arquitectura de un SPA es claramente una implementación del patrón arquitectónico MVC, que en este caso en *Controlador* como una capa específica está desapareciendo, por esa razón no aparece en el diagrama, porque cuando existe algún cambio en el *DOM*, este cambio también se registra en la *Vista* y ya que el *Modelo* tiene el control de la información, la implementación de la lógica del negocio se puede repartir entre estas 2 capas. El patrón arquitectónico MVC no es inmutable y se lo puede implementar dependiendo del tipo de aplicación.

En la arquitectura de un SPA la lógica de la aplicación es implementada, generalmente y en el presente proyecto, en un API y la capa de la presentación al estar en el navegador es implementada con una combinación de HTML, CSS y JavaScript, lo cual es bastante complejo, pero gracias a frameworks como ser *BackboneJS*, *AngularJS*, *EmberJS*, etc. Los cuales agilizan en gran medida la implementación de una aplicación de página única.

2.5. REST API

REST API, es como se denomina generalmente a un Web API o Web Service basado en el estilo arquitectónico REST. Pero qué es un Servicio Web?

2.5.1. Servicio Web

Los Servicios Web o Web Services, según la W3C “proveen un medio estándar de interoperabilidad entre diferentes aplicaciones de software, que se ejecutan en una variedad de plataformas y/o frameworks”. (Lafon, 2002)

Los servicios web se caracterizan por su gran interoperabilidad y extensibilidad, los Servicios Web más implementados son los basados en RPC, SOA y REST.

RPC: *Remote Procedure Calls*, en español, Llamadas a Procedimientos Remotos. Es un tipo de protocolo que permite a un programa en una computadora ejecutar un procedimiento en un servidor. El Cliente envía un mensaje al Servidor un mensaje con los argumentos necesarios y el servidor responde con un mensaje con los resultados del procedimiento ejecutado.

Los Web Services basados en el modelo RPC fueron los primeros en ser usados en la web por lo que su uso está bastante extendido. Suele ser implementado por medio del mapeo de servicios directamente a funciones específicas o llamadas a métodos.

SOA: *Service-oriented Architecture*, en español, Arquitectura Orientada a Servicios. Es una arquitectura de aplicación en donde todas las funciones o servicios están definidas usando un lenguaje descriptivo, XML es el lenguaje elegido para el intercambio de mensajes. Cada interacción es independiente de todas las demás interconexiones y los protocolos de comunicación entre dispositivos, por lo cual se lo conoce como débilmente desacoplado.

Cuando se usa un servicio web basado en la arquitectura SOA los clientes consumen servicios, en vez de ejecutar procedimientos, esto se conoce como servicio orientado a mensajes, por lo cual se mejora notablemente el flujo de la información.

REST: *REpresentational State Transfer* o Transferencia de Estado Representacional. Los Servicios Web basados en REST utiliza las operaciones del protocolo HTTP (GET, POST, PUT, PATCH, DELETE.) para establecer las acciones u operaciones que se ejecutarán sobre los “recursos” que maneja el servicio web.

2.5.2. REpresentational State Transfer (REST)

REST es un término descrito por Roy Fielding en su tesis doctoral “*Architectural Styles and the design of Network-based Software Architectures*”, describe estilos arquitectónicos de sistemas interconectados por red. (Fielding, 2000)

REST es un estilo arquitectónico que especifica cómo los recursos van a ser definidos y direccionados, especifica la importancia del protocolo *cliente-servidor-sin estado*, ya que cada request o petición tiene toda la información necesaria para entenderla.

REST tiene como modelo a la Web, a las características que permitieron que el Internet pueda llegar a tener el tamaño y la capacidad que presenta en nuestros días. En particular, al manejo de direcciones o locación de recursos, los cuales se pueden identificar mediante las

URIs (*Uniform Resource Identifier*). (*Uniform Resource Identifier (URI): Generic Syntax*, 2005)

Los principios de diseño sobre los que se basa REST son: (Marset, 2007)

- Escalabilidad de la interacción con los componentes. La Web maneja una gran cantidad de información, y a pesar de que continuamente crece el número de dispositivos que pueden acceder a la red, no se puede apreciar un decremento de la calidad de comunicación existente.
- Generalidad de interfaces. Gracias al protocolo HTTP, cualquier cliente puede interactuar con cualquier servidor HTTP sin ninguna configuración especial.
- Puesta en funcionamiento independiente. Este hecho se aprecia cuando se observan servidores que al estar en funcionamiento durante bastante tiempo acaban de comunicarse con servidores o dispositivos de última generación y no se puede apreciar que exista un rechazo en la comunicación.
- Compatibilidad con componentes intermedios. Estos componentes pueden ser los proxys que son utilizados para filtrar la información, las caches que se utilizan para mejorar el rendimiento, firewalls para reforzar las políticas de seguridad, etc.

Al implementar una aplicación *RESTful*² significa que los componentes del sistema (por ejemplo los usuarios) son modelados como recursos que pueden ser creados, leídos, actualizados y borrados, usando los “verbos” HTTP POST, GET, PUT y DELETE, estas acciones corresponden a las operaciones CRUD (Create, Read, Update, Delete) de las base de datos relacionales.

- **GET** es la operación HTTP más común, es usado para leer datos en este caso páginas, se puede leer como “get a page”.
- **POST** es la operación usada para crear objetos o recursos, la información va en el cuerpo del request.
- **PUT** se usa para actualizar objetos.
- **DELETE** es usado para borrar objetos.

En el siguiente cuadro 2.1, se muestra como se pueden leer las peticiones al API del servicio que se comunica con los *lugares*, las acciones mostradas son las que se encuentran en un API REST, pero no es necesario declararlas todas para considerar que un API es restful.

²Se denomina RESTful a los sistemas que siguen los principios REST

HTTP request	URI	ACCIÓN	USADO PARA
GET	/lugares	index	devuelve una lista con todos los lugares.
POST	/lugares	create	inserta un nuevo lugar en la base de datos.
GET	/lugares/1	show	muestra el lugar con identificador 1.
PUT	/lugares/1	update	actualiza los datos de un lugar específico.
DELETE	/lugares/1	delete	elimina el lugar con id 1 de la base de datos.

Tabla 2.1: REST URIs para los lugares

Fuente: Elaboración propia

Por ejemplo, si se genera una petición *GET* hacia la dirección */places/1* el servidor interpreta la dirección y gracias al verbo HTTP responde mostrando la información del lugar “1”, en cambio si se genera una petición *PUT* a la misma dirección (*/places/1*), el servidor actualiza los datos del lugar “1” con la información enviada en el cuerpo de la petición HTTP.

2.6. NodeJS

NodeJS apareció en 2009 y está construido sobre el Motor de JavaScript de Google “V8” que fue sacado del browser y aplicado en el servidor.

Para desarrollar en el lado del browser (cliente) el programador sólo tiene disponible JavaScript como lenguaje de desarrollo pero en el lado del servidor existen muchas alternativas (Ruby, C#, Python, Java, etc.), JavaScript no estaba disponible.

NodeJS se beneficia del Motor de JavaScript “V8” ya que este es rápido y tiene integrado un sistema para manejar las instrucciones de forma asincrónica, pero el mayor beneficio y el porqué Node adquirió una gran popularidad es la facilidad de compartir código entre el cliente (browser) y el servidor.

NodeJS provee características pero estas pueden parecer complicadas o que necesitan más instrucciones de las necesarias para llevar a cabo acciones que ya son comunes en la creación de aplicación en lado del servidor, por ejemplo a la hora de crear un servidor web, *NodeJS* se popularizó en gran medida por poder crear servidores web personalizables pero como ya dijimos esto tiene su grado de complejidad, acá es donde entra en acción *ExpressJs*. (Abernethy, 2011)

2.7. ExpressJS

ExpressJS es un framework que está construido sobre la funcionalidad de servidor web de *NodeJS*, *ExpressJS* ayuda a simplificar el API de Node y añadir nuevas características, diseñadas para mejorar y facilitar la organización de una aplicación *Express*. (Hahn, 2014)

El Cliente (navegador web, aplicación móvil, etc) envía una petición web y el servidor web de *NodeJS* maneja los protocolos web, leyéndolos y enviándolos a una aplicación *ExpressJS* que se encarga de añadir características a la petición y espera la respuesta del “Middleware Stack”, la función responde a la llamada y el servidor HTTP de Node envía la respuesta mediante los protocolos web al Cliente.

Para escribir un servidor web con *ExpressJS* no es necesario una gran función para manejar un request, *ExpressJS* contiene utilidades que permite escribir funciones más pequeñas para facilitar el manejo de las peticiones web, haciendo uso de “middleware” y “routing”.

2.7.1. Middleware

NodeJS maneja una función para trabajar con una petición web, en cambio *ExpressJS* maneja la llamada con varias funciones, cada función se encarga de una pequeña parte del trabajo. Estas pequeñas funciones que manejan la petición web se denominan *Middleware functions* o simplemente *Middleware*.

2.7.2. Routing

Muy parecido al *Middleware*, el *Routing* se encarga de partir una petición web monolítica en pequeñas piezas, pero a diferencia del *Middleware*, estos manejadores de peticiones se ejecutan condicionalmente dependiendo del URL y la petición HTTP (GET, POST, DELETE) que el cliente envía.

ExpressJS es bastante extensible y cuenta con gran popularidad en la comunidad de desarrollo, la cual provee herramientas para renderizar dinámicamente HTML o interfaces para comunicarse con Bases de Datos, por ejemplo para manejar la conexión y las llamadas a la base de datos PostgreSQL en el presente proyecto se uso la libreria *knex*.

2.8. Ember JS

Un *framework* o *marco de trabajo* es una abstracción de soluciones a problemas comunes en el desarrollo de software y también provee funcionalidad la cual puede ser modificada por el usuario final, *EmberJS* se define a sí misma como un framework usado para crear aplicaciones web ambiciosas, el cual es eslogan de *EmberJS*, con el que trata de decirnos que usando este framework se podría implementar una aplicaciones web con tanta funcionalidad como si se tratara de una aplicación web nativa.

Para explicar lo que es *EmberJS* hay que mencionar que centró su desarrollo en 3 objetivos (Chan, 2015):

- Enfocarse en aplicaciones web ambiciosas.
- Previsión de Futuros estándares web.
- Estabilidad sin estancamiento.

Ember provee una solución completa a los “problemas” más comunes en el desarrollo de aplicaciones web, pero esto significa mucho “más trabajo” y una curva de aprendizaje más empinada, pero con una consiguiente ayuda para el desarrollador ya que los “problemas” más comunes están resueltos y el desarrollador tiene que enfrentarse a los problemas propios o del modelo de negocio propio de la aplicación a desarrollar.

Ember cuenta con su capa de persistencia o la capa del **Modelo** en el patrón *Modelo-Vista-Controlador*, *Ember-Data*, el cual maneja los datos mientras están en memoria y se asegura de sincronizar con el servidor cuando se requiere y modifica la base de datos. El formato por defecto para manejar la información es *JSON* o *JavaScript Object Notation*, el cual es un formato de texto ligero para el intercambio de datos.

Para facilitar el trabajo de desarrollo en la capa de la **Vista**, Ember implementa *HandleBars* el cual es un motor de plantillas HTML, es decir que realmente no se escribe una página HTML, en cambio se crea un template que con el cual *HandleBars*³ crea las páginas HTML que los navegadores entienden, esto permite escribir código mucho más limpio y también permite sincronizar variables con el Controlador, esto significa que si actualizamos código en la Vista, este es actualizado en el Controlador y viceversa.

En *EmberJS* La capa del **Controlador** es la encargada de recibir los datos de la Vista y de acuerdo a la interacción del usuario con la aplicación, dispara o activa diferentes acciones que en general modifican los datos ingresados y ya sea para mostrar en UI o guardarlos en la base de datos.

Ember provee de una herramienta de línea de comandos, *Ember-CLI* o *Ember Command Line Interface*, el cual ofrece para agilizar el desarrollo, usado para automatizar procesos repetitivos, por ejemplo, estableciendo la estructura de directorios del proyecto esto basado en la experiencia de numerosos proyectos, realiza la concatenación, compilación, compresión, y demás manejos de archivos. Como también provee un ecosistema de addons o complementos, que añaden características nuevas al ecosistema que ofrece *EmberJS*, hay que notar que al ser un proyecto de Software Libre existe una gran cantidad de addons disponibles y cada día aparecen mas. Para el desarrollo de este proyecto se hará uso de distintos addons, los cuales se listan a continuación:

- **ember-paper:** Este *addon* es el encargado de adaptar la Vista de la aplicación web en la pantalla de un smartphone, necesario ya que por ejemplo el smartphone no tiene

³<http://handlebarsjs.com/>

un mouse para hacer click, por el contrario es necesario hacer “tap” con un dedo para ejecutar la misma acción que el mouse, también está el hecho que el tamaño de la pantalla del smartphone es muy inferior a la de un monitor estándar pero la experiencia del usuario tiene que estar diseñada para interactuar con las características que ofrece un smartphone.

- **ember-leaflet:** Este *addon* está diseñado para ayudar a desplegar un mapa, en este caso de estudio se está usando los mapas de *OpenStreetMaps™*, y optimizado para no usar demasiados recursos, ya que muchas veces los smartphones aún teniendo buenas características no se comparan a una computadora de escritorio.
- **cloudinaryJS:** *Addon* diseñado para poder manejar las imágenes en la nube, provee varias características como adaptación del tamaño de la imagen de forma automática de acuerdo al tamaño del navegador, es de uso libre pero con limitaciones uso en cuanto a las transacciones que se pueden realizar o la cantidad de imágenes que se pueden almacenar.

2.9. Base de Datos

En una aplicación web es necesario alguna forma de persistencia de datos, en especial si se están usando datos complejos como la información geoespacial, para realizar esta tarea, la base de datos es un factor primordial. Para este proyecto de grado se hará uso de *PostgreSQL* como base de datos relacional y su extensión *Postgis* para manejar los datos geoespaciales.

2.9.1. PostgreSQL

PostgreSQL es un sistema de gestión de bases de datos objeto-relacional, Open Source y distribuido bajo licencia BSD. PostgreSQL utiliza un modelo cliente/servidor y usa multiprocesos en vez de multihilos para garantizar la estabilidad del sistema. Un fallo en uno de los procesos no afectará el resto y el sistema continuará funcionando.

La última versión estable de PostgreSQL es la 9.5, su desarrollo comenzó hace más de 16 años, y cuenta con una gran comunidad que aporta con el desarrollo y el testeo de nuevas versiones. PostgreSQL está considerada como uno de los mejores *Sistemas de gestión de bases de datos*, es muy completo y está muy bien documentado⁴. Entre sus características se pueden nombrar las siguientes.

- Es una base de datos 100 % *ACID* (Atomicity, Consistency, Isolation and Durability).
- Integridad referencial.
- Replicación asincrónica/sincrónica.
- Múltiples métodos de autenticación.
- Disponible para Linux y UNIX en todas sus variantes.

⁴<https://www.postgresql.org/about/>

- Funciones/procedimientos almacenados.
- Soporte a la especificacion SQL.

Se escogió trabajar con PostgreSQL como DBMS porque cuenta con una extensa documentación, a su caracter “Open Source” y su gran flexibilidad en poder definir nuevos tipos de datos, esto se hace posible que empresas como *Refractions Research* puedan crear recursos como *PostGIS*, necesario para trabajar con datos geográficos o espaciales.

2.9.2. PostGIS

PostGIS es un módulo que anade soporte de objetos geográficos al DBMS PostgreSQL, convirtiéndola en una base de datos espacial para su utilización en un Sistema de Informacion Geografica o *SIG*, es bastante común utilizar el acrónimo en Inglés, *Geographic Information System* o *GIS* y de hay viene el término de PostGIS, que combina Postgres y GIS.

El desarrollo de PostGIS está a cargo de Refractions Research, está liberada con la *Licencia pública general de GNU*, declarandola como software libre que lo protege de cualquier intento de apropiacion.

PostGIS implementa la especificación “SFSQL” (Simple Features for SQL, define los tipos y funciones que necesita implementar cualquier base de datos espacial) de la *OGC* (Open Geospatial Consortium, es un consorcio internacional, formado por un conjunto de empresas, agencias gubernamentales y universidades, dedicado a desarrollar especificaciones de interfaces para promover y facilitar el uso global de la información espacial).

PostGIS al igual que *PostgreSQL* cuenta con una documentacion bastante extensa y equipo de desarrollo que continuamente va sacando nuevas versiones, actualmente se encuentra la version 2.2.2, pero para el desarrollo de la aplicacion se hizo uso de la version 2.1.0.

PostGIS es gratis, pero no por ello es una herramienta de baja calidad, al contrario se la considera una herramienta de nivel empresarial, y muchas instituciones la estan usando de manera exitosa, aparte de numerosas aplicaciones.

Manejar los datos geográficos con PostGIS es sencillo y eficiente, por esta razón se utilizó esta herramienta, pero para conseguir la ruta óptima entre 2 puntos se necesitaba el uso del algoritmo de Dijkstra y para PostGIS existe el módulo *PgRouting*, que tiene implementado este algoritmo.

2.9.2.1. pgRouting

pgRouting es una extension de PostGIS para proveer funcionalidades de ruteo espacial. *pgRouting* es un desarrollo posterior de *pgDijkstra* y actualmente está siendo mantenido por

Georepublic, la última versión estable es la 2.1, y es la que fue usada para desarrollar el sistema⁵.

Las ventajas del ruteo en la base de datos son:

- Los datos y atributos pueden ser modificados desde varios clientes, como *Quantum GIS* y *uDig* a través de *JDBC*, *ODBC*, o directamente usando *Pl/pgSQL*. Los clientes pueden ser PCs o dispositivos móviles.
- Los cambios pueden ser reflejados instantáneamente a través del motor de ruteo. No hay necesidad de hacer cálculos previos.
- El parámetro de “costo” puede ser calculado dinámicamente a través de SQL y su valor puede provenir de múltiples campos y tablas.

pgRouting provee funciones para:

- Camino mínimo (Dijkstra): algoritmo de ruteo sin heurística
- Camino mínimo (A-Star): ruteo para conjunto de datos grandes (con heurística)
- Camino mínimo (Shooting-Star): ruteo con restricciones de giro (con heurística)
- El problema del viajante (TSP: Traveling Salesperson Problem)
- Cálculo de ruta (Isolíneas)

⁵<http://pgrouting.org/>

Capítulo 3

Metodología de Desarrollo

3.1. Introducción

La metodología para el desarrollo de software permite gestionar y administrar un proyecto de desarrollo de software para llevarlo a término de una forma más eficiente y con altas probabilidades de éxito, seguir una metodología es importante ya que ayudará a organizar y a seguir un ritmo de trabajo.

3.2. Metodologías Ágiles

Este término nace en una reunión celebrada en febrero de 2001 en Utah - USA por expertos en la industria del software ya que pretendían encontrar una forma alternativa de desarrollo de software a las que estaban vigentes hasta esa fecha por ejemplo la metodología en cascada que es rígido y obliga una planeación extensiva antes siquiera de tocar una línea de código, está demostrado que este tipo de metodologías son muy rígidas y les falta flexibilidad a la hora de hacer frente a los cambios que invariablemente sufre un proyecto de desarrollo de software.

Para contravenir estas dificultades es que se definieron los principios del manifiesto ágil¹:

Individuos e interacciones sobre procesos y herramientas.

Software funcionando sobre documentación extensiva.

Colaboración con el cliente sobre negociación contractual.

Respuesta ante el cambio sobre seguir un plan.

El principal objetivo de las metodologías ágiles es la habilidad de soportar los cambios, los cuales generalmente por no decir casi siempre aparecen en un ambiente que sufre muchos cambios y rápidamente, los cuales son difíciles de predecir.(Hoffer, George, y Valacich, 2005)

Para alcanzar este objetivo es que las metodologías ágiles se basan en tres principios: (Hutagalung y Sauter, 2006)

¹<http://agilemanifesto.org/iso/es/manifesto.html>

- Enfoque en metodologías que se adapten al cambio
- Enfocarse en las personas
- Enfocarse en procesos que se auto-adapten al cambio

Las metodologías ágiles no se refieren a un único y específico método o técnica de desarrollo, en cambio son un grupo de metodologías que implementan los principios ágiles. Entre los cuales se pueden apreciar las siguientes metodologías:

- Scrum
- Dynamic Systems Development Method (DSDM)
- Crystal Methods
- Feature Driven Development
- Lean Development
- Extreme Programming (XP)
- Adaptive Software Development

3.3. Programación Extrema

La *programación extrema* o *XP* es una metodología de trabajo creada a mediados de 1990 por Kent Beck cuando estaba trabajando en un proyecto de desarrollo de software en *Chrysler Comprehensive Compensation* (C3) en un intento de mejorar el proceso de desarrollo de software y posteriormente con una segunda implementación de un proyecto usando la metodología XP en *Vehicle Cost and Profitability System (VCAPS)* propiedad de *Ford Motor Co.*, se demostró que esta metodología de desarrollo es un método apropiado para llevar a buen término el proyecto de desarrollo de software. (Wells, 2013)

XP se enfoca en la adaptabilidad ya que el desarrollo de software debería ser un proceso fluido donde los requerimientos no pueden ser totalmente predichos desde el principio del desarrollo ya que estos siempre o casi siempre tienden a cambiar a medida que el software se va desarrollando ya sea por cambios en el mercado o a medida que el cliente va aprendiendo y modificando sus requerimientos en el transcurso del ciclo de desarrollo del producto.

Kent Beck declaró los siguientes cuatro enunciados que son la base de la filosofía de XP:

- Es necesario mejorar la comunicación
- Es necesario encontrar simplicidad
- Es necesario obtener feedback o retroalimentación de parte del cliente
- Es necesario proceder con coraje.

Combinando estos principios, la programación extrema se trata acerca de mejorar el trabajo en equipo cohesionado y con la ayuda de la retroalimentación propia del equipo se puede apreciar donde se encuentra y mejorarlo, siempre tomando en cuenta que cada equipo es único, ya sea por el tipo de software que se está desarrollando y por las personas que conforman el equipo.

Las prácticas usadas en XP son de hecho prácticas comúnmente usadas en las metodologías ágiles pero en XP estas prácticas son llevadas al extremo de ahí el nombre de programacion extrema.

La programación extrema se caracteriza por las siguiente practicas: (Ponce, 2004)

- **Code reviews:** O revisión de código, en programación extrema esto se llama programación en pareja (pair programing), esto significa que dos programadores escriben código usando o compartiendo una máquina, esto se traduce en que el código es constantemente revisado y por lo tanto es menos proclive de producir errores.
- **Testeo:** En XP significa hacer unit testing o pruebas unitarias durante todo el proceso de desarrollo de software, una vez el producto es entregado al cliente este se encarga de probar la funcionalidad del sistema.
- **Diseño:** En XP se necesita que todos los involucrados en el proyecto estén siempre y constantemente refactorizando y mejorando el producto. Simplicidad: Siempre dejar el sistema con el diseño más simple posible para que soporte la funcionalidad deseada o lo más simple que funciona. Se basa en la filosofía de que el mayor valor de negocio es entregado por el programa más sencillo que cumpla los requerimientos.
- **Arquitectura:** Todos trabajando definiendo y redefiniendo constantemente la arquitectura del sistema. Testeo de integración: Unir o integrar y probar las diferentes características del software que se están trabajando, constantemente o por lo menos una vez al dia.
- **Iteraciones cortas:** Trabajar en ciclos realmente cortos, puede ser de horas o días pero no semanas o meses, permitiendo que el programa, el verdadero valor del negocio, pueda ser evaluado.
- **Propiedad colectiva del código:** un código con propiedad compartida. Nadie es el propietario de nada, todos son el propietario de todo. Este método difiere en mucho a los métodos tradicionales en los que un simple programador posee un conjunto de código.
- **Estándar de codificación:** define la propiedad del código compartido así como las reglas para escribir y documentar el código y la comunicación entre diferentes piezas de código desarrolladas por diferentes equipos
- **Bienestar del programador:** La semana de 40 horas, la programación extrema sostiene que los programadores cansados escriben código de menor calidad. Minimizar las horas extras y mantener los programadores frescos, generará código de mayor calidad.

3.3.1. Las historias de usuario

Es la técnica que utiliza XP para especificar los requisitos del software. Se trata de tarjetas en las cuales el cliente escribe las características que el sistema debe poseer, sean requisitos

funcionales o no funcionales. El proceso de manejar las historias de usuario es muy dinámico ya que se pueden añadir, eliminar o modificarse de acuerdo a la exigencia que puede aparecer a cualquier momento, las historias deben ser lo bastante simples como para que los programadores las implementen en unas semanas. (Ponce, 2004)

3.3.2. Proceso de desarrollo

La programación extrema identifica las siguientes fases en el proceso de desarrollo de software. (Ponce, 2004)

- **Interacción con el cliente:** El cliente es una parte importante en el equipo de desarrollo, tiene gran importancia en el equipo ya que expresa su opinión sobre el producto después de cada cambio o iteración, mostrando las prioridades y expresando su opinión sobre los problemas que se podrían identificar.
- **Planificación del proyecto:** En este punto se elabora la planificación por etapas o iteraciones. Para hacerlo será necesaria la existencia de reglas que han de seguir las partes implicadas en el proyecto.
- **Diseño, desarrollo y pruebas:** El desarrollo es la parte más importante en el proceso de la programación extrema. Todos los trabajos tienen como objetivo que se programen lo más rápidamente posible, sin interrupciones y en la dirección correcta.

3.3.3. Roles de la programación extrema

- **Programador:** Escribe las pruebas unitarias y produce el código del sistema.
- **Cliente:** Escribe las historias de usuario y las pruebas funcionales para validar su implementación. Asigna la prioridad a las historias de usuario y decide cuáles se implementan en cada iteración centrándose en aportar el mayor valor de negocio.
- **Tester:** Ayuda al cliente a escribir las pruebas funcionales. Ejecuta pruebas regularmente, difunde los resultados en el equipo y es responsable de las herramientas de soporte para pruebas.
- **Tracker:** Es el encargado de seguimiento. Proporciona realimentación al equipo. Debe verificar el grado de acierto entre las estimaciones realizadas y el tiempo real dedicado, comunicando los resultados para mejorar futuras estimaciones.
- **Entrenador (coach):** Responsable del proceso global. Guía a los miembros del equipo para seguir el proceso correctamente.
- **Consultor:** Es un miembro externo del equipo con un conocimiento específico en algún tema necesario para el proyecto. Ayuda al equipo a resolver un problema específico.
- **Gestor (Big boss):** Es el dueño de la tienda y el vínculo entre clientes y programadores. Su labor esencial es la coordinación.

3.4. El Proceso XP

XP propone un proceso iterativo e incremental. El proyecto es dividido en pequeños “mini-proyectos”, los cuales terminan con un *release* que es una versión del producto que se libera al final de un ciclo de desarrollo de software, un *release* contiene requerimientos implementados, tal vez no acabados en un 100 % pero funcional de tal forma que el cliente es capaz de ofrecer *feedback* del producto. (Dudziak, 2000)

Estos “mini-proyectos” o iteraciones, son negociados entre los clientes y los desarrolladores, donde el cliente determina el valor de negocio de las características que se quiere desarrollar y los desarrolladores especifican el tiempo y esfuerzo que se necesita para desarrollar estas características, el cliente sopesa el valor, el esfuerzo y el tiempo de la iteración y decide qué características se desarrollaran, como último paso en la iteración los desarrolladores implementan las características seleccionadas.

En la figura 3.1 se puede apreciar el diagrama del proceso de desarrollo de la metodología XP.

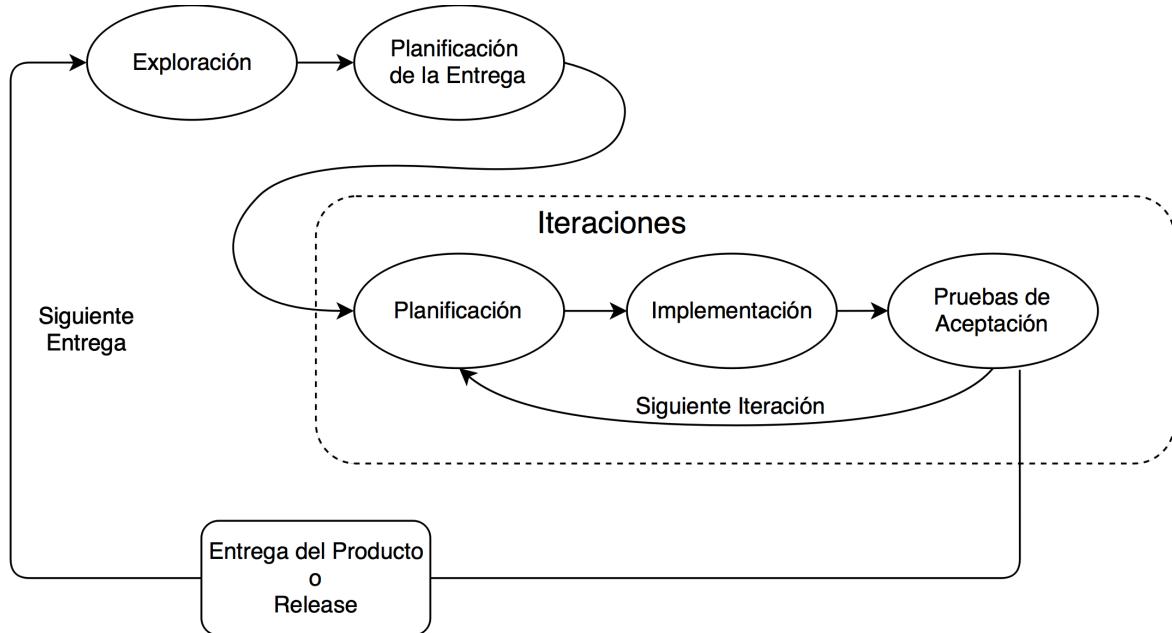


Figura 3.1: Diagrama del proceso XP

Fuente: Implementación propia

3.4.1. Exploración

Es la primera fase, durante la cual los clientes definen lo que desean que tenga el sistema y los desarrolladores estiman el tiempo necesario que necesitan para realizar esas tareas.

Durante esta etapa el cliente escribe las tarjetas de *historias de usuario*, estas historias definen lo que el cliente quiere que el sistema haga, en otras palabras representan las características que el sistema debería tener implementado.

Es gracias a las *historias de usuario*, que el equipo de desarrollo comienza a aprender de las herramientas y las tecnologías que va a utilizar, para posteriormente asignar una estimación de tiempo y esfuerzo que necesitará el desarrollo de las *historias del usuario*.

3.4.2. Planificación de la Entrega

Cuando se tienen las historias de usuario, el cliente determina la prioridad y los desarrolladores estiman el esfuerzo necesario para cada una, se procede a una negociación donde se aceptan qué historias se van a desarrollar en primer lugar y en cuanto tiempo se entregaran resultados, en esta etapa se define el “plan de entregas”.

Las historias de usuario se las crea para propósitos de planeación y estimación de tiempo y esfuerzo que cada característica va a necesitar, los detalles se crean y dividen posteriormente en las *tareas de ingeniería* cuando las historias están por ser implementadas.

3.4.3. Iteraciones

Esta fase es un conjunto de iteraciones de entre 2 y 3 semanas, cada iteración consta de las siguientes etapas: Planificación, Implementación y Pruebas de Aceptación.

3.4.3.1. Planificación

Cada iteración tiene su propia etapa de *planificación* donde se tienen en cuenta las historias de usuario seleccionadas, las faltantes y las tareas que se realizarán.

Los desarrolladores dividen las historias de usuario en *Tareas de Ingeniería*, las cuales son más pequeñas que las historias, si se encuentra que una *Tarea de Ingeniería* es casi tan grande como una historia de usuario es porque es una historia de usuario y debería ser tratada como tal.

Una tarea puede estar relacionada a 2 o mas historias o no estar relacionada con ninguna. Dentro de la metodología XP es una buena práctica el escribir las tareas en tarjetas, similares a las historias de usuario, esto debido a que las tarjetas son más fáciles de manipular.

Durante esta fase un desarrollador acepta la responsabilidad de implementar una *Tarea de Ingeniería* de acuerdo de su experiencia personal en el área y tecnologías que se usarán en el desarrollo. El desarrollador debe estimar el tiempo necesario para completar la tarea en un Tiempo de Ingeniería Ideal, una regla de XP consiste en que no se debe realizar trabajo el cual no se va necesitar en ese momento, **YAGNI**².

3.4.3.2. Implementación

Durante esta etapa el equipo de desarrollo empieza a escribir código que satisfaga los criterios de aceptación descritos en la *Historia de Usuario* seleccionada para ser trabajada en la Iteración, *XP* propone los siguientes procedimientos:

²You aren't gonna need it. En español, Tu no lo vas a necesitar o No vas a necesitarlo.

- Analizar lo que hay que hacer, esto envuelve lo que es analizar las *Tarjetas de Ingeniería* y/o las *Historias de Usuario*.
- Escribir Pruebas Unitarias, son bastante útiles para determinar cuando la tarea está completada.
- Implementar el código suficiente para lograr que las pruebas unitarias pasen exitosamente.
- Simplificar el código si es necesario (*Refactor Mercilessly*).
- Integrar los cambios continuamente (*Continuous Integration*).

3.4.3.3. Pruebas de Aceptación

Cada *Historia de Usuario* lleva asociado pruebas, estas son diseñadas para verificar que los criterios de aceptación de cada *Historia de Usuario* fueron implementadas correctamente. Si durante esta fase las pruebas fallan, la historia de usuario relacionada se marca para volver a trabajarla en la siguiente iteración.

Las pruebas de aceptación son escritas y ejecutadas para validar la calidad del producto que se está entregando, hay que subrayar que ningún producto o aplicación desarrollado está exento de fallas, porque los que escriben los programas son personas y las personas siempre tienden a cometer errores.

Es por esta razón que las pruebas de aceptación son de gran importancia porque ayudan a detectar las fallas o posibles fallas que el sistema puede tener. La detección temprana de errores es de gran importancia en el desarrollo de software ya que los errores encontrados en etapas de implementación son más baratos que los errores encontrados por el cliente, estos errores cuestan más porque socavan la confianza del cliente o usuario hacia el producto desarrollado.

Tomando en cuenta los alcances demarcados para el presente proyecto se hará énfasis en los siguiente criterios de calidad durante la fase de Iteración.

- **Funcionalidad:** Las *pruebas funcionales*, son diseñadas tomando como referencia las especificaciones funcionales de un componente o sistema, sirven para comprobar si el software cumple las funciones esperadas. Estas pruebas están comprendidas en lo que se denominan *pruebas de caja negra*. (Escudero y Sosa, 2017)

Dentro de las pruebas funcionales se pueden encontrar 2 tipos de pruebas, las pruebas positivas y las pruebas negativas. Las *pruebas positivas* se caracterizan por usar datos de entrada válidas y así comprobar que el sistema funciona como espera el usuario. Las *pruebas negativas* usan datos de entrada no-válidas o negativas esperando un error como resultado, el cual sea el esperado en relación a la funcionalidad del sistema. (Bascopé, 2017, p. 24)

- **Usabilidad:** El objetivo del atributo de usabilidad es lograr que al usuario le sea cómoda y fácil la operación como también el manejo de la aplicación, las aplicaciones que no

son usables son fácilmente cambiadas por aplicaciones que cumplen esta característica. (Offutt, 2002, p. 27)

Es buena práctica ejecutar todas las pruebas escritas al final de cada iteración, *pruebas de regresión*, de esta forma se puede comprobar que durante la implementación de código durante la presente iteración no se haya roto la funcionalidad de las características ya implementadas en anteriores iteraciones. (Priolo, 2009, p. 25)

3.4.4. Producción

En la fase de producción se realizarán pruebas para validar el correcto funcionamiento del sistema en general, *pruebas de rendimiento*, como resultado de estas pruebas se evalúa que el software desarrollado cumple con las características mínimas acordadas para su entrega al cliente. (Bascopé, 2017, p. 139)

Tomando en cuenta que se desarrollara una aplicación web, se utilizarán *pruebas de carga* para validar el rendimiento de la aplicación.

- **Pruebas de Carga:** Este tipo de prueba es realizado para evaluar el comportamiento de los componentes del sistema con un incremento de carga constante, por ejemplo una cantidad determinada de usuarios o transacciones en paralelo que interactúan con el sistema. Las pruebas de carga sirven para determinar la cantidad de carga que el sistema puede soportar antes fallar. (Escudero y Sosa, 2017)

Para realizar esta prueba se utilizó *Apache JMeterTM*, el cual es una herramienta diseñada para ejecutar pruebas de carga, comprobar el comportamiento funcional de las pruebas y medir el rendimiento, originalmente diseñada para probar aplicaciones web³.

³<http://jmeter.apache.org/>

Capítulo 4

Campus Universitario

4.1. Introducción

La Universidad Mayor de San Simón fue fundada mediante ley de 5 de noviembre de 1832 por el Mariscal Andrés de Santa Cruz. La misma ley dispuso la creación y funcionamiento de una Academia de Practicantes Juristas, con la que, en realidad, se inicia la Facultad de Derecho, y hasta la fecha la Universidad cuenta con 12 Facultades de las cuales 7 se encuentran dentro del campus. (*Historia de la Universidad Mayor de San Simón*, 2005)

Actualmente dentro la página de la Universidad se puede encontrar un “Mapa Universitario”, el cual se puede apreciar en la figura 4.1, este mapa consiste en una vista del campus universitario como un todo dentro del plano de la ciudad de Cochabamba, la misma página cuenta con la sección “Paseo Virtual” la cual lamentablemente no muestra nada.



Figura 4.1: Mapa universitario

Fuente: (*Mapa Universitario*, 2005)

Los estudiantes, docentes y visitantes que usan y circulan dentro de los predios del campus, necesitan contar con un mapa más actualizado

Actualmente dentro del campus universitario “Las Cuadras” se puede encontrar las siguientes sectores:

1. FACH - Facultad de Arquitectura y Ciencias del Hábitat
 2. FCJyP - Facultad de Ciencias Jurídicas y Políticas
 3. FACES - Facultad de Ciencias Económicas
 4. FHCE - Facultad de Humanidades y Ciencias de la Educación
 5. FCyT - Facultad de Ciencias y Tecnología
 6. Multiacademico
 7. Complejo Deportivo

La Universidad cuenta con Aulas numeradas hasta el 758, repartidas entre todas las facultades y hasta para un estudiante que pasa gran parte de su tiempo dentro del campus es difícil encontrar un lugar cuando no se sabe su ubicación exacta, entonces para un visitante ocasional esto puede llegar a ser un gran problema.

Pero aparte de Aulas también se encuentra del campus, oficinas administrativas, centros de estudiantes, snacks, etc. por lo que conocer con exactitud la locación de un lugar es de gran importancia para no extraviarse dentro del campus universitario.

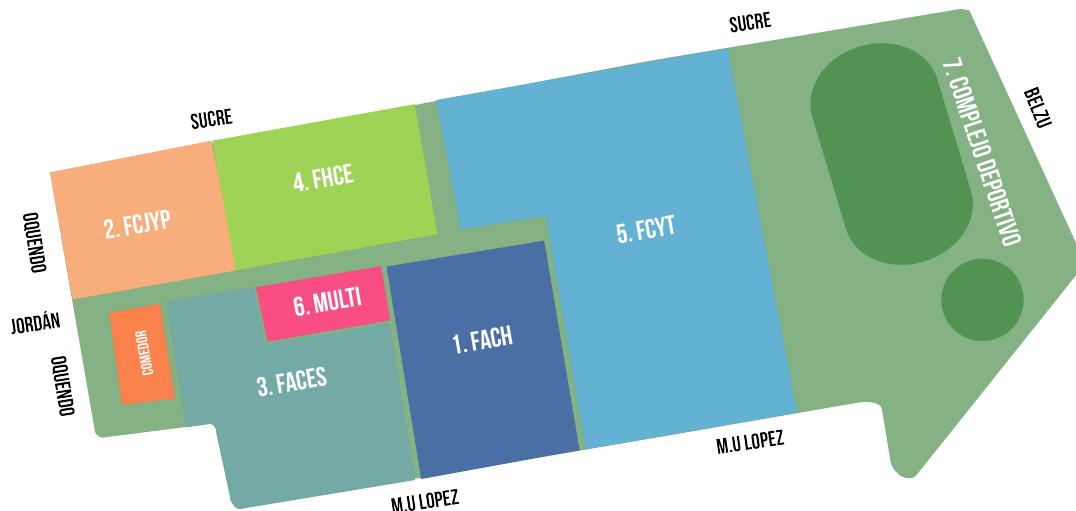


Figura 4.2: Facultades dentro del Campus

Fuente: Elaboración propia

En la figura 4.2 se puede apreciar a grandes rasgos la distribución de las facultades dentro del campus, el mapa sirve para dar una idea de la ubicación de las facultades y edificios administrativos, los cuales son detallados a continuación.

4.2. Facultad de Arquitectura y Ciencias del Hábitat

La Facultad de Arquitectura y Ciencias del Hábitat colinda con la calle M. U. Lopez, dentro de los predios del campus Universitario se halla entre las facultades de Economía hacia el Sur-Este y con la facultad de Tecnología hacia el Nor-Oeste, en la figura 4.3 se puede apreciar la facultad con las aulas y oficinas que cuenta en su interior.

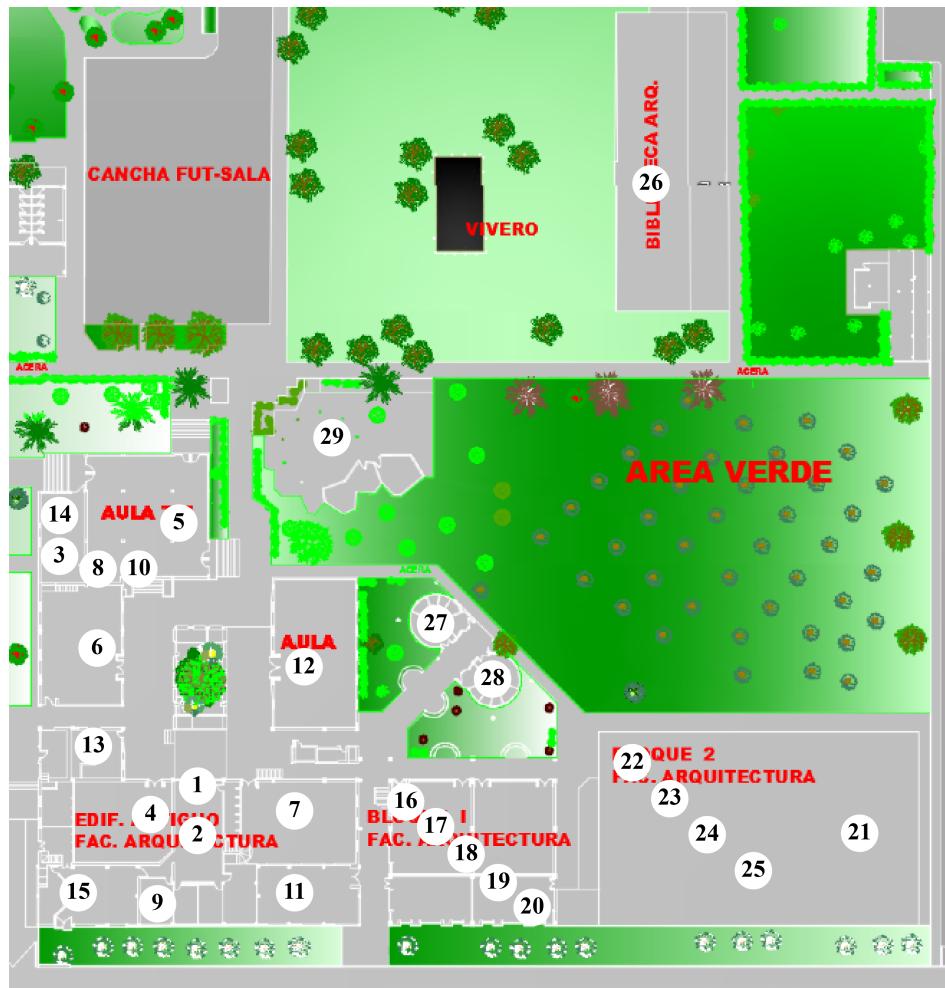


Figura 4.3: Facultad de Arquitectura - UMSS

Fuente: Departamento Infraestructura - UMSS

Punto	Detalle
1	Decanato FACH
2	Dirección Carrera Arquitectura
3	Dirección Carrera Turismo
4	Asociación Docente de Arquitectura
5	Aula 701
6	Aula 702
7	Aula 703
8	Aula 704
9	Aula 705
10	Aula 706
11	Aula 707
12	Aula 709
13	Centro de Estudiantes de Arquitectura
14	Centro de Estudiantes de Turismo
15	Laboratorio de Arquitectura
16	Planta Baja - Aulas: 720, 721, 722, 723
17	1er Piso - Aulas: 724, 725
18	2do Piso - Aulas: 726, 727
19	3er Piso - Aulas: 728, 729
20	4to Piso - Dirección de Formación Continua Grado y Postgrado FACH
21	Planta Baja - Auditorio "M.Sc. Arq. Brownie Mostajo Medinaceli"
22	1er Piso - Aulas: 740, 741, 742, 743
23	2do Piso - Aulas: 745, 746, 747, 748
24	3er Piso - Aulas: 750, 751, 752, 753
25	4to Piso - Aulas: 755, 756, 757, 758
26	Biblioteca FACH
27	Aula 729
28	EDAV - Estudio de Artes Visuales
29	Snack Arquitectura
A30	2do Piso - Aulas: 760, 761, 762, 763 - Ver en el mapa del Multiacadémico, figura 4.8
A31	Aulas: TODO* - Ver en el mapa de Tecnología, figura 4.7

Tabla 4.1: Locaciones de la Fac. Arquitectura

4.3. Facultad de Ciencias Jurídicas y Políticas

Generalmente conocida como "Facultad de Derecho" se encuentra al nor-oeste del Campus Universitario, en la Av. Oquendo esquina Sucre. En la figura 4.4 se puede ver las locación de los lugares que se puede encontrar dentro de esta facultad.

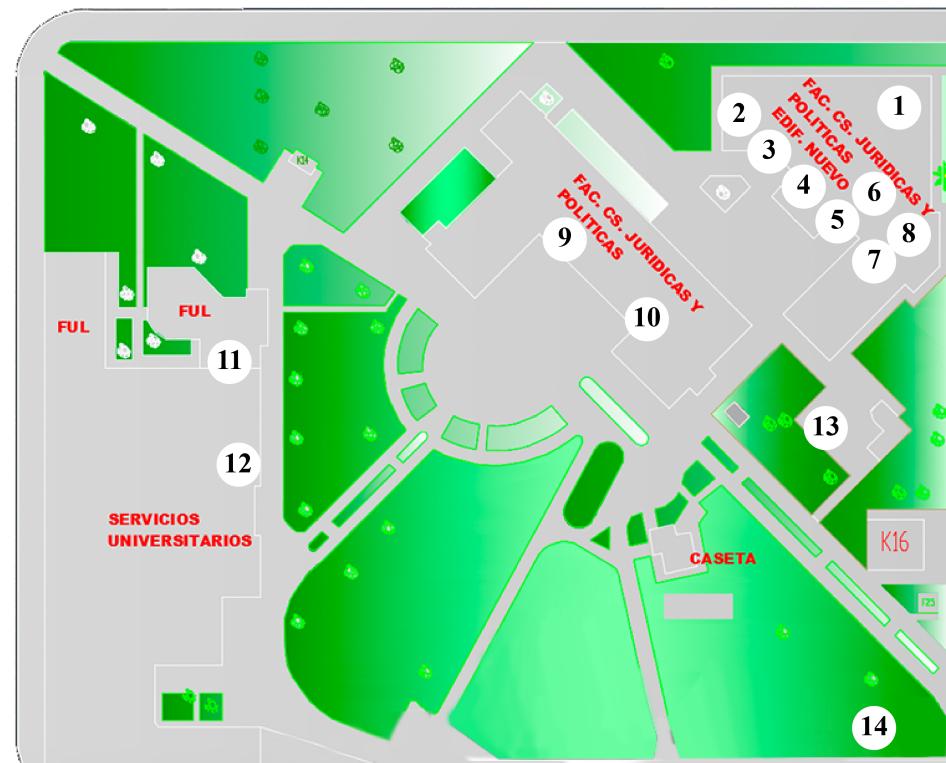


Figura 4.4: Facultad de Derecho - UMSS

Fuente: Departamento Infraestructura - UMSS

Punto	Detalle
1	Centro de Estudiantes de Derecho
2	1er Piso - Aula PP1, Aula PP2, Aula PP3, Aula PP4, Aula PP5, Aula PP6
3	2do Piso - Aula SP1, Aula SP2, Aula SP3, Aula SP4, Aula SP5
4	3er Piso - Carrera de Ciencia Política
5	3er Piso - Aula TP1, Aula TP2, Aula TP3, Aula TP4, Aula TP5
6	3er Piso - Salón Auditorio "Lic. Orlando Mercado Camacho"
7	4to Piso - Aula CP1, Aula CP2, Aula CP3, Aula CP4, Aula CP5
8	5to Piso - Aula QP1, Aula QP2, Aula QP3, Aula QP4, Aula QP5, Aula QP6
9	Aula BA1
10	Oficina Educativa Virtual Facultativa
11	FUL
12	SITUMSS - Sindicato Trabajadores UMSS
13	Snack Derecho
14	Torre Fotos UMSS

Tabla 4.2: Locaciones de la Fac. Derecho

4.4. Facultad de Ciencias Económicas

La Facultad de Ciencias Económicas o comúnmente conocida como “Facultad de Economía” colinda con las calles Oquendo y M. U. López, dentro del campus al Nor-Este se encuentra la facultad de Arquitectura y al Nor-Oeste la facultad de Derecho, tal como se puede apreciar en la figura 4.5.

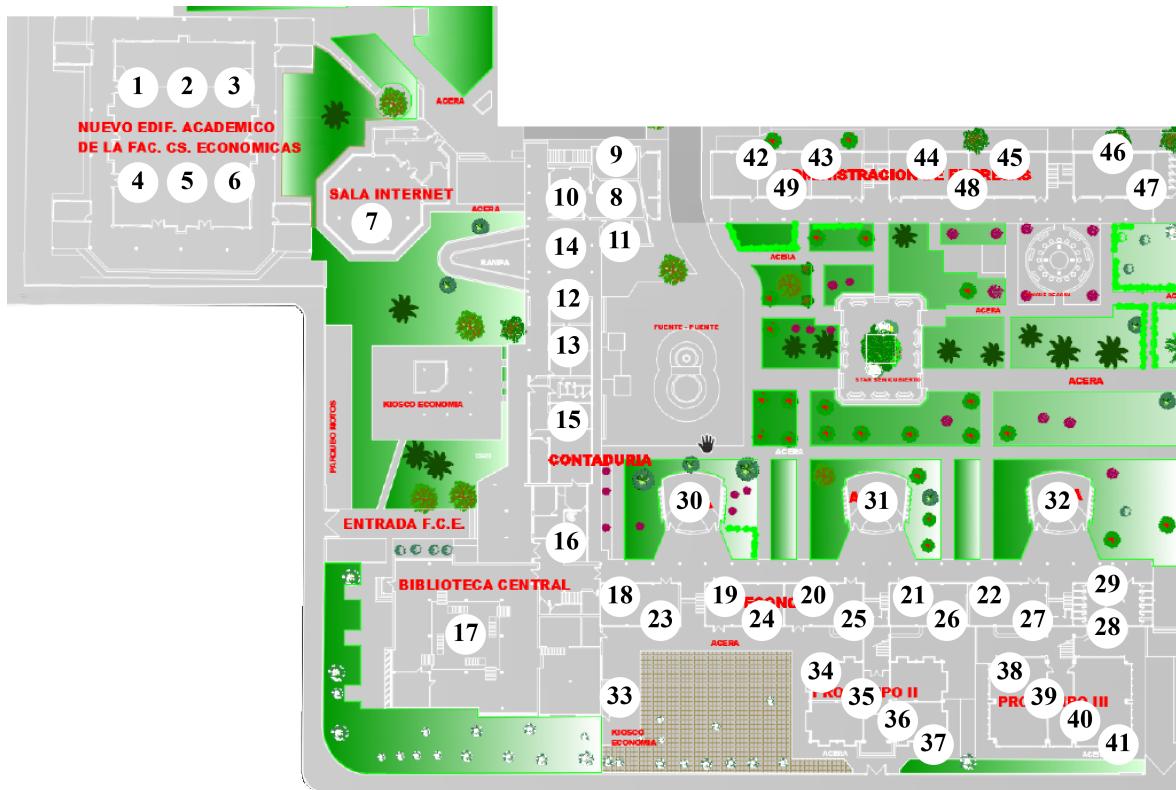


Figura 4.5: Facultad de Economía - UMSS

Fuente: Departamento Infraestructura - UMSS

Punto	Detalle
1	Direccion de Carrera de Ing. Comercial, Aulas: PB-001, PB-002, PB-003, PB-004
2	1er Piso - Aulas: 101, 102, 103, 104, 105, 106, 107, 108, 109, 110
3	2do Piso - Aulas: 201, 202, 203, 204, 205, 206, 207, 208, 209, 210
4	3er Piso - Aulas: 301, 302, 303, 304, 305, 306, 307, 308, 309, 310
5	4to Piso - Auditorio 1 y 2 FCE, Aulas: 401, 402, 403, 404, 405, 406
6	5to Piso - Salon Auditorio “Lic. Ramiro Perez”
7	Unidad Tecnologias de Informacion, Centro de Computo FCE
8	Direccion de Carrera Administracion de Empresas
9	1er Piso - Centro de Estudiantes Ing. Comercial
10	Centro de Estudiantes de TODO

Punto	Detalle
11	Centro de Estudiantes de TODO
12	Centro de Estudiantes de TODO
13	Direccion de Carrera de Economia
14	1er Piso - Aula Magna “Amauta”
15	Asociacion Universitaria Docentes de Economia
16	1er Piso - Aulas: 501, 502, 503, 504
17	Biblioteca FCE
18	Centro de Estudiantes de Ing. Financiera
19	Aula 514
20	Aula 513
21	Aula 512
22	Aula 511
23	1er Piso - Auditorio Carrera de Economia, Biblioteca - Sala Estudio CCP
24	1er Piso - Aula 506
25	1er Piso - Aula 507
26	1er Piso - Aula 508
27	1er Piso - Aula 509
28	1er Piso - Aula 510
29	1er Piso - Caja Facultativa FCE
30	Aula 516
31	Aula 517
32	Aula 518
33	Snack Economia - Simona's
34	Edificio Prototipo I - Aula PB 534
35	1er Piso - Direccion de Carrera CCP, Aulas: 536, 537
36	2do Piso - CEPLAG
37	3er Piso - Decanato CPE, Direccion Academica
38	Edificio Prototipo II - Aulas: PB 541, 542
39	1er Piso - Aulas: 544, 543
40	2do Piso - Aulas: 545, 546
41	3er Piso - Aulas: 547, 548, 549, 550
42	Aula 520
43	Aula 521
44	Aula 522
45	Aula 523
46	Aula 524
47	1er Piso - Taller Economia
48	1er Piso - Aulas: 525, 526, 527
49	1er Piso - Aulas: 528, 529, 530

Tabla 4.3: Locaciones de la Fac. de Economia

4.5. Facultad de Humanidades y Ciencias de la Educación

La entrada a la facultad de Humanidades se encuentra sobre la calle Sucre en frente de la *Plaza Sucre* acera Sud, en la figura 4.6 se puede apreciar la *FHCE* dentro del campus Universitario y en la tabla 4.4 se puede ver el detalle de los lugares que se pueden ubicar dentro de la facultad de Humanidades y Ciencias de la Educación.

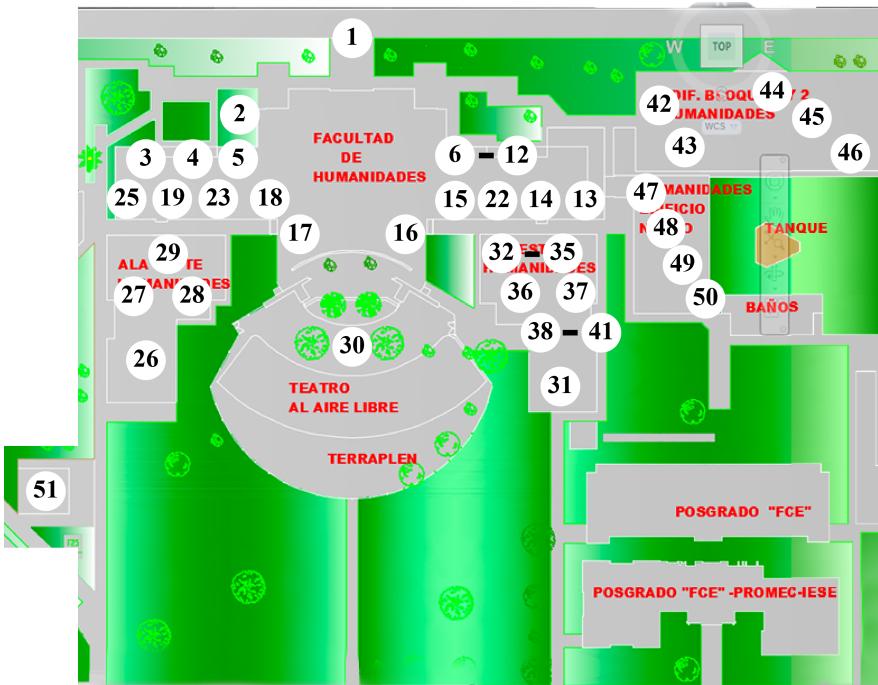


Figura 4.6: Facultad de Humanidades - UMSS

Fuente: Departamento Infraestructura - UMSS

Punto	Detalle
1	Puerta Facultad de Humanidades y Ciencias de la Educación
2	Cajas - Ventanillas FHCE
3	1er Piso - Decanato y Secretaría FHCE
4	1er Piso - Dirección y Secretaría de Ciencias de la Educación
5	1er Piso - Departamento Pedagógico
6	2do Piso - Dirección Académica y Secretaria FHCE
7	1er Piso - Instituto Confucio
8	1er Piso - Programa de Licenciatura en Música
9	1er Piso - Dirección y Secretaría de Psicología
10	1er Piso - Departamento de Coordinación Académica - Psicología
11	1er Piso - Dirección y Secretaría de Lingüística
12	2do Piso - Secretaria Administrativa FHCE
13	IIHCE - Instituto de Investigación de Humanidades y Ciencias de la Educación

Punto	Detalle
14	SOCCED - Sociedad Científica de Ciencias de la Educación
15	Centro de Estudiantes Trabajo Social
16	Centro de Estudiantes Psicología
17	Centro de Estudiantes Ciencias de la Educación
18	Sala Múltiple (Sala Azul)
19	Sociedad Científica de Estudiantes de Comunicación Social
20	EIB - Licenciatura especial en educación bilingüe
21	Aula Magna de la Facultad de Humanidades “Prof. Rafael Gumucio Irigoyen”
22	Aula 213
23	Aula 207
24	Departamento de Servicios Psicológicos
25	Centro de Estudiantes de Lingüística e Idiomas
26	Aula 2 - Especialidad en Innovación Pedagógica en la Docencia Universitaria
27	Biblioteca “Dr. Luis Enrique López-Hurtado”
28	Aula 1
29	1er Piso - Postgrado Humanidades PROEIB Andes
30	Teatro al Aire Libre
31	Aula 221
32	Aula 223
33	Aula 224
34	UTI Facultativa
35	Ventanilla UTI Facultativa
36	1er Piso - Dirección y Secretaría de Trabajo Social
37	1er Piso - Centro de Traducción Laboratorio de Lenguas
38	2do Piso - Aula 242 Sala de Música
39	2do Piso - Dirección y Secretaría de Comunicación Social
40	2do Piso - Aula 4, Postgrado
41	2do Piso - Laboratorio Audiovisual Facultativo
42	Auditorio 1, Auditorio 2
43	Aulas: 1D, 1E, 1F
44	1er Piso - Biblioteca “Paulo Freire”, Salas de Lectura
45	2do Piso - Aulas: 3D, 3E, 3F, 3G, 3H, 3I, 3J, 3K
46	3er Piso - Auditorio Principal FHCE, Aulas: 4H, 4I, 4J, 4K
47	Aulas: 1A, 1B, 1C
48	1er Piso - Aulas: 2A, 2B, 2C
49	2do Piso - Aulas: 3A, 3B, 3C
50	3er Piso - Ludotecas, Aulas: 4A, 4B
H51	PB - Aulas: 228, 229 (ver el multiacademico, figura 4.8)
H52	1er Piso - Aulas: 233, 234, 237, 240, 241, 243 (ver el multiacademico, figura 4.8)
H53	Centro de Estudiantes de Comunicacion Social, Unidad de Produccion Audivisual, Aula 004 (ver el mapa de Tecnología, figura 4.7)

Tabla 4.4: Locaciones de la Fac. Humanidades

4.6. Facultad de Ciencias y Tecnología

Comúnmente conocida como “facultad de Tecnología”, se encuentra en el sector Nor-Este dentro del campus Universitario, se puede encontrar una entrada a la facultad de tecnología sobre la calle al frente del parque *la Torre* y otra entrada sobre la calle MU Lopez debajo de los edificios nuevos de tecnología, dentro del campus Universitario colinda con las facultades de Arquitectura y Humanidades que se encuentran hacia el Sur-Este y Sur-Oeste correspondientemente, en la siguiente figura 4.7 se puede apreciar la facultad de Tecnología.



Figura 4.7: Facultad de Tecnología - UMSS

Fuente: Departamento Infraestructura - UMSS

Punto	Detalle
1	MEMI - Centro de Mejoramiento de la enseñanza de la Matemática y la Informática
2	1er Piso - Postgrado FCyT - UMSS
3	Auditorio MEMI
4	Departamento Informática - Sistemas
5	Laboratorio
6	Parqueo Tecnología
7	Aula 623
8	Aula 622
9	Aula 624
10	Biblioteca FCYT
11	2do Piso - Proyecto CAE, Aula 625C, Aula 625D
12	Auditorio Tecnología
13	Centro de Servicios
14	Instituto de Investigaciones
15	Cafe Docente - Tecnología
16	Departamento de Fisica, Aulas: 618, 619, 619A, 620, 620B, 621, 621A
17	1er Piso - Aula 617
18	Aula 617C, Aula 617B
19	Departamento de Quimica, Aulas: 613, 614, 615, 616, 616A
20	Aula 612
21	Aula 607
22	Departamento de Biologia, Aulas: 606, 608, 609, 608A, 608B, 609A
23	Departamento de Industrial, Aulas: 631, 632
24	Aula 635
25	Aulas: 640, 642, 643
26	1er Piso - Aulas 644, 644A
27	3er Piso - Auditorio Civil
28	Aulas: 651, 652
29	1er Piso - Decanatura FCyT
30	2do Piso - Auditorio Mecanica, Aula magna Civil
31	Edificio ELEKTRO, Aulas: 667A, 667B, 668
32	1er Piso - Aulas: 669A, 669B, 670
33	2do Piso - Aulas: 671, 671A, 671B, 671C, 672
34	3er Piso - Aulas: 674A, 674B, 675
35	Planta Baja - Aulas: 690A, 690B, 690C, 690D
36	1er Piso - Aulas: 691A, 691B, 691C, 691D, 691E, 691F
37	2do Piso - Aulas: 692A, 692B, 692C, 692D, 692E, 692F, 692G, 692H
38	3er Piso - Auditorio 2 FCyt, Aulas: 693A, 693B, 693C, 693D
39	Planta Baja - Aulas: 680-A, 680-B, 680-C, 680-D, 680-E, 680-F, 680-G, 680-H, 680-I, 680-J, 680-K, 680-L, 680-M
40	1er Piso - Aulas: 681-A, 681-B, 681-C, 681-D, 681-E, 681-F, 681-G, 681-H, 681-I, 681-J, 681-K, 681-L
41	2do Piso - Aulas: 682-A, 682-B, 682-C, 682-D, 682-E, 682-F, 682-G, 682-H, 682-I, 682-J, 682-K, 682-L
42	3er Piso - Aulas: 683-A, 683-B, 683-C, 683-D, 683-E, 683-F, 683-G, 683-H, 683-I, 683-J, 683-K, 683-L

Punto	Detalle
43	4to Piso - Biblioteca, Aulas: 684-A, 684-B, 684-C, 684-D, 684-E, 684-F, 684-G, 684-H, 684-I, 684-J, 684-K
44	Centro de Aguas y Saneamiento Ambiental CASA
45	Planta de Alimentos y Productos Naturales CAPN
46	Planta de Tratamientos de Agua
47	Departamento de Infraestructura
48	Departamento de Mantenimiento
49	Centros CTA, CBG, Biotecnología
50	Planta Agroquímico
51	Laboratorio de Materiales
52	Planta Biogás (Biodigestor)
53	Planta Amoniaco
54	Laboratorio Simulación Métodos y Seguridad
55	Sub Estación de Potencia

Tabla 4.5: Locaciones de la Fac. Tecnología

4.7. Multiacademico

En la parte central del campus universitario se encuentra el edificio “multiacademico”, el cual en su mayoría está compuesto por oficinas administrativas pero también se puede encontrar aulas, en la figura 4.8 se puede ver el mapa del edificio y en la tabla 4.6 se describe el detalle de los “lugares” ubicados al interior del multiacademico.



Figura 4.8: Multiacademico - UMSS

Fuente: Departamento Infraestructura - UMSS

Punto	Detalle
1	PB - Unidad de Archivos Diplomas y Títulos (Legalizaciones)
2	PB - Entrega de Diplomas
3	PB - Departamento de Registros e Inscripciones
4	1er Piso - DPA, Dirección de Planificación Académica
5	2do Piso - Planificación del Territorio y del Medio Ambiente
6	3er Piso - Sala Consejo Universitario, Vice-Rectorado, PTAANG (Programa de Titulación de Alumnos Antiguos No Graduados)
7	4to Piso - Jefatura de Personal Administrativo
8	PB - DISU (Dirección de Interacción Social Universitaria)
9	1er Piso - Aula Internado Psicología, Brigadas Universitarias de Salud
10	1er Piso - PROTICS (Programa de Tecnologías de Información y Comunicación Aplicadas a la Educación)
11	2do Piso - PROGEO (Programa de Geografía), Instituto de Investigaciones de Arquitectura, PROMESHA, Biblioteca IIACH
12	2do Piso - CLAS (Centro de Levantamientos aeroespaciales y aplicaciones SIG para el desarrollo sostenible de los recursos naturales)
13	3er Piso - DICyT (Dirección de Investigación Científica y Tecnológica)
14	3er Piso - DUBE (Dirección Universitaria de Bienestar Estudiantil)
15	4to Piso - DUEA (Dirección Universitaria de Evaluación y Acreditación - UMSS)
16	Comedor Universitario
M17	Caja Central UMSS (ver en el mapa de la Fac. Tecnología, figura 4.7)
M18	Oficina Almacenes Adquisiciones (ver en el mapa de la Fac. Tecnología, figura 4.7)

Tabla 4.6: Locaciones del Multiacadémico

4.8. Complejo Deportivo

También podemos encontrar dentro del campus Universitario, el complejo deportivo cuyo mapa se puede ver en la figura 4.9 y el correspondiente detalle en la tabla 4.7



Figura 4.9: Complejo Deportivo - UMSS

Fuente: Departamento Infraestructura - UMSS

Punto	Detalle
1	Cancha de Futbol
2	Canchas Polifuncionales
3	Colegio CENDI
4	Frontón
5	Kiosko
6	Coliseo UMSS
7	Puerta de Entrada

Tabla 4.7: Locaciones del Complejo Deportivo

4.9. Facultades fuera del campus Universitario

Las siguientes facultades no se hallan dentro del campus Universitario “Las Cuadras” por lo que escapan del alcance del presente proyecto de grado, pero se las nombrara para el conocimiento del lector.

FACSO: La Facultad de Ciencias Sociales o “SOCIOLOGÍA” está ubicada Nataniel Aguirre No. S 0360 entre Santivañez y Jordán, también conocida como “Campus Sociología”.

FByF: Facultad de Ciencias Farmacéuticas y Bioquímicas ubicada sobre Av. Aniceto Arce frente Parque La Torre.

FCAPFyV: Facultad de Ciencias Agrícolas, Pecuarias, Forestales y Veterinarias Facultad de Ciencias Agrícolas, Pecuarias y Forestales “Martin Cardenas” (FCAPyP) ubicada en la Avenida Petrolera Km 5.

ODONTOLOGÍA: Facultad de Odontología ubicada dentro el “Campus Salud”, Calle Venezuela y Av. Oquendo.

MEDICINA: Facultad de Medicina ubicada en el “Campus Salud”, Av. Aurelio Melean 379.

FPVA: Facultad Politecnica del Valle Alto ubicada en la Av. Mayor Rocha Provincia Punata.

FDRyT: Facultad de Desarrollo Rural y Territorial ubicada en la Av. Petrolera Km 5.5 carretera antigua a Santa Cruz.

Capítulo 5

Implementación del Proyecto

5.1. Exploración

Como parte del proceso de Exploración se empezó identificando los usuarios que tomarán parte del sistema y los requerimientos funcionales que el sistema deberá proveer, para posteriormente definir las *historias de usuario*.

5.1.1. Identificación de Usuarios

Se puede identificar 3 tipos de usuario en el sistema:

- **Usuario Visitante:** Un usuario visitante puede ver los lugares que están registrados en el sistema, su información y la ruta óptima al lugar.
- **Usuario Registrado:** Un usuario registrado tiene privilegios para agregar lugares y poder editarlos según se requiera.
- **Usuario Administrador:** Un administrador tiene privilegios para remover usuarios registrados que estén haciendo uso indebido del sistema y además pueden ver los reportes que el sistema ofrezca.

5.1.2. Requerimientos Funcionales

Requerimiento	Descripción
RF001	Los usuarios pueden ver los lugares registrados en el sistema en una lista
RF002	Los usuarios pueden filtrar los lugares de acuerdo al nombre del lugar
RF003	Dentro de la información de un lugar se puede observar una foto del lugar, así como el teléfono, una descripción y en qué piso se encuentra.

Requerimiento	Descripción
RF004	El usuario puede ver la ruta óptima hacia un lugar en específico
RF005	Un usuario visitante puede empezar el proceso de registro al sistema
RF006	El sistema tendrá un módulo de autenticación y autorización de usuarios
RF007	Un usuario registrado puede editar la información de un lugar
RF008	Un usuario registrado puede agregar una foto al lugar para una mejor identificación de este.
RF009	Un usuario registrado puede eliminar un lugar de sistema
RF010	Un usuario Administrador puede ver una lista con los usuarios registrados
RF011	Un Administrador puede ver y aceptar la solicitud de regisra de parte de un usuario
RF012	Un Administrador puede eliminar un usuario del sistema.
RF013	Un Administrador puede ver los reportes que el sistema ofrezca.

Tabla 5.1: Requerimientos Funcionales

5.1.3. Historias de Usuario

Las *Historias de Usuario*, están escritas en lenguaje del cliente (no técnico) y serán la pauta para determinar que los requerimientos del sistema están correctamente implementados, en la siguiente tabla 5.2, están nombradas las *historias de usuario* que serán implementadas en el presente proyecto de grado, cada historia de usuario será analizada con más detalle en la *iteración* en la que será implementada.

Código	Nombre de la Historia de Usuario
US01	Implementar la lista de lugares.
US02	Implementar la vista de la información del lugar.
US03	Georeferenciar un lugar sobre el mapa del campus Universitario.
US04	Encontrar la ruta óptima entre 2 puntos dentro del campus Universitario.
US05	Implementar el módulo de Registro de un Visitante.
US07	Editar la información de un lugar.
US06	Añadir más lugares al sistema.
US08	Implementar el módulo para Administrar Usuarios.
US09	Implementar el reporte de los lugares más visitados.

Tabla 5.2: Historias de Usuario

5.2. Planificación de la Entrega

5.2.1. Plan de Entregas

Como parte del proceso XP, el equipo de desarrollo estima el esfuerzo que requerirá la implementación de las *historias de usuario*, tal como se puede apreciar en la tabla 5.3.

Código	Prioridad	Esfuerzo [puntos]
US01	Alta	8
US02	Media	3
US03	Alta	8
US04	Alta	13
US05	Baja	3
US06	Media	8
US07	Media	3
US08	Alta	3
US09	Alta	8

Tabla 5.3: Estimación de las historias de usuario

Como parte del plan de entregas, se definió que cada iteración será de 2 semanas, como se puede ver en la figura 5.1.

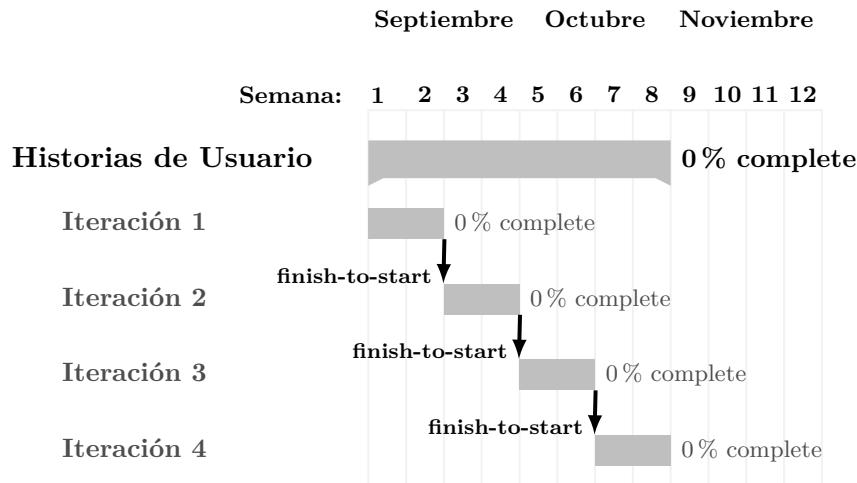


Figura 5.1: Calendario de Entregas

También se definió el orden de implementación de las *historias de usuario* según la prioridad, el esfuerzo y el valor de negocio que el cliente le asignó a las *historias de usuario*, se puede ver en la tabla 5.4,

Iteración	Historia de Usuario	Estimación [días]
Iteración 1	US01	6
	US02	4
Iteración 2	US03	5
	US04	5
Iteración 3	US06	3
	US07	4
Iteración 4	US05	4
	US08	4
	US09	3

Tabla 5.4: Estimación de la implementación de las Historias de Usuario.

Una vez que la fase de *planificación de entrega* es completado, se empieza con la fase de las *iteraciones*.

5.3. Iteración 1

5.3.1. Planificación

En esta etapa se analizaran las Historias de Usuario seleccionadas para esta iteración, y se las dividirá en *tareas de ingeniería*.

En primer lugar se analizará la *historia de usuario* US01, tal como se puede ver en el cuadro 5.5.

Historia de Usuario: US01	Prioridad: Alta
	Riesgo: Medio
Nombre: Implementar la lista de lugares.	
Descripción:	
Yo como visitante Deseo ver una lista de lugares Para encontrar el lugar al que deseo ir	
Criterios de Aceptación:	
Quiero tener los lugares en una base de datos Quiero ver una lista de lugares Quiero filtrar la lista de lugares por el nombre o parte de este	

Tabla 5.5: Historia de Usuario - US01

Número de Tarea: T001	Historia de Usuario: US01
Descripción: Crear un archivo shapefile con información inicial de lugares principales dentro el campus de la UMSS.	
Tipo de Tarea: Desarrollo	Estimación [días]: 1
Programador Responsable: Edmundo Figueroa	

Tabla 5.6: Tarea de Ingeniería - T001

Número de Tarea: T002	Historia de Usuario: US01
Descripción: Crear una base de datos que pueda manejar información geoespacial.	
Tipo de Tarea: Desarrollo	Estimación [días]: 1
Programador Responsable: Edmundo Figueroa	

Tabla 5.7: Tarea de Ingeniería - T002

Número de Tarea: T003	Historia de Usuario: US01
Descripción: Popular la base de datos creada en T002 con la información de T001.	
Tipo de Tarea: Desarrollo	Estimación [dias]: 0.5
Programador Responsable: Edmundo Figueroa	

Tabla 5.8: Tarea de Ingeniería - T003

Número de Tarea: T004	Historia de Usuario: US01
Descripción: Mostrar una lista de los lugares.	
Tipo de Tarea: Desarrollo	Estimación [dias]: 2
Programador Responsable: Edmundo Figueroa	

Tabla 5.9: Tarea de Ingeniería - T004

Número de Tarea: T005	Historia de Usuario: US01
Descripción: Filtrar los lugares ingresando el nombre o parte de este.	
Tipo de Tarea: Desarrollo	Estimación [dias]: 1
Programador Responsable: Edmundo Figueroa	

Tabla 5.10: Tarea de Ingeniería - T005

Posteriormente se analizará la la *historia de usuario* US02, ver el cuadro 5.11.

Historia de Usuario: US02	Prioridad: Baja Riesgo: Alto
Nombre: Implementar la vista de la información del lugar.	
Descripción:	
Yo como visitante Deseo ver la información de un lugar Para decidir si es el lugar que estoy buscando	
Criterios de Aceptación:	
Quiero leer una descripción del lugar Quiero ver un teléfono asociado al lugar Quiero ver en qué piso se encuentra el lugar	

Tabla 5.11: Historia de Usuario - US02

Número de Tarea: T006	Historia de Usuario: US02
Descripción: Mostrar la Descripción del lugar.	
Tipo de Tarea: Desarrollo	Estimación [dias]: 0.5
Programador Responsable: Edmundo Figueroa	

Tabla 5.12: Tarea de Ingeniería - T006

Número de Tarea: T007	Historia de Usuario: US02
Descripción: Mostrar el teléfono del lugar.	
Tipo de Tarea: Desarrollo	Estimación [dias]: 0.5
Programador Responsable: Edmundo Figueroa	

Tabla 5.13: Tarea de Ingeniería - T007

Número de Tarea: T008	Historia de Usuario: US02
Descripción: Mostrar el nivel o el piso del lugar.	
Tipo de Tarea: Desarrollo	Estimación [dias]: 0.5
Programador Responsable: Edmundo Figueroa	

Tabla 5.14: Tarea de Ingeniería - T008

Número de Tarea: T009	Historia de Usuario: US02
Descripción: Mostrar una imagen o foto del lugar.	
Tipo de Tarea: Desarrollo	Estimación [dias]: 2
Programador Responsable: Edmundo Figueroa	

Tabla 5.15: Tarea de Ingeniería - T009

5.3.2. Diseño

- **Diagrama Entidad - Relación:**

En la figura 5.2, se observa el diagrama Entidad - Relación correspondiente a los *lugares* dentro del campus Universitario.

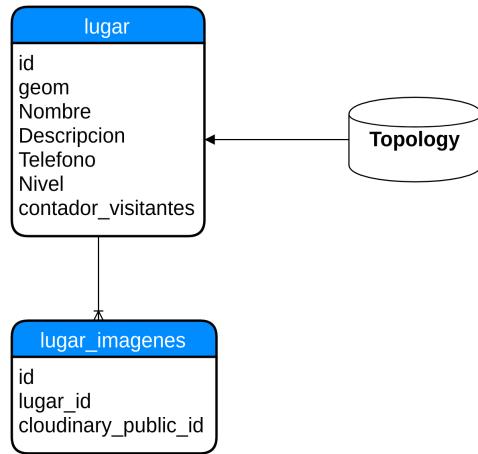


Figura 5.2: Diagrama ER: Lugares

Fuente: Elaboración propia

■ Diagrama de Secuencia:

En la figura 5.3, se observa el diagrama de secuencia correspondiente obtención de la lista e información de lugares.

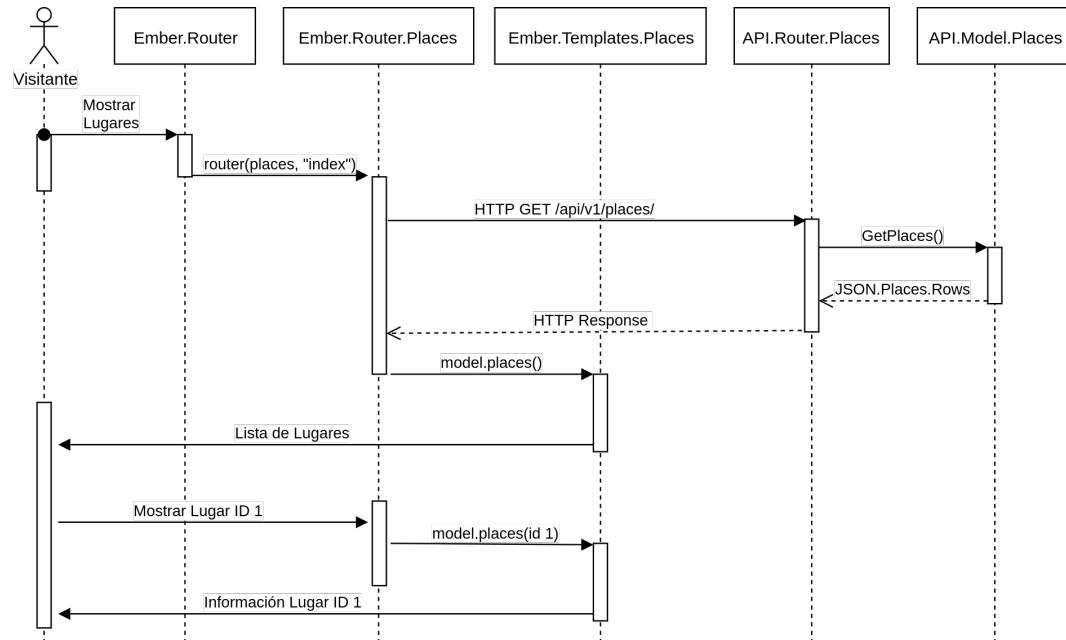


Figura 5.3: Diagrama de Secuencia: Lista e Información de Lugares

Fuente: Elaboración propia

■ Diagrama de Clases:

En la figura 5.4, se observa el diagrama de clases correspondiente a los lugares y la

información de estos.

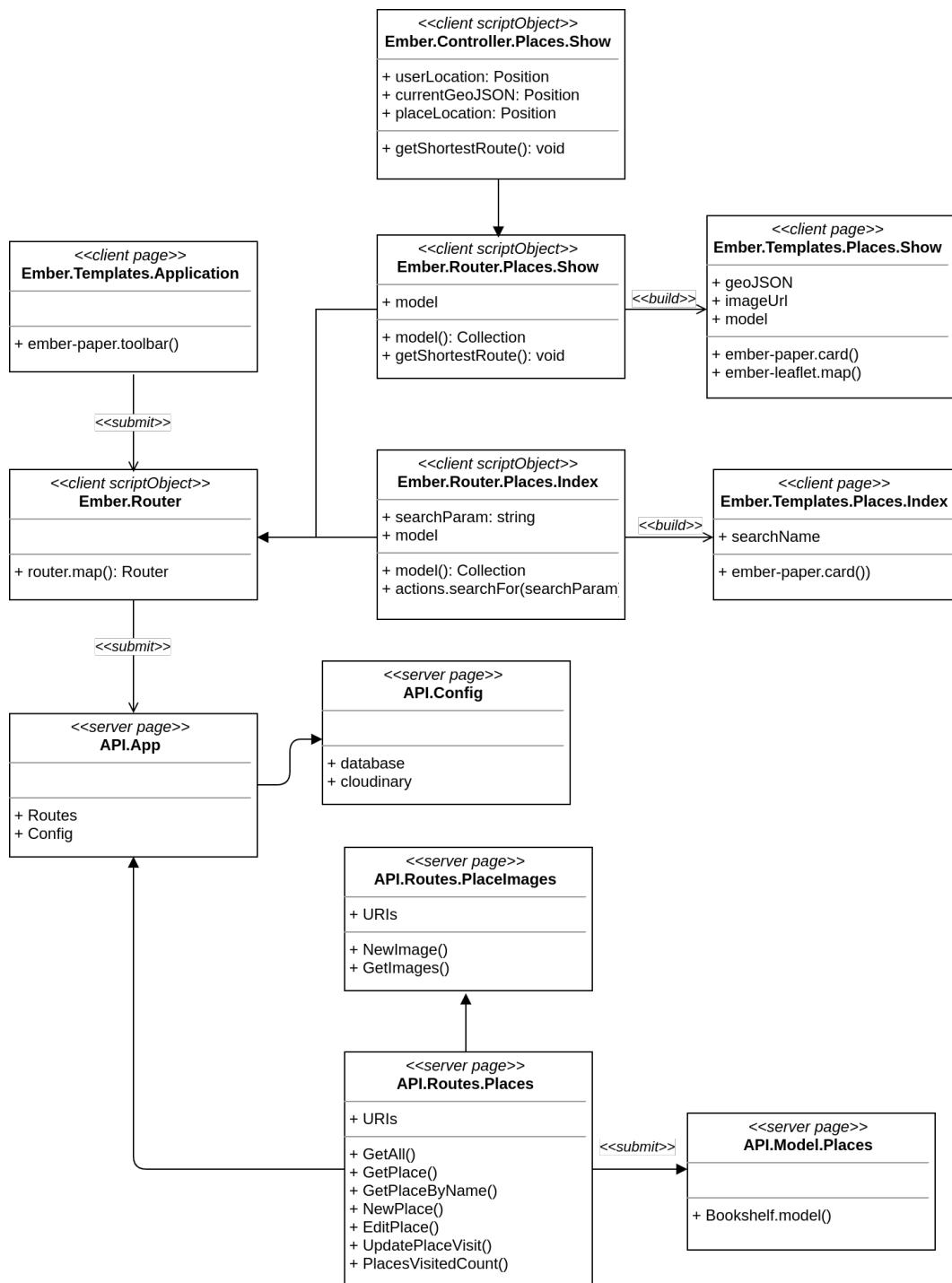


Figura 5.4: Diagrama de Clases: Lugares

Fuente: Elaboración propia

5.3.3. Implementación

5.3.3.1. Acceder a la Información de los lugares mediante un REST API

La información de un *lugar* está almacenada en una base de datos, entonces para poder acceder a esta información se implementó un REST API, que como ya se especificaron sus características, es la más apropiada para construir una aplicación web.

Las peticiones al API llegan a través del protocolo HTTP, en forma de URIs por lo que el servidor implementado con *ExpressJS* tiene que escuchar estas peticiones y actuar en relación al tipo de petición HTTP está llegando, por lo cual es necesario “mapear” las URIs con acciones o métodos, como se puede ver en el código 5.1, los cuales contienen el código necesario para comunicarse con la base de datos.

Code 5.1: Declarando API REST con ExpressJS

```
const router = express.Router();
router.get('/', places.getAll);
router.get('/:id', places.getPlace);
router.post('/', places.newPlace);
router.put('/:id', places.editPlace);
router.delete('/:id', places.deletePlace);

app.use('/api/v1/places', router);
```

En realidad, *ExpressJS* no tiene restricciones a la hora de mapear las URIs, pero para una mejor comprensión del API que se está desarrollando, es necesario seguir convenciones que aseguran que cualquier aplicación pueda consumir la información que el API pueda ofrecer, y un REST API cumple con estas características.

A cada URI mapeada se lo conoce como *endpoint*, por lo que para obtener la lista de lugares que están registradas en el sistema, se usará el *endpoint*: `router.get('/', places.getAll);`, el cual es el indicado para obtener todos los lugares, según la implementación del REST API visto en la tabla 2.1.

El método asociado al *endpoint* contendrá la lógica para comunicarse con la base de datos, por lo que es necesario implementarla.

5.3.3.2. Implementación de la base de datos para almacenar los lugares

La característica especial que tiene esta base de datos, es la de manejar datos geoespaciales, en el caso de los *lugares* estos son representados por el tipo POINT.

La base de datos PostgreSQL por defecto no maneja datos geoespaciales, pero como ya se explicó, añadiendo la extensión PostGIS, ya es posible el manejo de datos geoespaciales, pero es nuestra tarea el comprobar que realmente esté manejando los tipos de datos geoespaciales.

Entonces se procedió a recolectar información de un conjunto de lugares, para esta tarea se utilizó un *GPS Garmin Nuvi 1300*, el cual es un dispositivo de posicionamiento global, que cuenta con la opción de guardar ubicaciones, el dispositivo GPS almacena su información en un archivo *gpx* (que básicamente es un fichero XML estándar usado para compartir datos entre GPS's) y con la ayuda de *QGIS* se generó el archivo shapefile que se utilizó para popular la base de datos con información geoespacial de algunos *lugares* del campus Universitario.

Posteriormente se convierte la información geoespacial almacenada en el shapefile a la base de datos, para lo cual se hizo uso de una herramienta propia de PostgreSQL, que permite la conversión de un archivo shapefile a un archivo sql, el cual se puede ver en el siguiente código.

```
$ shp2pgsql -s 3785 -I -S -c -d ~/Documents/places.shp > places.sql
```

Con el anterior comando se genera un archivo *sql*, el cual es usado para popular la base de datos ya preparada para contener datos geoespaciales, para ingresar la información a la base de datos se utilizó el siguiente comando, propio de PostgreSQL.

```
$ psql -d db_ubikate -U db_admin -f /Documents/places.sql
```

Al finalizar este proceso, se puede ver en la figura 5.5, que la tabla correspondiente a los *lugares* está correctamente poblada con la información geoespacial de los lugares registrados con el dispositivo GPS.

The screenshot shows the pgAdmin III interface. At the top, there are tabs for 'SQL Editor' (which is active) and 'Graphical Query Builder'. Below that is a 'Previous queries' dropdown. The main area contains a SQL query:

```
1  SELECT * FROM place
2  |
```

Below the query is the 'Output pane' tab, which is also active. It contains a 'Data Output' tab and other tabs for 'Explain', 'Messages', and 'History'. The data output pane displays a table of place records:

	gid integer	name character varying(254)	elevation numeric	geom geometry(Point,4326)
1	1	607	2595.90999999999854	0101000020E61000003BE0BA62468950C0DB334B02D46431C0
2	3	642 No Se	2582.210000000000036	0101000020E6100000E46723D74D8950C037E0F3C3086531C0
3	4	Baño 1	2566.82999999999927	0101000020E6100000DC48D922698950C06B11514CDE6431C0
4	5	Baquita	2573.320000000000164	0101000020E6100000AAF1D24D628950C0876BB587BD6431C0
5	6	Biblioteca	2575.000000000000000	0101000020E6100000A7B393C1518950C06DAE9AE7886431C0
6	7	Bloque	2571.869999999999891	0101000020E6100000F9A23D5E488950C01152B7B3AF6431C0
7	8	Canchas	2589.179999999999836	0101000020E6100000E82E89B3228950C0EB025E66D86431C0
8	9	Carrera De Biología	2595.90999999999854	0101000020E610000058C51B99478950C01041D5E8D56431C0
9	10	Centro De Tecn Agroindustrial	2583.409999999999854	0101000020E61000004A5E9D63408950C0B6D8EDB3CA6431C0
10	11	Centro Est Arquit	2569.949999999999818	0101000020E610000036C82423678950C0F06DFAB31F6531C0
11	12	Centro Sistemas	2568.51000000000218	0101000020E61000007E1D3867448950C0CC2A6C06B86431C0
12	14	Comedor	2582.929999999999836	0101000020E6100000E0D74812848950C098F90E7EE26431C0
13	15	Decanato Y Dirección Académica	2583.170000000000073	0101000020E6100001233FB3C468950C0800C1D3BA86431C0

Figura 5.5: Herramienta gráfica de PostgreSQL (*pgAdmin*).

Fuente: Elaboración propia

Una vez implementado el servicio web, necesitamos empezar con el desarrollo del frontend de la aplicación, que como ya se explicó se usará *EmberJS*.

5.3.3.3. Mostrar la lista de lugares

EmberJS consume la información del API implementado, por lo tanto se hará una llamada *GET* al URI *places/*, que dentro de la estructura de *EmberJS* se tiene que implementar en el *router* dedicado al URI correspondiente. El método mostrado en el código 5.2, es el encargado de hacer generar la petición GET que el API está listo para responder con la lista de los lugares registrados en el sistema.

Code 5.2: Método para obtener la lista de lugares del API

```
model() {
  var url = (ENV.APP.API_HOST || '') + '/api/v1/places/';
  return jQuery.ajax({
    url: url,
    type: 'GET'
  });
}
```

Una vez que se obtiene la lista de lugares del servidor, es necesario desplegarlo en el navegador, para tal efecto se implementó el método mostrado en el código 5.3, y se utilizó el template correspondiente al URI *templates/places/index.hbs*.

Code 5.3: Template de la lista de lugares

```
{{#paper-list}}
{{#each model.data as |place|}}
  {{#paper-item class="md-1-line" onClick=(transition-to 'places.show' place)}}
    <div class="md-list-item-text">
      <span>{{place.name}}</span>
    </div>
  {{/paper-item}}
  {{paper-divider}}
{{/each}}
{{/paper-list}}
```

En la anterior implementación se hizo uso de *ember-paper*, que como ya se explicó ayudará en el *look and feel* de la aplicación, tal como se puede observar en la figura 5.6.

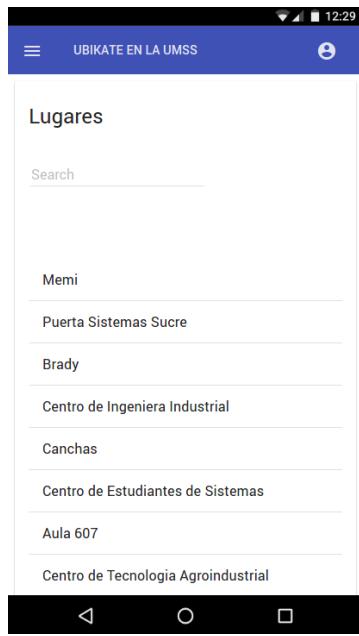


Figura 5.6: Lista de Lugares

Fuente: Elaboración propia

5.3.3.4. Búsqueda de los lugares

Uno de los criterios de aceptación para la implementación de la *historia de usuario* 2, correspondiente a la búsqueda de los lugares, es que sea posible la búsqueda usando el nombre del lugar o parte del mismo. Al implementar esta *historia de usuario* se añadió un *endpoint* adicional a nuestro servicio web, ver el código 5.4, que básicamente realiza una consulta SQL la cual obtiene de la base de datos los lugares que concuerden con el criterio de búsqueda.

Code 5.4: Implementación de la búsqueda de lugares en el Servicio Web

```
router.get('places/search/:name', places.getPlacesByName);
```

5.3.3.5. Mostrar la información del lugar

Para mostrar la información de un *lugar*, es necesario realizar una consulta al URI *places/:id* usando el verbo HTTP *GET*, el cual obtiene la información del *lugar* en formato JSON la cual es mostrada en el navegador, para tal efecto se emplea el template correspondiente al URI, *app/templates/places/show*, la implementación del template se puede ver en el código 5.5.

Code 5.5: Template para mostrar la información de un lugar

```
 {{#text.headline}}{{model.name}}{{/text.headline}}
{{#card.content}}
{{#paper-list}}
  {{model.description}}
  {{/paper-item}}
    {{paper-icon "local_phone"}} {{model.phone}}
  {{/paper-item}}
  {{#paper-item class="md-2-line" }}
    {{paper-icon "layers"}} Piso N# {{model.level}}
  {{/paper-item}}
{{/paper-list}}
{{/card.content}}
```

El resultado en el navegador del template implementado se puede apreciar en la figura 5.7.



Figura 5.7: Vista de la Información de un Lugar.

Fuente: Elaboración propia

5.3.4. Pruebas de Aceptación

Código: CP001

Historia de Usuario: US01

Tipo: Prueba de Funcionalidad - Positiva.

Nombre: Verificar la lista de lugares.

Descripción: Validar que un usuario visitante puede ver la lista de lugares cuando ingresa al menú *lugares*.

Condiciones de Ejecución:

- a. El usuario no debe estar registrado.
- b. Deben existir lugares registrados en el sistema.

Entradas / Pasos de Ejecución:

1. Hacer tap sobre el botón *menú* en la esquina superior-izquierda.
2. Seleccionar el menú *lugares*.

Resultado Esperado: El Usuario debe ver una lista con los lugares registrados en la Condición de Ejecución b.

Evaluación de la Prueba: Prueba exitosa.

Tabla 5.16: Prueba de Aceptación - CP001

Código: CP002

Historia de Usuario: US01

Tipo: Prueba de Funcionalidad - Positiva.

Nombre: Verificar la búsqueda de lugares.

Descripción: Validar que un usuario puede filtrar un lugar de la lista mediante el nombre.

Condiciones de Ejecución: Ingresar en la base de datos el lugar con nombre “MEMI”.

Entradas / Pasos de Ejecución:

1. Hacer tap sobre el botón *menú* en la esquina superior-izquierda.
2. Seleccionar el menú *lugares*.
3. Ingresar el nombre “MEMI” en el cajón de búsqueda.

Resultado Esperado: Se debe mostrar un solo ítem en la lista de lugares con el nombre “MEMI” desplegado.

Evaluación de la Prueba: Prueba exitosa.

Tabla 5.17: Prueba de Aceptación - CP002

Código: CP003	Historia de Usuario: US02
Tipo: Prueba de Funcionalidad - Positiva.	
Nombre: Verificar la información de un lugar.	
Descripción: Validar que un usuario puede ver la descripción, el teléfono, el nivel y la foto de un lugar.	
Condiciones de Ejecución: Ingresar en la base de datos el lugar con nombre “MEMI” con su información completa.	
Entradas / Pasos de Ejecución:	
1. Hacer tap sobre el botón <i>menú</i> en la esquina superior-izquierda. 2. Seleccionar el menú <i>lugares</i> . 3. En la lista de lugares seleccionar el ítem “MEMI”.	
Resultado Esperado: Se debe mostrar una pantalla con la foto del lugar “MEMI”, su descripción, el teléfono y el nivel.	
Evaluación de la Prueba: Prueba exitosa.	

Tabla 5.18: Prueba de Aceptación - CP003

Código: CP004	Historia de Usuario: US01
Tipo: Prueba de Funcionalidad - Negativa.	
Nombre: Verificar la lista de lugares cuando se busca un lugar no registrado.	
Descripción: Validar que la lista de lugares muestre una lista vacía cuando se ingresa el nombre de un lugar no registrado.	
Condiciones de Ejecución: En la base de datos no debe existir el lugar con nombre “Aula 123”.	
Entradas / Pasos de Ejecución:	
1. Hacer tap sobre el botón <i>menú</i> en la esquina superior-izquierda. 2. Seleccionar el menú <i>lugares</i> . 3. En el cajón de búsqueda ingresar “Aula 123”.	
Resultado Esperado: La lista de lugares debe mostrar una lista vacía.	
Evaluación de la Prueba: Prueba exitosa.	

Tabla 5.19: Prueba de Aceptación - CP004

Código: CP005

Historia de Usuario: US02

Tipo: Prueba de Funcionalidad - Negativa.

Nombre: Verificar la información de un lugar mediante el URI.

Descripción: Validar que la página con la información de un lugar pueda ser accedida mediante la URL correcta.

Condiciones de Ejecución: En la base de datos debe existir el lugar con *id* 50.

Entradas / Pasos de Ejecución:

1. Ingresar en el URL del navegador la dirección, */places/50*.
2. Aceptar el URL ingresado.

Resultado Esperado: La pagina con información del lugar con *id* 50 debe ser mostrado.

Evaluación de la Prueba: Prueba exitosa.

Tabla 5.20: Prueba de Aceptación - CP005

Código: CP006

Historia de Usuario: US02

Tipo: Prueba de Usabilidad.

Nombre: Verificar que el *Menú* sea desplegado dinámicamente.

Descripción: Validar que el *Menú* se muestra presionando el icono *menú* en la esquina superior-izquierda y al seleccionar alguna opción el *Menú* se oculte.

Condiciones de Ejecución: Sin condición.

Entradas / Pasos de Ejecución:

1. Hacer tap sobre el botón *menú* en la esquina superior-izquierda de la pantalla.
2. Seleccionar el menú *lugares*.

Resultado Esperado: El *Menú* debe ser desplegado y colapsado sin dejar que el *Menú* oculte la pantalla del navegador.

Evaluación de la Prueba: Prueba exitosa.

Tabla 5.21: Prueba de Aceptación - CP006

5.3.4.1. Resultado de las pruebas de la Iteración 1

Al finalizar la Iteración 1, se ejecutaron todas las pruebas escritas durante la presente iteración, en el cuadro 5.22 se puede ver el detalle.

Código	Título de la Prueba	Resultado
CP001	Verificar la lista de lugares.	Exitoso
CP002	Verificar la búsqueda de lugares.	Exitoso
CP003	Verificar la información de un lugar.	Exitoso
CP004	Verificar la lista de lugares cuando se busca un lugar no registrado.	Exitoso
CP005	Verificar la información de un lugar mediante el URI.	Exitoso
CP006	Verificar que el <i>Menú</i> sea desplegado dinámicamente.	Exitoso

Tabla 5.22: Pruebas de regresión de la Iteración 1

- Se ejecutaron 5 pruebas de funcionalidad, todas pasaron exitosamente.
- Se ejecutó 1 prueba de usabilidad, pasó exitosamente.

5.4. Iteración 2

5.4.1. Planificación

A continuación se analizará la *historia de usuario* US03, ver el cuadro 5.23.

Historia de Usuario: US03	Prioridad: Baja Riesgo: Alto
Nombre: Georeferenciar un lugar sobre el mapa del campus Universitario.	
Descripción:	
Yo como visitante. Deseo ver el lugar en un mapa. Para saber en qué parte del campus se encuentra el lugar.	
Criterios de Aceptación:	
Deseo ver sobre un mapa un punto del lugar buscado a donde quiero ir. Quiero ver un marcador sobre el lugar que estoy buscando con alguna información para asegurarme que es a donde quiero ir.	

Tabla 5.23: Historia de Usuario - US03

Número de Tarea: T010	Historia de Usuario: US03
Descripción: Investigar e instalar una herramienta que permita usar un servicio de mapas.	
Tipo de Tarea: Desarrollo	Estimación [dias]: 1
Programador Responsable: Edmundo Figueroa	

Tabla 5.24: Tarea de Ingeniería - T010

Número de Tarea: T011	Historia de Usuario: US03
Descripción: Mostrar el mapa del campus Universitario.	
Tipo de Tarea: Desarrollo	Estimación [dias]: 0.5
Programador Responsable: Edmundo Figueroa	

Tabla 5.25: Tarea de Ingeniería - T011

Número de Tarea: T012	Historia de Usuario: US03
Descripción: Mostrar un marcador en el mapa sobre el lugar.	
Tipo de Tarea: Desarrollo	Estimación [dias]: 0.5
Programador Responsable: Edmundo Figueroa	

Tabla 5.26: Tarea de Ingeniería - T012

Número de Tarea: T013	Historia de Usuario: US03
Descripción: Mostrar en el marcador la información básica del lugar, por ejemplo el nombre y el piso.	
Tipo de Tarea: Desarrollo	Estimación [dias]: 0.5
Programador Responsable: Edmundo Figueroa	

Tabla 5.27: Tarea de Ingeniería - T013

A continuación se analizará la *historia de usuario* US04, ver el cuadro 5.28.

Historia de Usuario: US04	Prioridad: Alta
	Riesgo: Alto

Nombre: Encontrar la ruta óptima entre 2 puntos dentro del campus Universitario.

Descripción:

- Yo como visitante
- Deseo ver una ruta sobre el mapa
- Para encontrar el lugar de forma rápida

Criterios de Aceptación:

Deseo ver sobre un mapa un punto del lugar actual donde me encuentro

Deseo ver una línea roja que muestre la ruta más corta para llegar de mi ubicación al lugar donde quiero ir

Tabla 5.28: Historia de Usuario - US04

Número de Tarea: T014	Historia de Usuario: US04
Descripción: Crear un archivo shapefile con las rutas dentro el campus de la UMSS.	
Tipo de Tarea: Desarrollo	Estimación [dias]: 2
Programador Responsable: Edmundo Figueroa	

Tabla 5.29: Tarea de Ingeniería - T014

Número de Tarea: T015	Historia de Usuario: US04
Descripción: Preparar la base de datos para manejar información geográfica de rutas.	
Tipo de Tarea: Desarrollo	Estimación [dias]: 1
Programador Responsable: Edmundo Figueroa	

Tabla 5.30: Tarea de Ingeniería - T015

Número de Tarea: T016	Historia de Usuario: US04
Descripción: Mostrar un marcador en el mapa sobre la posición actual del usuario.	
Tipo de Tarea: Desarrollo	Estimación [dias]: 0.5
Programador Responsable: Edmundo Figueroa	

Tabla 5.31: Tarea de Ingeniería - T016

Número de Tarea: T017	Historia de Usuario: US04
Descripción: Desarrollar un módulo que encuentra la ruta más corta usando la base de datos con información geográfica ruteable.	
Tipo de Tarea: Desarrollo	Estimación [dias]: 2
Programador Responsable: Edmundo Figueroa	

Tabla 5.32: Tarea de Ingeniería - T017

Número de Tarea: T018	Historia de Usuario: US04
Descripción: Mostrar una línea roja que une el marcador de la posición del usuario con el marcador del lugar.	
Tipo de Tarea: Desarrollo	Estimación [dias]: 1
Programador Responsable: Edmundo Figueroa	

Tabla 5.33: Tarea de Ingeniería - T018

5.4.2. Diseño

- **Diagrama Entidad - Relación:**

En la figura 5.8, se observa el diagrama Entidad - Relación correspondiente a los *caminos* o mapa de rutas dentro del campus Universitario.

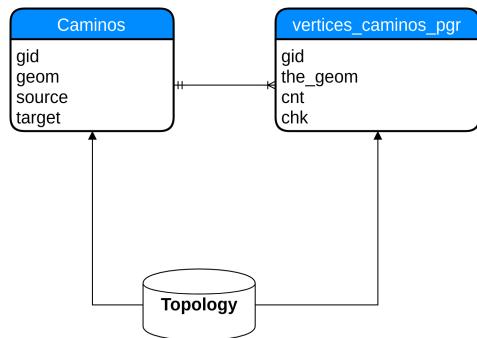


Figura 5.8: Diagrama ER: Caminos

Fuente: Elaboración propia

■ Diagrama de Secuencia:

En la figura 5.9, se observa el diagrama de secuencia correspondiente obtención de la ruta óptima.

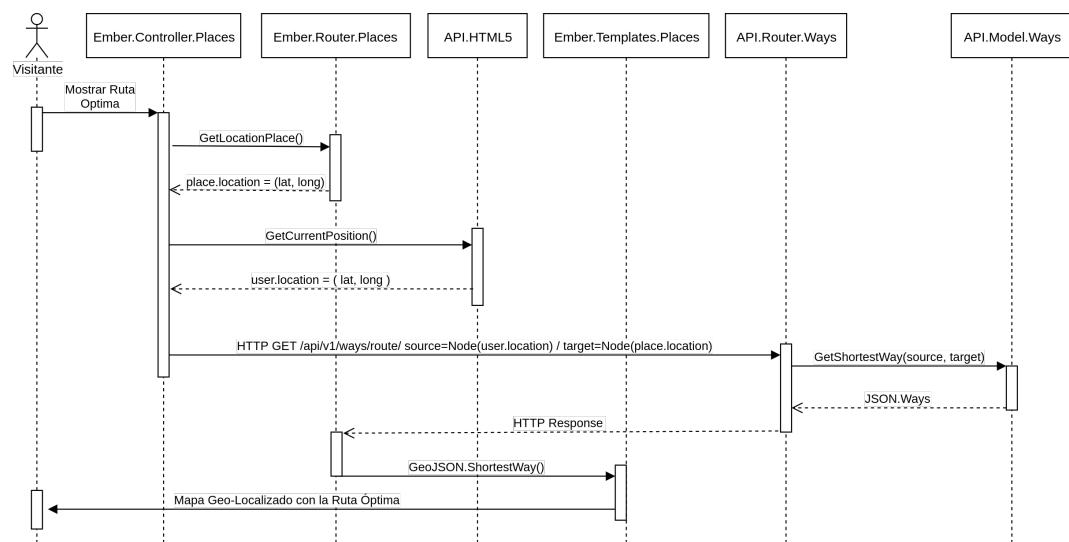


Figura 5.9: Diagrama de Secuencia: Ruta Óptima

Fuente: Elaboración propia

■ Diagrama de Clases:

En la figura 5.10, se observa el diagrama de clases correspondiente a los caminos del campus Universitario.

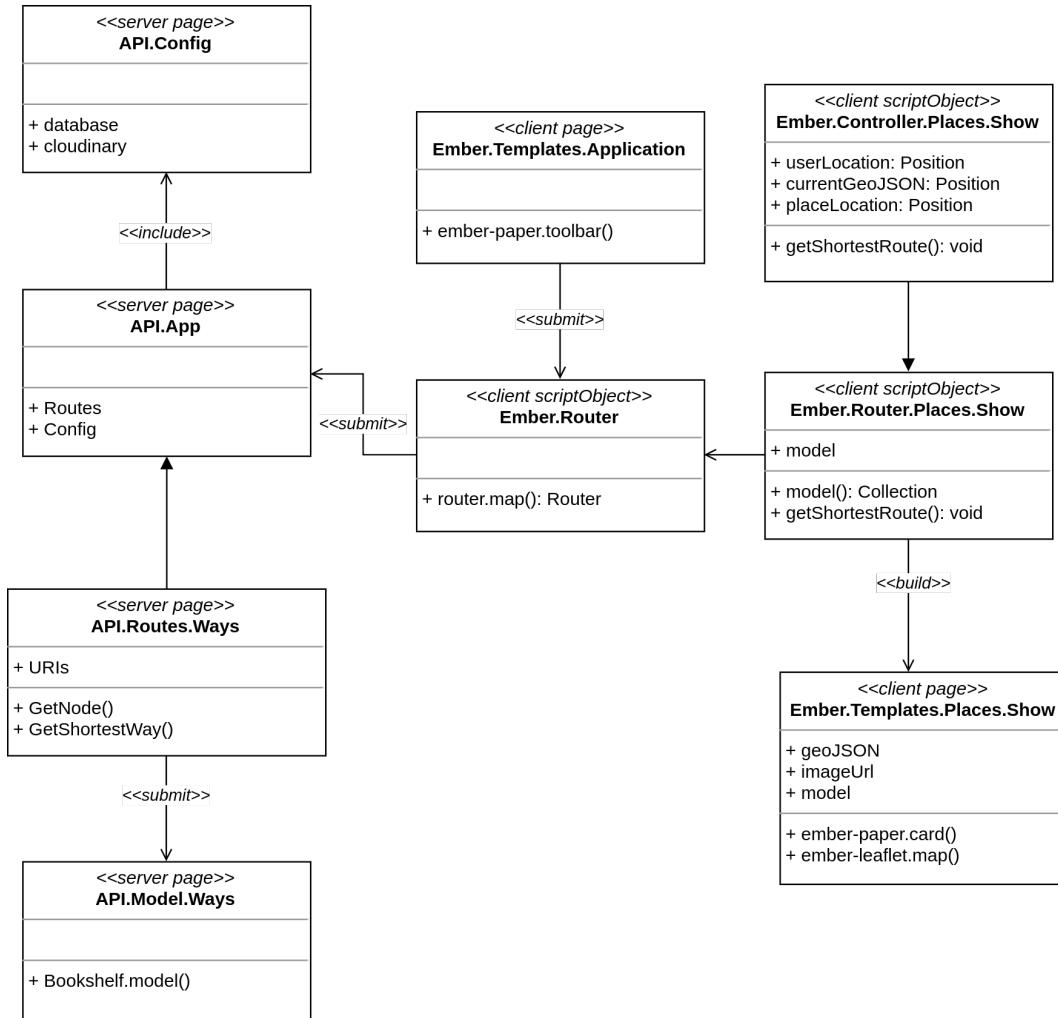


Figura 5.10: Diagrama de Clases: Caminos

Fuente: Elaboración propia

5.4.3. Implementación

5.4.3.1. Geolocalizar los Lugares

El hecho de localizar un lugar significa que se lo va a ubicar sobre un mapa gracias a sus coordenadas, tales que se encuentran almacenadas en la base de datos, por lo tanto es necesario hacer una petición al *endpoint* del REST API dedicado a obtener la información de un lugar en específico, el *endpoint* implementado se lo puede ver a continuación.

```
router.get('api/v1/places/:id', places.getPlace);
```

El método *places.getPlace* (código 5.6) obtiene las coordenadas en formato GeoJSON, el cual es el formato estándar al trabajar con información geográfica en plataformas que usan JavaScript, este objeto GeoJSON es usado para mostar el *lugar* en el navegador.

Code 5.6: Método para obtener la información de un lugar.

```
var getPlace = (req, res) => {
  var id = req.params.id;
  var raw = 'SELECT
    ST_AsGeoJSON(p.geom)::json As geometry,
    p.name,
    p.description,
    p.phone,
    p.level,
    p.gid As id,
    (string_agg(i.cloudinary_public_id, ',')) as images
  FROM place p
  LEFT JOIN place_images i ON p.gid = i.place_id
  WHERE p.gid = ${id}
  GROUP BY p.gid;';

  Bookshelf.knex.raw(raw)
    .then((data) => {
      res.json(data.rows[0]);
    })
    .catch((error) => {
      console.log(error);
      res.send("Error");
    });
};
```

Una vez que ya se tienen las coordenadas del *lugar* solo faltaría ubicarlo sobre un mapa, para resolver esta tarea se utilizó *ember-leaflet*, una librería creada para manipular mapas y diseñada especialmente para su implementación en dispositivos móviles.

Para instalar esta librería solo se necesita ejecutar el siguiente comando y posteriormente ya se puede empezar a utilizarla.

```
$ ember install ember-leaflet
```

En *EmberJS* las coordenadas son procesadas dentro del *controlador*, el cual maneja las variables implementadas en el *template*, tal como se puede observar en el siguiente código 5.7, las variables: *lat* y *lng* llamadas dentro del tag *#leaflet-map*, que son la latitud y longitud respectivamente del punto georeferenciado.

Code 5.7: Método para obtener la información de un lugar.

```
 {{#leaflet-map lat=lat lng=lng zoom=zoom}}
  {{tile-layer url="http://{{s}}.tile.openstreetmap.fr/hot/{{z}}/{{x}}/{{y}}.png" }}
  {{#marker-layer location=location}}
    <h3>{{model.name}}</h3>
    {{model.description}} <br>
    <strong>telf:</strong> {{model.phone}} <br>
    <strong>piso </strong>#{{model.level}}
  {{/marker-layer}}
{{/leaflet-map}}
```

Ember-leaflet permite manejar distintos servidores de mapas, pero se escogió *Open Street Maps* por su facilidad de uso, ser *open source* y no necesitar adquirir licencias para su uso. El resultado del anterior código se puede apreciar en la figura 5.11.



Figura 5.11: Mapa mostrado con la ayuda de *ember-leaflet*

Fuente: Elaboración propia.

Ember-leaflet también permite incluir elementos en el mapa que se está renderizando, por ejemplo como parte de las tareas de esta iteración es necesario añadir un *marcador* sobre las coordenadas del *lugar* y *ember-leaflet* permite hacer tal cosa, así como también personalizar el marcador incluyendo la información del lugar en un cajón de información que se despliega sobre el marcador al ser presionado, como se puede ver en el código 5.7, dentro del tag **#marker-layer** se está dando formato a la información del lugar y el resultado de esta tarea se puede apreciar en la figura 5.12.



Figura 5.12: Marcador con la información de un lugar.

Fuente: Elaboración propia.

5.4.3.2. Geolocalizar a los Usuarios

Para encontrar la ruta más corta entre el *usuario* y un *lugar*, es necesario conocer la locación o las coordenadas del usuario y para tal efecto se utilizó el API de geolocalización propio de HTML5.

La especificación de HTML5 indica que el navegador puede acceder y usar los recursos nativos de un *smartphone*, en este caso la locación del usuario es encontrada mediante la triangulación de Coordenadas por GPS (el mas exacto a la hora de encontrar la locación del dispositivo) o Wi-Fi, dependiendo de las características del dispositivo móvil.

En el navegador solo es necesaria la ejecución de la siguiente línea para poder obtener la posición actual del usuario usando el API de geolocalización de HTML5.

```
var coords = Geolocation.getCurrentPosition();
var latitud = coords.latitude;
var longitud = coords.longitude;
```

Donde la *latitud* y *longitud* obtenidas son fácilmente trasladadas al mapa usando *ember-leaflet* y mediante un marcador se puede visualizar la ubicación actual del usuario, como se puede ver en la siguiente figura 5.13.

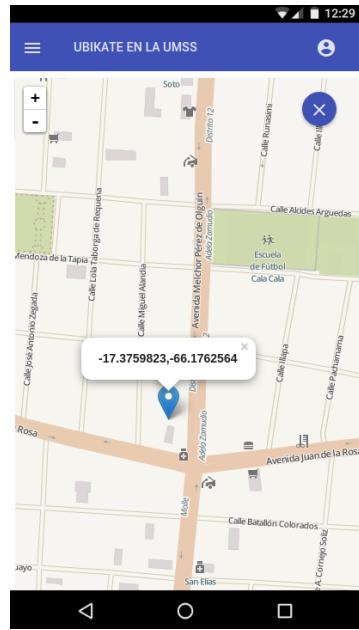


Figura 5.13: Marcador sobre la posición actual del usuario.

Fuente: Elaboración propia.

5.4.3.3. Encontrar la ruta óptima entre el usuario y el lugar

Para solucionar este problema se necesita geolocalizar todas rutas que existen dentro del campus Universitario.

Por lo que se procedió a caminar a través del campus Universitario con un *GPS Garmin Nuvi 1300*, que es un dispositivo GPS básico pero puede guardar información geográfica, se recorrió los principales caminos que existen e interconectan las distintas facultades y oficinas dentro del campus universitario. Una vez realizado este recorrido, se procedió a extraer la información del dispositivo GPS y exportarlo a un archivo *shapefile*, para esta tarea se utilizó *QGis*.

Una vez exportado a un archivo *shapefile* fue necesario editar la información recogida con el dispositivo GPS porque la ruta recogida por el dispositivo es una línea única, y para usarla como un *mapa de rutas* o *red topológica* necesaria para buscar una ruta óptima entre 2 nodos de la red, es necesario que la línea única sea dividida o separada en muchas líneas, este paso se lo realizó con *QGis* con la opción de *explode lines*.

Una vez realizado el paso anterior se obtiene un conjunto de *LINESTRINGS* o líneas únicas, resultado que se puede apreciar en la figura 5.14.

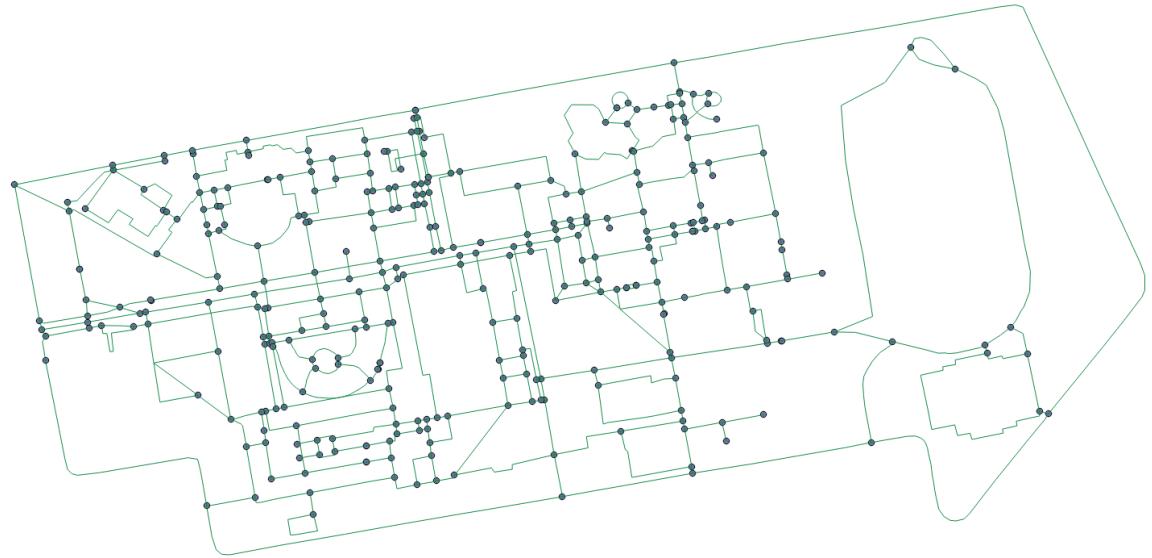


Figura 5.14: Mapa de rutas del campus Universitario.

Fuente: Elaboración propia

El mapa generado con las rutas cumple con las características de un *grafo ponderado no-dirigido* en el que el costo o peso de las aristas es la distancia entre los nodos y los nodos son los puntos de intersección de las rutas. Por lo tanto se puede implementar el algoritmo de *Dijkstra* para encontrar el camino mínimo.

Primeramente se preparó la base de datos añadiendo la extensión *pgRouting*, herramienta que ya tiene implementada el algoritmo de *Dijkstra* que como ya se explicó es una herramienta dedicada a encontrar la ruta óptima entre 2 nodos en una *red topológica*.

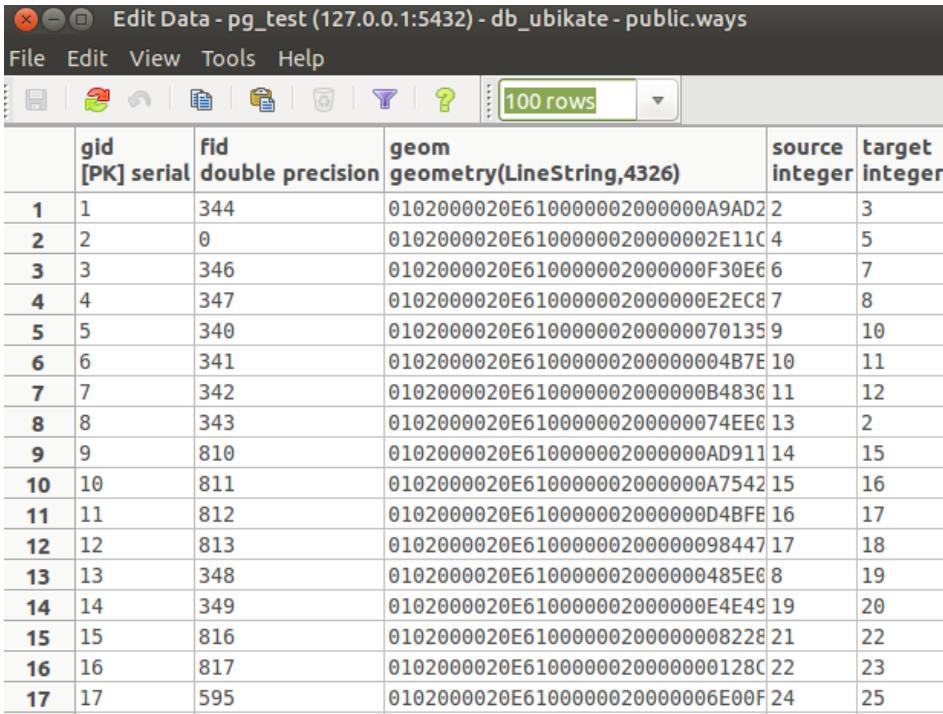
Posteriormente se creó la tabla *ways* para que contenga las rutas georeferenciadas, para preparar esta tabla para que soporte las funciones instaladas por *pgRouting* es necesario ejecutar un query propio de *pgRouting* el cual se puede ver en el siguiente código, y tiene como objetivo analizar los datos geo-espaciales de la tabla y añadirle una *topología*.

```
select pgr_createTopology('ways', 0.00000001, 'geom', 'gid');
```

Dentro lo que es la *topología geoespacial* se denomina *topología de red* a la representación de las relaciones entre segmentos en una red lineal o una colección de segmentos de línea. (Blake, 2007)

En un *SIG* la topología ayuda a mejorar el análisis de datos geo-espaciales, para resolver el problema de la ruta corta *pgRouting* genera una *topología de red* usando los datos que

existen en la tabla *ways*, es necesario ejecutar una instrucción, la que se muestra a continuación y *pgRouting* se encarga de llenar los datos que se pueden observar en la figura 5.15, las columnas *source* y *target* son populadas con el análisis topológico y en la figura 5.16, se puede observar que la tabla *ways_vertices_pgr* es creada enteramente en la ejecución de la instrucción.



The screenshot shows a database viewer window titled "Edit Data - pg_test (127.0.0.1:5432) - db_ubikate - public.ways". The table has the following structure:

	gid [PK] serial	fid double precision	geom geometry(LineString,4326)	source integer	target integer
1	1	344	0102000020E61000002000000A9AD	2	3
2	2	0	0102000020E61000002000002E11C	4	5
3	3	346	0102000020E6100000200000F30E6	6	7
4	4	347	0102000020E6100000200000E2EC8	7	8
5	5	340	0102000020E610000020000070135	9	10
6	6	341	0102000020E610000020000004B7E	10	11
7	7	342	0102000020E61000002000000B4838	11	12
8	8	343	0102000020E610000020000074EE6	13	2
9	9	810	0102000020E6100000200000AD911	14	15
10	10	811	0102000020E6100000200000A7542	15	16
11	11	812	0102000020E61000002000000D4BFE	16	17
12	12	813	0102000020E610000020000098447	17	18
13	13	348	0102000020E6100000200000485E8	8	19
14	14	349	0102000020E6100000200000E4E49	19	20
15	15	816	0102000020E610000020000008228	21	22
16	16	817	0102000020E61000002000000128C	22	23
17	17	595	0102000020E61000002000006E00F	24	25

Figura 5.15: Vista de la tabla *ways*.

Fuente: Elaboración propia

En la figura 5.15 se puede apreciar que cada fila es una parte de la línea original obtenida por el dispositivo GPS y explosionada por QGIS, hay que notar que las columnas *source* y *target* hacen referencia a los nodos o vértices de la línea entre sus extremos, esta línea es una arista dentro del grafo generado.

En la siguiente figura 5.16 se observa la tabla *ways_vertices_pgr* que contiene los vértices creados a partir del análisis de los datos en la tabla *ways*.

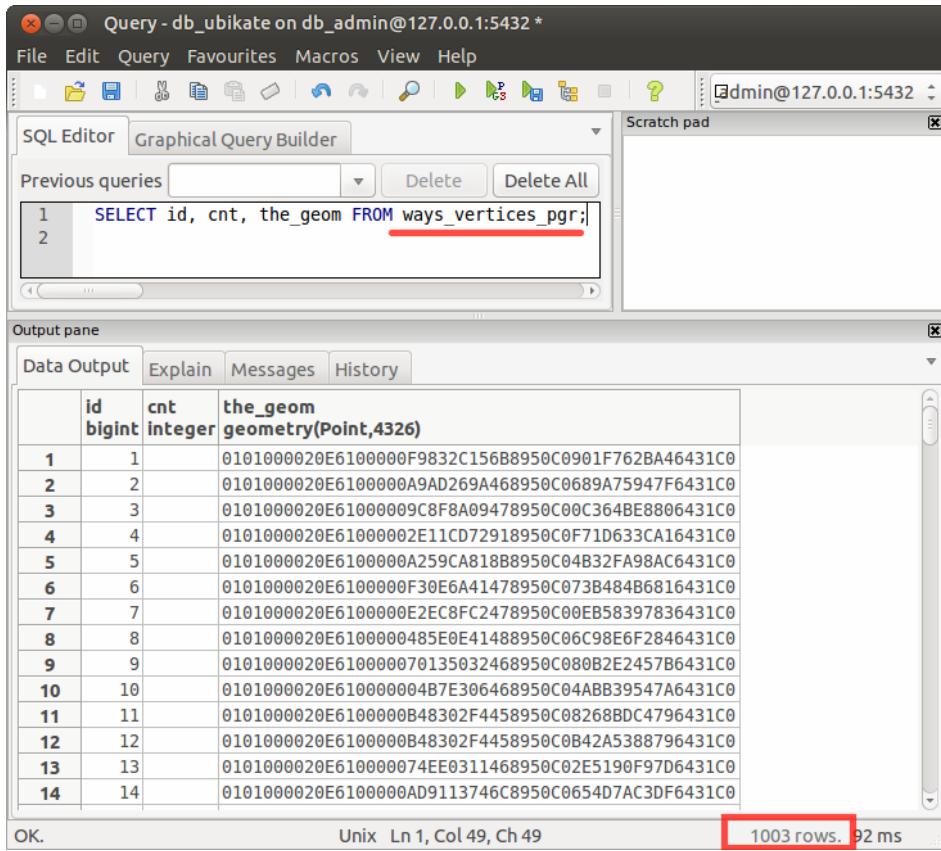


Figura 5.16: Vista de la tabla *ways_vertices_pgr*.

Fuente: Elaboración propia

Para entender los datos generados por *pgRouting* hay leer la información de las 2 tablas, por ejemplo en la primera fila (gid 1) de la tabla *ways*, se observa que el contenido de la columna *source* es igual a **2** y *target* es igual a **3**, eso quiere decir que los vértices de la arista son los vértices con **id** 2 y 3 respectivamente de la tabla *ways_vertices_pgr*.

PgRouting ya tiene implementado el algoritmo de *Dijkstra*, pero solo puede ser utilizado cuando ya se le haya añadido una topología a la tabla con las rutas. Una vez cumplido con este requisito se puede hacer uso del método implementado (ver el código 5.8) por *pgRouting*, esta consulta SQL utilizada encuentra la ruta óptima entre 2 nodos de la red topológica y utiliza la distancia entre nodos para calcular la ruta óptima.

Code 5.8: Algoritmo de Dijkstra implementado en *pRouting*

```
SELECT seq, id1 AS node, id2 AS edge, cost
FROM pgr_dijkstra(SELECT gid AS id,
                  source::integer,
                  target::integer,
```

```

        st_length(geom) AS cost
    FROM public.ways, targetId, sourceId, false, false);

```

Los nodos de destino y origen, *targetId* y *sourceId* respectivamente, son datos obtenidos por una combinación de acciones ya que los nodos son propios del mapa de rutas ubicados en la tabla *ways* y el punto destino que se están utilizando es en realidad un *lugar* del campus Universitario ubicado en la tabla *places* y el punto origen es la posición actual del usuario.

Por lo que para obtener los nodos utilizados en la consulta SQL previamente mostrada, es necesario encontrar los nodos “más” cercanos a estos puntos y gracias *PostGIS* encontrar el nodo más cercano es sencillo, en la siguiente consulta SQL se puede ver esta funcionalidad donde *lon* y *lat* son la longitud y latitud de los puntos origen o destino.

```

SELECT id
FROM ways_vertices_pgr
ORDER BY the_geom <-> ST_GeometryFromText('POINT(lon lat)', 3857)
LIMIT 1

```

Una vez ejecutado el método *pgr_dijkstra* se obtiene un conjunto de líneas, las cuales son un conjunto de latitudes y longitudes que representan la ruta más corta entre el punto origen y el punto destino, pero esta información realmente no dice nada a la persona que lo lee por lo tanto es necesario que el resultado de la anterior consulta SQL sea procesada para poder ser consumida desde el navegador y la mejor forma para apreciar la ruta es mostrarla sobre el *mapa*, todo este proceso es llevada a cabo en el servidor y entregada al cliente en formato GeoJSON.

A continuación se puede apreciar el *request* que el frontend hace al API y el objeto GeoJSON que es recibido, este objeto contiene la información geoespacial necesaria para “dibujar” la línea roja entre 2 puntos georeferenciados, uno de los puntos es donde se encuentra el usuario (*sourceData*) y el otro punto es el lugar (*targetData*).

Code 5.9: Llamada al *endpoint* que retorna la ruta óptima.

```

GET /api/v1/ways/route/930/77 200 276.217 ms - 3911

$ curl http://localhost:3000/api/v1/ways/route/930/77 | python -m json.tool
{
  "features": [
    {
      "geometry": {
        "coordinates": [
          [
            [
              -66.1467397848201,
              -17.3935321732846
            ],
            [
              -66.1467190789842,
              -17.3935294725234
            ]
          ]
        ]
      }
    }
  ]
}

```

```

        ],
        "type": "LineString"
    },
    "type": "Feature"
},
.
.
.
]
}

```

Una vez que se tiene los datos, es necesario representarlo en el mapa y para tal efecto se seguirá utilizando *ember-leaflet* y que con el tag *geojson-layer* (ver el siguiente código) se dibuja sobre el mapa una línea roja que representa la ruta más corta entre el punto donde se encuentra el usuario y el *lugar*, tal como se puede apreciar en la figura 5.17.

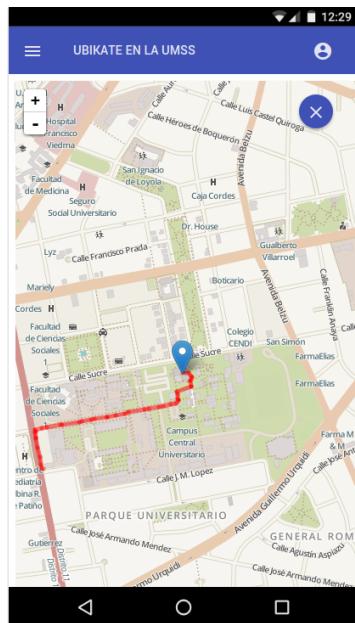


Figura 5.17: Ruta más corta dibujada con una línea roja.

Fuente: Elaboración propia.

5.4.4. Pruebas de Aceptación

Código: CP007

Historia de Usuario: US03

Tipo: Prueba de Funcionalidad - Positiva.

Nombre: Verificar el marcador del lugar sobre el mapa del campus Universitario.

Descripción: Validar que un usuario puede ver un marcador sobre el mapa indicando la posición del lugar dentro del campus Universitario cuando se pide la ruta óptima al lugar.

Condiciones de Ejecución: Un lugar registrado dentro del campus Universitario.

Entradas / Pasos de Ejecución:

1. Seleccionar el menú *Lugares*.
2. Buscar el lugar registrado.
3. Seleccionar la opción para buscar la ruta al lugar.

Resultado Esperado: Una vista del mapa del campus Universitario debe ser desplegado con un marcador sobre la posición del lugar en el centro de la pantalla.

Evaluación de la Prueba: Prueba exitosa.

Tabla 5.34: Prueba de Aceptación - CP007

Código: CP008

Historia de Usuario: US03

Tipo: Prueba de Funcionalidad - Positiva.

Nombre: Verificar que el marcador del lugar muestre la información.

Descripción: Validar que un usuario puede ver un *tooltip* sobre el marcador mostrando el nombre del lugar, teléfono y nivel.

Condiciones de Ejecución: Un lugar registrado dentro del campus Universitario.

Entradas / Pasos de Ejecución:

1. Seleccionar el menú *Lugares*.
2. Buscar el lugar registrado.
3. Seleccionar la opción para buscar la ruta al lugar.
4. Presionar el marcador mostrado sobre el mapa.

Resultado Esperado: Un *tooltip* o cajón de información debe de aparecer sobre el marcador mostrando el nombre, teléfono y nivel del lugar.

Evaluación de la Prueba: Prueba exitosa.

Tabla 5.35: Prueba de Aceptación - CP008

Código: CP009 **Historia de Usuario:** US04

Tipo: Prueba de Funcionalidad - Positiva.

Nombre: Verificar que una línea roja muestra la ruta óptima hacia el lugar.

Descripción: Validar que el usuario puede ver una línea roja mostrando la ruta óptima entre la posición del usuario y el lugar.

Condiciones de Ejecución: Registrar un lugar dentro del campus Universitario.

Entradas / Pasos de Ejecución:

1. Seleccionar el menú *Lugares*.
2. Buscar el lugar registrado.
3. Seleccionar la opción para buscar la ruta al lugar.

Resultado Esperado: Una vista del mapa del campus Universitario debe ser desplegado con una línea roja uniendo el marcador sobre la posición del usuario y la posición del lugar.

Evaluación de la Prueba: Prueba exitosa.

Tabla 5.36: Prueba de Aceptación - CP009

Código: CP010 **Historia de Usuario:** US04

Tipo: Prueba de Funcionalidad - Negativa.

Nombre: Verificar que la ruta óptima se muestre dentro del campus Universitario.

Descripción: Validar que la ruta óptima sea visible dentro del campus Universitario.

Condiciones de Ejecución: El usuario debe estar fuera del campus Universitario.

Entradas / Pasos de Ejecución:

1. Seleccionar el menú *Lugares*.
2. Buscar el lugar registrado.
3. Seleccionar la opción para buscar la ruta al lugar.

Resultado Esperado: La ruta óptima debe ser visible solamente dentro del campus Universitario.

Evaluación de la Prueba: Prueba exitosa.

Tabla 5.37: Prueba de Aceptación - CP010

Código: CP011

Historia de Usuario: US03

Tipo: Prueba de Usabilidad.

Nombre: Verificar que se puede cerrar el mapa.

Descripción: Validar que la vista del mapa pueda ser cerrada y la vista de la información del lugar sea visible nuevamente.

Condiciones de Ejecución: Un lugar debe estar registrado en el sistema.

Entradas / Pasos de Ejecución:

1. Seleccionar el menú *Lugares*.
2. Buscar el lugar registrado.
3. Seleccionar la opción para buscar la ruta al lugar.
4. Presionar el botón cerrar disponible sobre el mapa.

Resultado Esperado: La vista del mapa debe ser cerrada y la información del lugar ser visible.

Evaluación de la Prueba: Prueba exitosa.

Tabla 5.38: Prueba de Aceptación - CP011

5.4.4.1. Resultado de las pruebas de la Iteración 2

Al finalizar la Iteración 2, se ejecutaron todas las pruebas escritas durante la presente y la anterior iteración, en el cuadro 5.39 se puede ver el detalle.

Código	Título de la Prueba	Resultado
CP001	Verificar la lista de lugares.	Exitoso
CP002	Verificar la búsqueda de lugares.	Exitoso
CP003	Verificar la información de un lugar.	Exitoso
CP004	Verificar la lista de lugares cuando se busca un lugar no registrado.	Exitoso
CP005	Verificar la información de un lugar mediante el URI.	Exitoso
CP006	Verificar que el Menú sea desplegado dinámicamente.	Exitoso
CP007	Verificar el marcador del lugar sobre el mapa del campus Universitario.	Exitoso
CP008	Verificar que el marcador del lugar muestre la información.	Exitoso
CP009	Verificar que una línea roja muestra la ruta óptima hacia el lugar.	Exitoso
CP010	Verificar que la ruta óptima se muestre dentro del campus Universitario.	Exitoso
CP011	Verificar que se puede cerrar el mapa.	Exitoso

Tabla 5.39: Pruebas de regresión de la Iteración 2

- Se ejecutaron 6 pruebas de funcionalidad positiva, todas pasaron exitosamente.

- Se ejecutaron 3 pruebas de funcionalidad negativa, todas pasaron exitosamente.
- Se ejecutaron 2 prueba de usabilidad, todas pasaron exitosamente.

5.5. Iteración 3

5.5.1. Planificación

A continuación se analizará la *historia de usuario* US06, ver el cuadro 5.40.

Historia de Usuario: US06	Prioridad: Alta Riesgo: Alto
Nombre: Añadir más lugares al sistema.	
Descripción:	
Yo como usuario registrado. Deseo añadir más lugares al sistema. Para mejorar los criterios de búsqueda.	
Criterios de Aceptación:	
Quiero que sea posible anadir un lugar si no lo encuentro en la lista de lugares. Quiero ver un formulario para poder ingresar los datos de un nuevo lugar. Quiero pararme cerca o en el lugar que necesito añadir para georeferenciarlo.	

Tabla 5.40: Historia de Usuario - US06

Número de Tarea: T019	Historia de Usuario: US06
Descripción: Implementar en el backend el <i>endpoint</i> para añadir y editar usuarios.	
Tipo de Tarea: Desarrollo	Estimación [días]: 1
Programador Responsable: Edmundo Figueroa	

Tabla 5.41: Tarea de Ingeniería - T019

Número de Tarea: T020	Historia de Usuario: US06
Descripción: Mostrar un botón para añadir lugares solamente a usuarios registrados.	
Tipo de Tarea: Desarrollo	Estimación [dias]: 0.5
Programador Responsable: Edmundo Figueroa	

Tabla 5.42: Tarea de Ingeniería - T020

Número de Tarea: T021	Historia de Usuario: US06
Descripción: Mostrar un formulario para añadir más lugares, con el nombre del lugar, descripción, teléfono y nivel.	
Tipo de Tarea: Desarrollo	Estimación [dias]: 0.5
Programador Responsable: Edmundo Figueroa	

Tabla 5.43: Tarea de Ingeniería - T021

Número de Tarea: T022	Historia de Usuario: US06
Descripción: Registrar las coordenadas del lugar al momento de crear un nuevo lugar.	
Tipo de Tarea: Desarrollo	Estimación [dias]: 1
Programador Responsable: Edmundo Figueroa	

Tabla 5.44: Tarea de Ingeniería - T022

Número de Tarea: T023	Historia de Usuario: US06
Descripción: Implementar el modulo para poder agregar una foto a un lugar.	
Tipo de Tarea: Desarrollo	Estimación [dias]: 1
Programador Responsable: Edmundo Figueroa	

Tabla 5.45: Tarea de Ingeniería - T023

A continuación se analizará la *historia de usuario* US07, ver el cuadro 5.46.

Historia de Usuario: US07	Prioridad: Alta Riesgo: Alto
----------------------------------	---

Nombre: Editar la información de un lugar.

Descripción:

Yo como usuario registrado.

Deseo editar la información de un lugar.

Para mejorar o corregir la información de ese lugar.

Criterios de Aceptación:

Al entrar a la información de un lugar quiero ser el único que vea un ícono para poder entrar a la edición de los datos.

Quiero acceder a un formulario que muestre la información actual del lugar y poder editar la información mostrada.

Tabla 5.46: Historia de Usuario - US07

Número de Tarea: T024	Historia de Usuario: US07
Descripción: Implementar el modulo para editar lugares.	
Tipo de Tarea: Desarrollo	Estimación [dias]: 1
Programador Responsable: Edmundo Figueroa	

Tabla 5.47: Tarea de Ingeniería - T024

Número de Tarea: T025	Historia de Usuario: US07
Descripción: Mostrar un botón para editar un lugar solamente a usuarios administradores.	
Tipo de Tarea: Desarrollo	Estimación [dias]: 0.5
Programador Responsable: Edmundo Figueroa	

Tabla 5.48: Tarea de Ingeniería - T025

Número de Tarea: T026	Historia de Usuario: US07
Descripción: Mostrar la información actual del lugar en el formulario de edición.	
Tipo de Tarea: Desarrollo	Estimación [dias]: 0.5
Programador Responsable: Edmundo Figueroa	

Tabla 5.49: Tarea de Ingeniería - T026

Número de Tarea: T027

Historia de Usuario: US07

Descripción: Mostrar un botón para eliminar el lugar solamente a un usuario administrador.

Tipo de Tarea: Desarrollo

Estimación [días]: 1

Programador Responsable: Edmundo Figueroa

Tabla 5.50: Tarea de Ingeniería - T027

5.5.2. Diseño

- **Diagrama de Secuencia:**

En la figura 5.18, se observa el diagrama de secuencia correspondiente al registro de lugares.

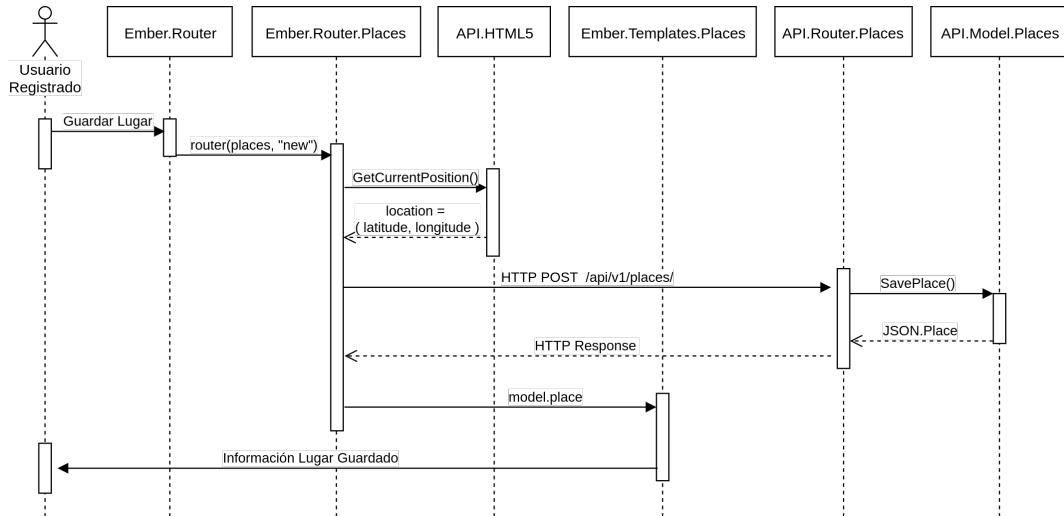


Figura 5.18: Diagrama de Secuencia: Guardar Lugar

Fuente: Elaboración propia

- **Diagrama de Clases:**

En la figura 5.19, se observa el diagrama de clases correspondiente al registro y edición de lugares.

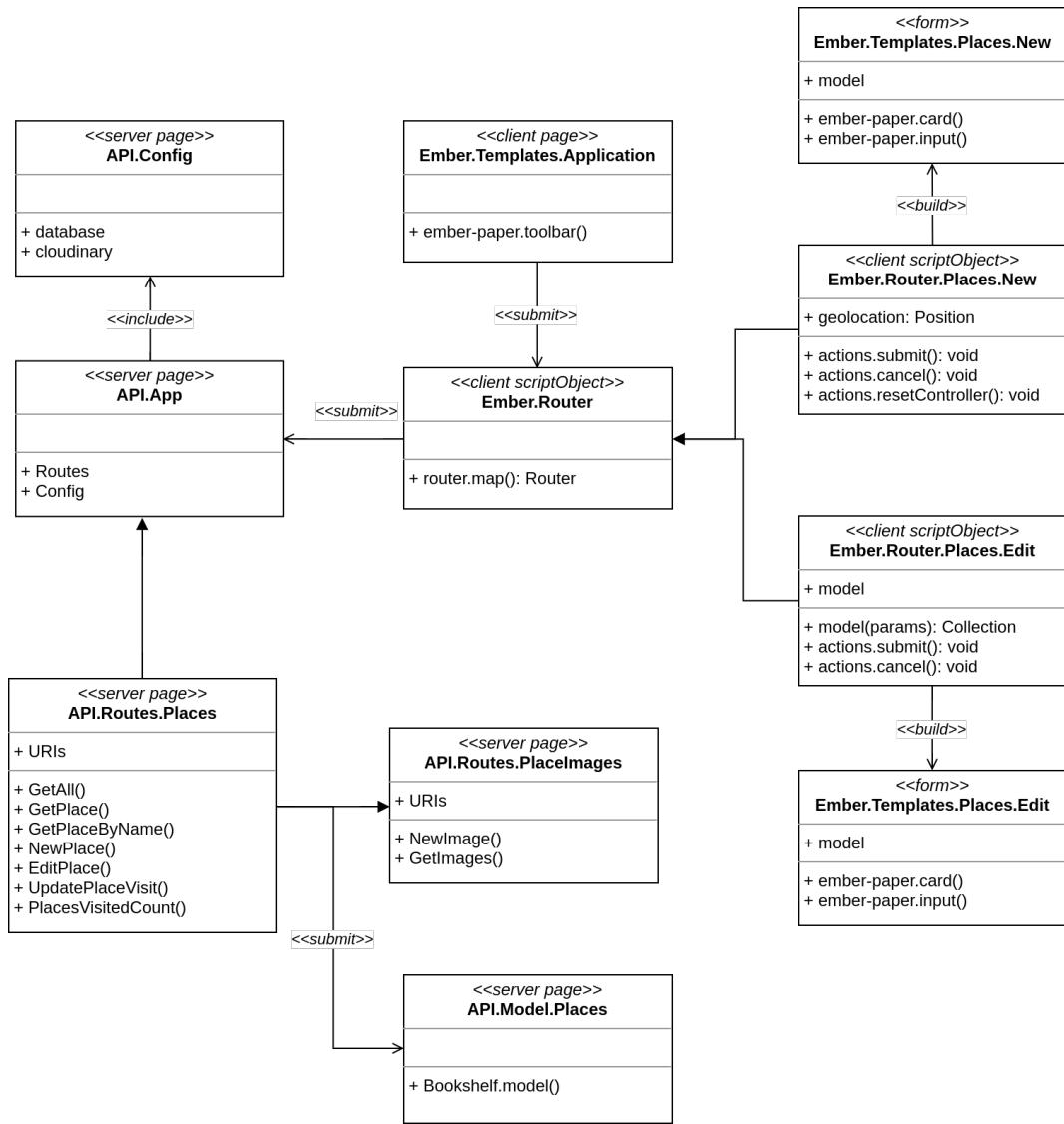


Figura 5.19: Diagrama de Clases: Guardar y Editar Lugar

Fuente: Elaboración propia

5.5.3. Implementación

5.5.3.1. Registro de Lugares en el sistema

Para registrar un *lugar* primeramente se implementó el formulario que se usará para recolectar la información del *lugar*, el formulario fue creado usando *EmberJs* y se lo puede ver en la figura 5.20.

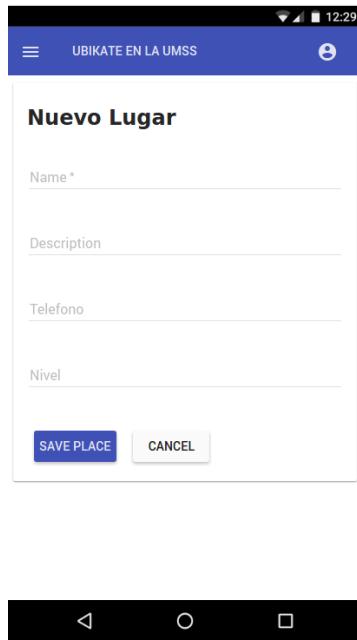


Figura 5.20: Formulario para añadir un nuevo *lugar*.

Fuente: Elaboración propia.

Posteriormente es necesario crear la petición HTTP desde el controlador de *EmberJS* hacia el API, en el cuerpo de la petición se incluye la posición georeferenciada del celular utilizando el API de Geolocalización de HTML5 usada anteriormente para encontrar la ubicación del usuario, también se incluye el nombre, la descripción, el teléfono y el nivel del *lugar*. En el código 5.10 se puede ver la implementación de la petición.

Code 5.10: POST request creado en el controlador de *ember*

```

var payload = {
    name: nombre,
    description: descripcion,
    phone: telefono,
    level: nivel,
    lat: latitud,
    lon: longitud
};

var url = (ENV.APP.API_HOST || '') + '/api/v1/places/';
jQuery.post(url, payload).then(
    function(data) {
        var transition = controller.get('transition');
        if (transition) {
            self.transitionTo('places.show');
        } else {
            self.transitionTo('places');
        }
    },

```

```

        function(error) {
            controller.set('message', error.responseText);
        }
    );

```

La petición creada es recibida en el API utilizando el *endpoint* RESTful designado para ello, `router.post('/', places.newPlace);`, el *endpoint* se encarga de mandar la información recibida hacia la base de datos, tomando en cuenta que la posición del lugar es un tipo especial de la base de datos, la latitud y longitud necesitan ser transformadas al momento de insertar el *lugar* en la base de datos, como se muestra en el código 5.11.

Code 5.11: Insertar un “lugar” en la base de datos.

```

router.post('/', places.newPlace);

var newPlace = (req, res) => {
    var name = req.body.name || '';
    var lat = req.body.lat || '';
    var lon = req.body.lon || '';
    var description = req.body.description || '';
    var phone = req.body.phone || '';
    var level = req.body.level || '';

    let raw = 'insert into place (name, geom, description, phone, level)
              values (' + ${name},
              ST_GeomFromText('POINT(${lon} ${lat})', 3875),
              ${description},
              ${phone},
              ${level}
            );';

    Knex.raw(raw)
        .then(function(data) {
            res.json({
                "message": "Place saved successfully!",
                "data": data
            });
        })
        .catch(function(error) {
            res.send("Error:", error);
        });
};

```

5.5.3.2. Edición de la información del Lugar

Para editar de la información de un lugar se rehusó el formulario ya creado para el registro del mismo, pero populando los campos con la información actual del lugar, el resultado se lo puede ver en la figura 5.21.



Figura 5.21: Formulario para editar un *lugar*

Fuente: Elaboración propia.

Los cambios a la información del *lugar* son enviados al API mediante una petición PUT, siguiendo los lineamientos del REST API implementado, esta petición se puede ver en el código 5.12.

Code 5.12: Petición HTTP para editar un lugar.

```
router.put('/:id', places.editPlace);

var body = {
  name: name,
  description: description,
  phone: phone,
  level: level
};

var url = (ENV.APP.API_HOST || '') + '/api/v1/places/${id}';
jQuery.ajax({
  url: url,
  type: 'put',
  data: body
}).then(
  function(data) {
    self.transitionTo('places.show', id);
  },
  function(error) {
    controller.set('message', error.responseText);
  }
);
```

A diferencia del registro del *lugar*, que guarda la posición georeferenciada al momento de registrar el *lugar*, en la edición de los datos no se está tomando en cuenta este dato porque se nota que generalmente cuando se requiere actualizar los datos del *lugar*, el usuario no se encuentra en la posición original del *lugar* y este dato se pierde, corrompiendo la base de datos. Es por esta razón que se decidió que no se actualizará la posición geográfica del *lugar* sin alguna opción explícita que advierta al usuario de las consecuencias de esta acción, pero como tal tarea no se halla dentro de la *historia de usuario* que se está implementando, se tomó la decisión de realizarla en una futura iteración.

5.5.4. Pruebas de Aceptación

Código: CP012

Historia de Usuario: US06

Tipo: Prueba de Funcionalidad - Positiva.

Nombre: Verificar el registro de lugares.

Descripción: Validar que un usuario registrado puede añadir un lugar nuevo al sistema.

Condiciones de Ejecución:

- a. El usuario debe estar registrado.

Entradas / Pasos de Ejecución:

1. Seleccionar el menú *Lugares*.
2. Hacer tap sobre el ícono de registro de lugares.
3. Ingresar el Nombre del lugar.
4. Ingresar una Descripción del lugar.
5. Ingresar el Teléfono del lugar.
6. Ingresar el Nivel del lugar.
7. Aceptar el formulario.

Resultado Esperado: Una vista de la información del lugar es desplegada con los datos ingresados en los pasos.

Evaluación de la Prueba: Prueba exitosa.

Tabla 5.51: Prueba de Aceptación - CP012

Código: CP013

Historia de Usuario: US07

Tipo: Prueba de Funcionalidad - Positiva.

Nombre: Verificar la edición de la información de un lugar.

Descripción: Validar que un usuario registrado puede editar la información de un lugar.

Condiciones de Ejecución:

- a. El usuario debe estar registrado.
- b. Registrar un lugar dentro del campus Universitario.

Entradas / Pasos de Ejecución:

1. Seleccionar el menú *Lugares*.
2. Buscar el lugar registrado en las condiciones.
3. Editar el Nombre del lugar.
4. Editar la Descripción del lugar.
5. Editar el Teléfono del lugar.
6. Editar el Nivel del lugar.
7. Aceptar el formulario.

Resultado Esperado: Una vista de la información del lugar es desplegada con la información editada en los pasos.

Evaluación de la Prueba: Prueba exitosa.

Tabla 5.52: Prueba de Aceptación - CP013

Código: CP014

Historia de Usuario: US06

Tipo: Prueba de Funcionalidad - Negativa.

Nombre: Verificar que no pueda registrar un lugar fuera del campus Universitario.

Descripción: Validar que los lugares que están fuera del campus Universitario no pueda ser registrado en el sistema.

Condiciones de Ejecución:

- a. El usuario debe estar registrado.
- b. El usuario debe estar fuera de los predios del Campus Universitario.

Entradas / Pasos de Ejecución:

1. Seleccionar el menú *Lugares*.
2. Hacer tap sobre el icono de registro de lugares.

Resultado Esperado: Un mensaje debería ser desplegado informando que no se pueden registrar Lugares fuera del Campus Universitario.

Evaluación de la Prueba: Prueba fallada.

Tabla 5.53: Prueba de Aceptación - CP014

Código: CP015

Historia de Usuario: US06, US07

Tipo: Prueba de Usabilidad.

Nombre: Verificar los campos de texto de los formularios de registro y edición.

Descripción: Validar que los campos de texto dentro los formularios de registro y edición de lugares no exceda la longitud de la pantalla del dispositivo móvil.

Condiciones de Ejecución:

- a. El usuario debe estar registrado.

Entradas / Pasos de Ejecución:

1. Seleccionar el menú *Lugares*.
2. Presionar el ícono de registro de lugares.
3. Insertar texto en los campos de descripción, teléfono y nombre.
4. Repetir para el formulario de edición de lugares.

Resultado Esperado: El texto escrito no debería sobrepasar la longitud del dispositivo móvil en los formularios de registro y edición de lugares.

Evaluación de la Prueba: Prueba exitosa.

Tabla 5.54: Prueba de Aceptación - CP015

5.5.4.1. Resultado de las pruebas de la Iteración 3

Al finalizar la Iteración 3, se ejecutaron todas las pruebas escritas durante la presente y las anteriores iteraciones, en el cuadro 5.55 se puede ver el detalle.

Código	Título de la Prueba	Resultado
CP001	Verificar la lista de lugares.	Exitoso
CP002	Verificar la búsqueda de lugares.	Exitoso
CP003	Verificar la información de un lugar.	Exitoso
CP004	Verificar la lista de lugares cuando se busca un lugar no registrado.	Exitoso
CP005	Verificar la información de un lugar mediante el URI.	Exitoso
CP006	Verificar que el <i>Menú</i> sea desplegado dinámicamente.	Exitoso
CP007	Verificar el marcador del lugar sobre el mapa del campus Universitario.	Exitoso
CP008	Verificar que el marcador del lugar muestre la información.	Exitoso
CP009	Verificar que una línea roja muestra la ruta óptima hacia el lugar.	Exitoso
CP010	Verificar que la ruta óptima se muestre dentro del campus Universitario.	Exitoso
CP011	Verificar que se puede cerrar el mapa.	Exitoso
CP012	Verificar el registro de lugares.	Exitoso
CP013	Verificar la edición de la información de un lugar.	Exitoso

CP014	Verificar que no pueda registrar un lugar fuera del campus Universitario.	Fallada
CP015	Verificar los campos de texto de los formularios de registro y edición.	Exitoso

Tabla 5.55: Pruebas de regresión de la Iteración 3

- Se ejecutaron 8 pruebas de funcionalidad positiva, todas pasaron exitosamente.
- Se ejecutaron 4 pruebas de funcionalidad negativa, falló la prueba CP014 pero que al ser una prueba negativa se lo documentara como un *problema conocido*.
- Se ejecutaron 3 prueba de usabilidad, todas pasaron exitosamente.

5.6. Iteración 4

5.6.1. Planificación

A continuación se analizará la *historia de usuario* US05, ver el cuadro 5.56.

Historia de Usuario: US05	Prioridad: Alta Riesgo: Alto
Nombre: Implementar el módulo de Registro de un Visitante.	
Descripción:	
Yo como visitante. Deseo registrarme. Para tener mas opciones dentro el sistema.	
Criterios de Aceptación:	
Quiero ver un formulario donde me pueda registrar. Una vez registrado quiero poder ingresar al sistema con mis credenciales. Quiero ver tener la posibilidad de editar mis datos.	

Tabla 5.56: Historia de Usuario - US05

Número de Tarea: T028	Historia de Usuario: US05
Descripción: Generar un link para acceder al formulario de registro.	
Tipo de Tarea: Desarrollo	Estimación [dias]: 0.5
Programador Responsable: Edmundo Figueroa	

Tabla 5.57: Tarea de Ingeniería - T028

Número de Tarea: T029	Historia de Usuario: US05
Descripción: Mostrar un formulario con el nombre, el email y la razon para registrarse.	
Tipo de Tarea: Desarrollo	Estimación [dias]: 0.5
Programador Responsable: Edmundo Figueroa	

Tabla 5.58: Tarea de Ingeniería - T029

Número de Tarea: T030	Historia de Usuario: US05
Descripción: Implementar un modulo de Registro de Usuarios.	
Tipo de Tarea: Desarrollo	Estimación [dias]: 2
Programador Responsable: Edmundo Figueroa	

Tabla 5.59: Tarea de Ingeniería - T030

A continuación se analizará la *historia de usuario* US08, ver el cuadro 5.60.

Historia de Usuario: US08	Prioridad: Alta Riesgo: Alto
Nombre: Implementar el módulo para Administrar Usuarios.	
Descripción:	
Yo como usuario administrador. Deseo administrar usuarios. Para añadir o remover usuarios del sistema.	
Criterios de Aceptación:	
Quiero ver los usuarios que desean registrarse en el sistema. Quiero aceptar o rechazar solicitudes de registro. Quiero eliminar usuarios que no usen el sistema de forma adecuada.	

Tabla 5.60: Historia de Usuario - US08

Número de Tarea: T031	Historia de Usuario: US08
Descripción: Mostrar el Menú de Registros solo para usuario Administrador.	
Tipo de Tarea: Desarrollo	Estimación [dias]: 0.5
Programador Responsable: Edmundo Figueroa	

Tabla 5.61: Tarea de Ingeniería - T031

Número de Tarea: T032	Historia de Usuario: US08
Descripción: Mostrar una lista con los Usuarios a registrar.	
Tipo de Tarea: Desarrollo	Estimación [dias]: 1
Programador Responsable: Edmundo Figueroa	

Tabla 5.62: Tarea de Ingeniería - T032

Número de Tarea: T033	Historia de Usuario: US08
Descripción: Mostrar un botón para aceptar del registro del usuario.	
Tipo de Tarea: Desarrollo	Estimación [dias]: 0.5
Programador Responsable: Edmundo Figueroa	

Tabla 5.63: Tarea de Ingeniería - T033

Número de Tarea: T034	Historia de Usuario: US08
Descripción: Mostrar un botón para Rechazar del registro del usuario.	
Tipo de Tarea: Desarrollo	Estimación [dias]: 0.5
Programador Responsable: Edmundo Figueroa	

Tabla 5.64: Tarea de Ingeniería - T034

Número de Tarea: T035	Historia de Usuario: US08
Descripción: Mostrar una lista con Usuarios Registrados.	
Tipo de Tarea: Desarrollo	Estimación [dias]: 0.5
Programador Responsable: Edmundo Figueroa	

Tabla 5.65: Tarea de Ingeniería - T035

Número de Tarea: T036	Historia de Usuario: US08
Descripción: Implementar un botón para eliminar Usuarios.	
Tipo de Tarea: Desarrollo	Estimación [dias]: 0.5
Programador Responsable: Edmundo Figueroa	

Tabla 5.66: Tarea de Ingeniería - T036

A continuación se analizará la *historia de usuario* US09, ver el cuadro 5.67.

Historia de Usuario: US09	Prioridad: Alta Riesgo: Alto
Nombre: Implementar el reporte de los lugares más visitados.	
Descripción:	
Yo como usuario administrador. Deseo ver los lugares más visitados. Para obtener información y estadísticas de los lugares dentro del campus Universitario.	
Criterios de Aceptación:	
Quiero ver la cantidad de veces que los usuarios buscan un lugar. Quiero poder guardar el reporte.	

Tabla 5.67: Historia de Usuario - US09

Número de Tarea: T037	Historia de Usuario: US09
Descripción: Implementar un módulo de Reportes.	
Tipo de Tarea: Desarrollo	Estimación [dias]: 1
Programador Responsable: Edmundo Figueroa	

Tabla 5.68: Tarea de Ingeniería - T037

Número de Tarea: T038

Historia de Usuario: US09

Descripción: Mostrar el menú de Reportes.

Tipo de Tarea: Desarrollo

Estimación [días]: 0.5

Programador Responsable: Edmundo Figueroa

Tabla 5.69: Tarea de Ingeniería - T038

Número de Tarea: T039

Historia de Usuario: US09

Descripción: Mostrar el Reporte de Frecuencia.

Tipo de Tarea: Desarrollo

Estimación [días]: 0.5

Programador Responsable: Edmundo Figueroa

Tabla 5.70: Tarea de Ingeniería - T039

5.6.2. Diseño

▪ Diagrama Entidad - Relación:

En la figura 5.22, se observa el diagrama Entidad - Relación de la aplicación.

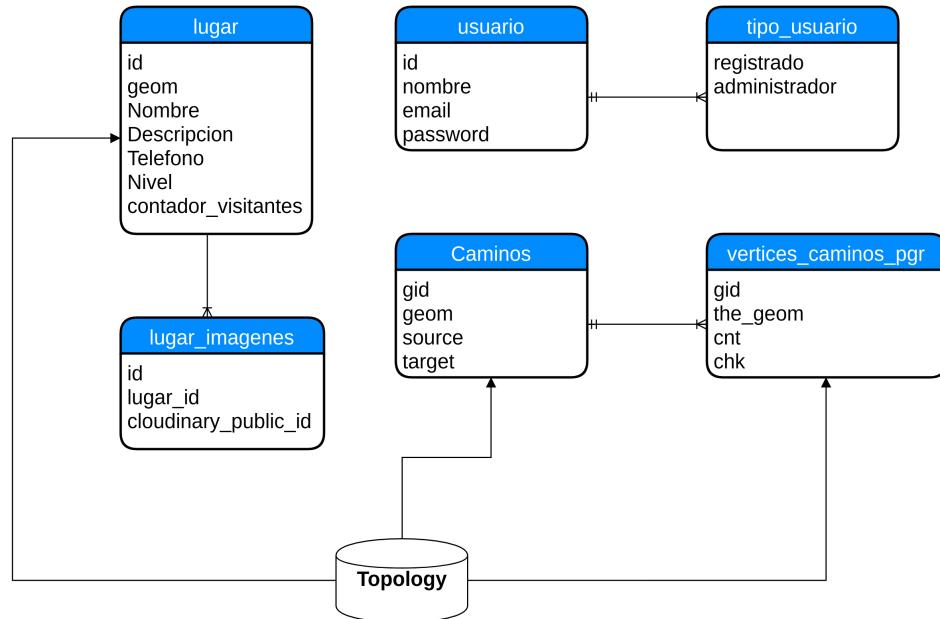


Figura 5.22: Diagrama ER: Ubikate UMSS

Fuente: Elaboración propia

■ **Diagrama de Secuencia:**

En la figura 5.23, se observa el diagrama de secuencia correspondiente al registro de un usuario.

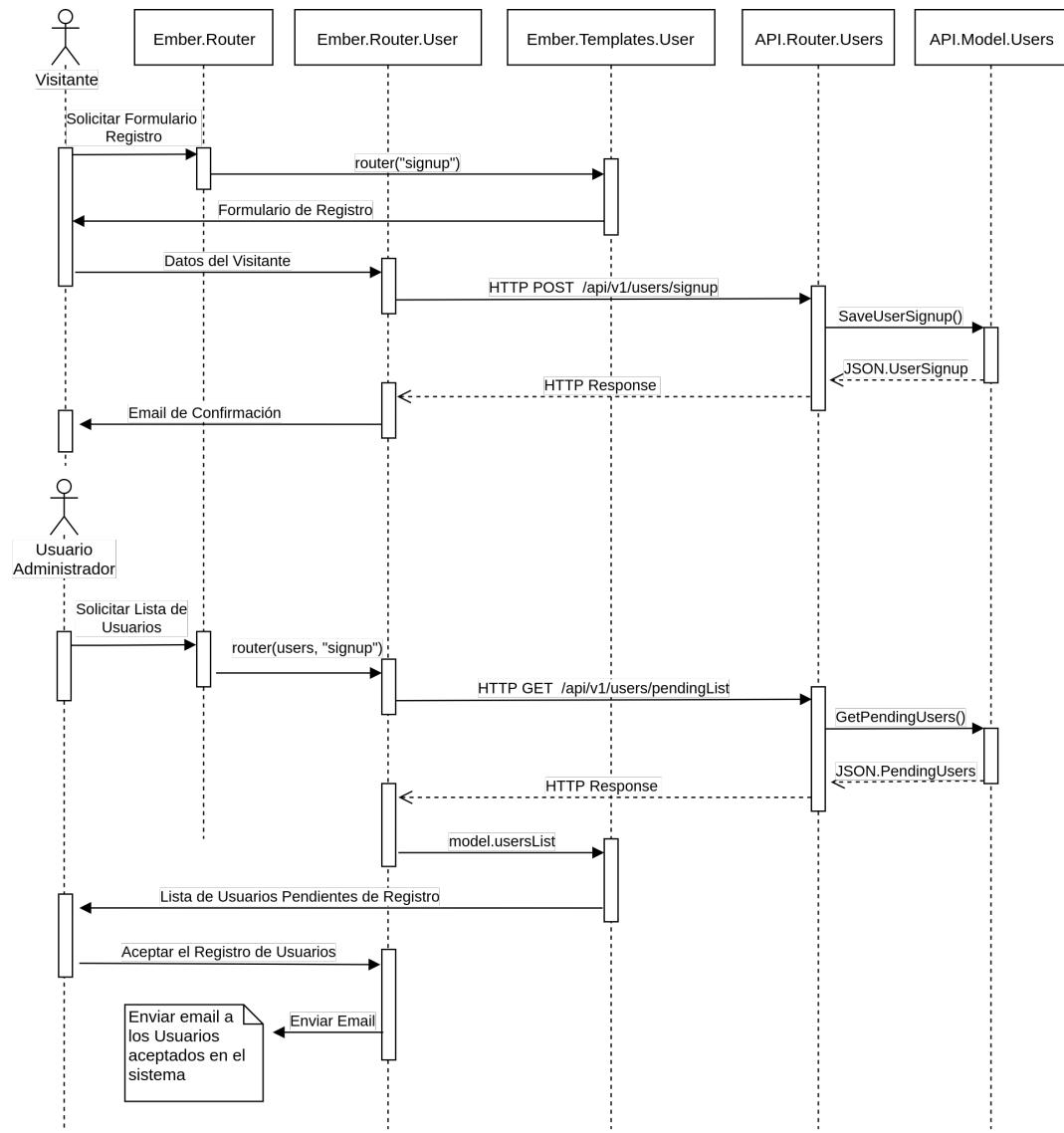


Figura 5.23: Diagrama de Secuencia: Registrar Usuarios

Fuente: Elaboración propia

En la figura 5.24, se observa el diagrama de secuencia correspondiente a la obtención del reporte.

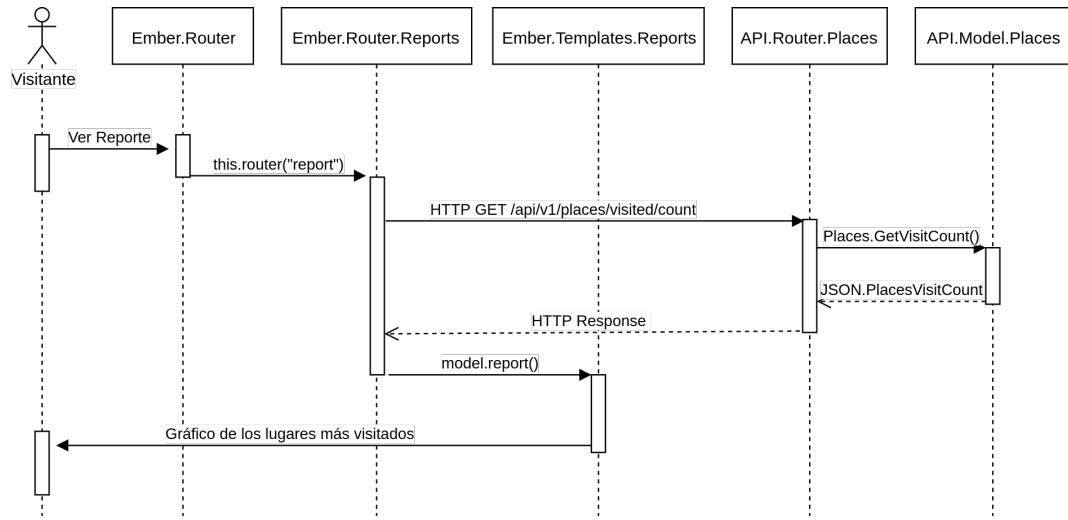


Figura 5.24: Diagrama de Secuencia: Reporte

Fuente: Elaboración propia

■ Diagrama de Clases:

En la figura 5.25, se observa el diagrama de clases correspondiente al registro de usuarios.

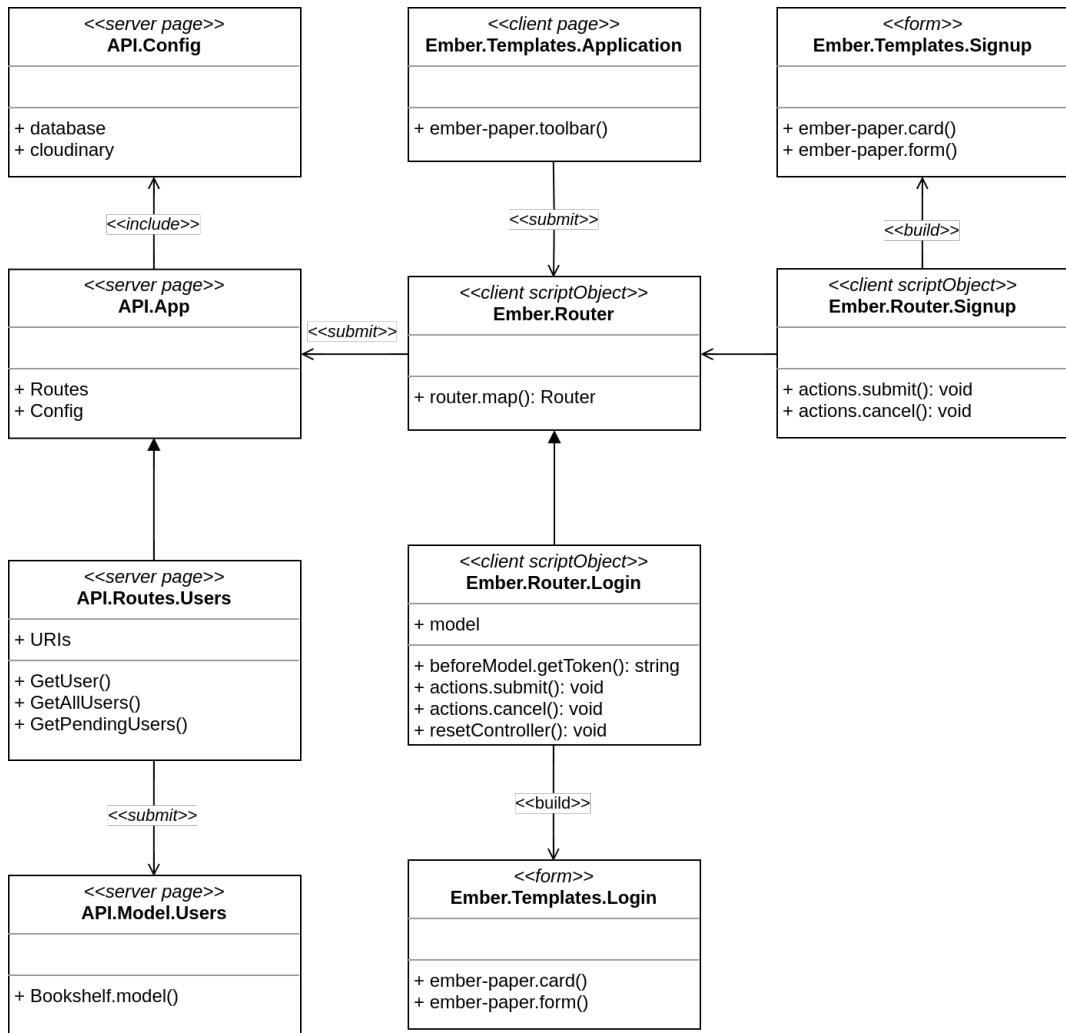


Figura 5.25: Diagrama de Clases: Registro de Usuario

Fuente: Elaboración propia

5.6.3. Implementación

5.6.3.1. Registro de Usuarios

El registro de Usuarios será mediante un formulario con el nombre, el email y la razón del registrante, y solamente el usuario administrador será capaz de aceptar o rechazar la solicitud de registro, para lo cual se implementó en la aplicación el *formulario de registro* y el *menú de registros*.

Formulario de Registro: Para completar esta tarea se implementó en la vista de la aplicación el formulario, ver la figura 5.26 usando *ember-paper*, que como ya se mencionó es la librería usada para implementar la vista de la aplicación dandole el *look and feel* de una aplicación móvil.

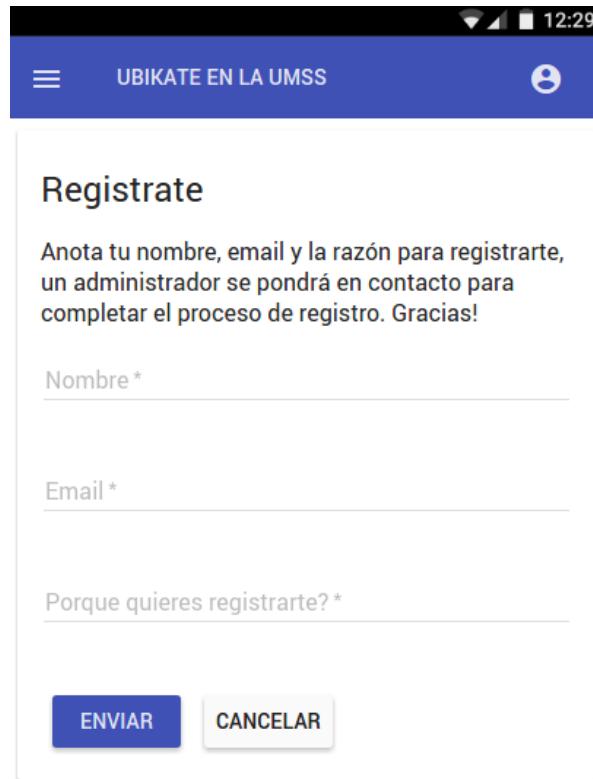


Figura 5.26: Formulario para Registro de Usuario

Fuente: Elaboración propia.

Posteriormente se implementó en el *backend* de la aplicación el módulo para guardar la solicitud de registro, para lo cual se añadió el *endpoint* en el API que escuche la petición POST generada al aceptar el *formulario de registro*, este *endpoint* inserta los datos colectados por el navegador en la base de datos para que estén disponibles posteriormente en el *Menú de Registros*.

Menú de Registros El *menú de registro* será donde un usuario administrador puede ver todas las solicitudes de registro, y de acuerdo de la *razón* de registro se puede determinar si la solicitud es de parte de un usuario responsable, en caso de necesitar más información del registrante se puede usar el *email* enviado, una vez que el administrador considera que el registrante no va a jugar en el sistema para asegurar la integridad de la misma, puede *aceptar* el registro, de esta forma el usuario recibirá un email donde se confirma el registro al sistema.

5.6.3.2. Generación del Reporte

Para la implementación del módulo que genere un reporte, se tuvo que modificar la tabla de los lugares, para saber cuántas veces un “lugar” es buscado, con esta información se puede proporcionar un reporte de frecuencia, para tal efecto se usó *ember-charts*, que como se puede observar en el código 5.13, sólo se necesita una línea para crear el reporte de frecuencia de la figura 5.27

Code 5.13: Componente de *Ember-charts*.

```
 {{horizontal-bar-chart data=model selectedSeedColor="green" sortAscending=false}}
```



Figura 5.27: Reporte de Frecuencia

Fuente: Elaboración propia.

Este reporte se obtiene al conocer la cantidad de veces que los usuarios han buscado un “lugar” mediante el SQL query, que se puede ver en el código 5.14 , el cual discrimina los primeros 10 lugares mas buscados dentro del campus universitario, ya que al existir tantas aulas y oficinas el reporte se haría muy extenso.

Code 5.14: Insertar un “lugar” en la base de datos.

```
router.get('places/visited/count', (req, res) => {
  Place.forge()
    .where('visit_count', '>', '1')
    .orderBy('visit_count', 'DESC')
    .query((qb) => qb.limit(10))
    .fetchAll({columns: ["name\u00d7as\u00d7label", "visit_count\u00d7as\u00d7value"]})
    .then((visited) => {
      res.json(visited.toJSON());
    })
    .catch((err) => {
      res.status(500);
    });
});
```

5.6.4. Pruebas de Aceptación

Código: CP016

Historia de Usuario: US005

Tipo: Prueba de Funcionalidad - Positiva.

Nombre: Verificar el Formulario de Registro.

Descripción: Validar que un usuario puede ver el formulario de registro.

Condiciones de Ejecución: El usuario no está registrado.

Entradas / Pasos de Ejecución:

1. Seleccionar el link *Registrarse*.
2. Ingresar el Nombre del usuario.
3. Ingresar el Email del usuario.
4. Ingresar una Razón de registro.
5. Aceptar el envío del formulario.

Resultado Esperado: El Usuario recibe un email de confirmación de Email.

Tabla 5.71: Caso de Prueba - CP016

Código: CP017	Historia de Usuario: US008
Tipo: Prueba de Funcionalidad - Positiva.	
Nombre: Verificar el Registro del Usuario.	
Descripción: La prueba verifica que un usuario se puede registrar en el sistema.	
Condiciones de Ejecución: El usuario ha ingresado datos válidos a la solicitud de registro.	
Entradas / Pasos de Ejecución:	
1. El Administrador ingresa al <i>Menú de Registros</i> . 2. Confirma la validez de los datos de registro del usuario. 3. Acepta el registro del usuario.	
Resultado Esperado: El usuario recibe un email de confirmación de registro.	

Tabla 5.72: Caso de Prueba - CP017

Código: CP018	Historia de Usuario: US008
Tipo: Prueba de Funcionalidad - Negativa.	
Nombre: Verificar que el usuario al registrarse no ingrese caracteres especiales.	
Descripción: La prueba verifica que un usuario al registrarse no pueda ingresar caracteres especiales en los campos del <i>formulario de registro</i> .	
Condiciones de Ejecución: El usuario no está registrado.	
Entradas / Pasos de Ejecución:	
1. Seleccionar el link <i>Registrarse</i> . 2. Ingresar en el campo <i>Nombre</i> caracteres especiales. 3. Repetir con el campo <i>Email</i> . 4. Repetir con el campo <i>Razón de registro</i> . 5. Aceptar el envío del formulario.	
Resultado Esperado: Un mensaje de información debe ser mostrado informando que los caracteres especiales no pueden ser ingresados en el formulario.	

Tabla 5.73: Caso de Prueba - CP018

Código: CP019	Historia de Usuario: US009
Tipo: Prueba de Funcionalidad - Positiva.	
Nombre: Verificar el Reporte de frecuencia.	
Descripción: La prueba verifica que el usuario puede ver el reporte de frecuencia.	
Condiciones de Ejecución: El usuario debe tener permisos de Administrador.	
Entradas / Pasos de Ejecución:	
1. El usuario ingresa al sistema con permisos de Administrador. 2. Selecciona al menú de <i>Reportes</i> .	
Resultado Esperado: El usuario puede ver la gráfica de barras con los lugares más visitados.	

Tabla 5.74: Caso de Prueba - CP019

5.6.4.1. Resultado de las pruebas de la Iteración 4

Al finalizar la Iteración 4 se ejecutaron todas las pruebas escritas durante la presente y las anteriores iteraciones, en el cuadro 5.75 se puede ver el detalle.

Código	Título de la Prueba	Resultado
CP001	Verificar la lista de lugares.	Exitoso
CP002	Verificar la búsqueda de lugares.	Exitoso
CP003	Verificar la información de un lugar.	Exitoso
CP004	Verificar la lista de lugares cuando se busca un lugar no registrado.	Exitoso
CP005	Verificar la información de un lugar mediante el URI.	Exitoso
CP006	Verificar que el <i>Menú</i> sea desplegado dinámicamente.	Exitoso
CP007	Verificar el marcador del lugar sobre el mapa del campus Universitario.	Exitoso
CP008	Verificar que el marcador del lugar muestre la información.	Exitoso
CP009	Verificar que una línea roja muestra la ruta óptima hacia el lugar.	Exitoso
CP010	Verificar que la ruta óptima se muestre dentro del campus Universitario.	Exitoso
CP011	Verificar que se puede cerrar el mapa.	Exitoso
CP012	Verificar el registro de lugares.	Exitoso
CP013	Verificar la edición de la información de un lugar.	Exitoso
CP014	Verificar que no pueda registrar un lugar fuera del campus Universitario.	Fallada
CP015	Verificar los campos de texto de los formularios de registro y edición.	Exitoso
CP016	Verificar el Formulario de Registro.	Exitoso
CP017	Verificar el Registro del Usuario.	Exitoso

CP018	Verificar que el usuario al registrarse no ingrese caracteres especiales.	Exitoso
CP019	Verificar el Reporte de frecuencia.	Exitoso

Tabla 5.75: Pruebas de regresión de la Iteración 4

- Se ejecutaron 11 pruebas de funcionalidad positiva, todas pasaron exitosamente.
- Se ejecutaron 5 pruebas de funcionalidad negativa, la prueba CP014 falló pero está documentada como un *problema conocido*.
- Se ejecutaron 3 prueba de usabilidad, todas pasaron exitosamente.

5.7. Producción

En esta fase se llevaron a cabo las pruebas de carga donde se simulará la interacción de los usuarios con la aplicación para validar el rendimiento de la aplicación cuando la aplicación está disponible para su uso.

Para tal efecto se probaron 3 *endpoints* o URIs del API. Que se escogieron por considerarse que son las que serán altamente usadas y posiblemente las más críticas, se las puede ver en la tabla 5.76.

Código	Objetivo	Endpoint
PR01	Obtener la lista de Lugares.	GET api/v1/places/
PR02	Obtener el <i>Nodo</i> (del mapa de rutas) más cercano a la posición (latitud y longitud) del Usuario o del Lugar.	GET api/v1/ways/node/:lat/:lon
PR03	Obtener la ruta óptima entre 2 <i>Nodos</i> , el nodo origen (<i>source</i>) y el nodo destino (<i>target</i>) .	GET api/v1/ways/route/:target/:source

Tabla 5.76: Pruebas de rendimiento

Se consideró que la aplicación debería soportar por lo menos la interacción de 100 usuarios, por lo que se tomó esta cantidad como caso base y sobre la cual se aumentará a 500 usuarios y por último lugar, 1000 usuarios interactuando con la aplicación al mismo tiempo.

Para llevar a cabo estas pruebas se hizo uso de *JMeter* y a continuación se describirán los resultados encontrados.

En la figura 5.28, se puede observar que los *requests* de 100 usuarios hacia la aplicación tiene tiempos de retorno buenos, para la prueba PR02 y PR03 se obtuvo un promedio de 0.8 segundos en cambio para la prueba PR01 el tiempo de ejecución se incrementó hasta un promedio de 8 segundos pero en ninguna de las 3 pruebas se generó algún error de respuesta del servidor. En cambio cuando 500 usuarios interactúan con la aplicación se puede observar

que obtener la lista de lugares ya presenta errores y los tiempos de retorno aumentaron considerablemente, la misma tendencia se puede ver con la ejecución de 1000 usuarios.

The figure consists of three vertically stacked screenshots of the JMeter 'Summary Report' page. Each screenshot shows performance metrics for three different requests: 'Lista lugares', 'get node', and 'ruta optima'. The first row contains the header for the table, and the second row contains the data for 100 users. The third row contains the data for 500 users. The fourth row contains the data for 1000 users. The columns include: Label, # Samples, Average, Min, Max, Std. Dev., Error %, Throughput, Received KB/s..., Sent KB/sec, and Avg. Bytes.

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/s...	Sent KB/sec	Avg. Bytes
Lista lugares	100	8805	1352	19897	2888.82	0.00%	4.9/sec	166.02	0.79	34629.0
get node	100	825	190	5460	823.48	0.00%	5.2/sec	1.98	0.95	392.0
ruta optima	100	868	195	9169	1092.60	0.00%	5.2/sec	22.37	0.88	4439.0
TOTAL	300	3500	190	19897	4181.15	0.00%	14.3/sec	184.01	2.45	13153.3
Lista lugares	500	52846	5627	128032	38983.29	19.20%	3.9/sec	108.16	0.51	28376.1
get node	500	1500	365	9331	1514.12	0.00%	3.9/sec	1.51	0.72	392.0
ruta optima	500	1868	194	35160	2961.92	0.00%	4.0/sec	17.24	0.68	4439.0
TOTAL	1500	18738	194	128032	33044.89	6.40%	11.2/sec	120.99	1.80	11069.0
Lista lugares	1000	83322	4175	128140	42045.51	44.80%	7.8/sec	152.45	0.69	20039.0
get node	1000	13264	364	127877	29151.60	4.80%	4.0/sec	1.84	0.70	472.2
ruta optima	1000	1832	194	40759	3076.21	0.00%	4.0/sec	17.33	0.68	4439.0
TOTAL	3000	32806	194	128140	46619.85	16.53%	11.6/sec	94.13	1.67	8316.7

Figura 5.28: Resultados obtenidos de JMeter.

Fuente: Elaboración propia.

En la figura 5.29, se puede ver el consumo de recursos por parte del servidor durante la ejecución de las pruebas de carga, se puede notar que a pesar de haber generado errores cuando la carga de usuarios es elevada, el servidor regresa a un estado estable.

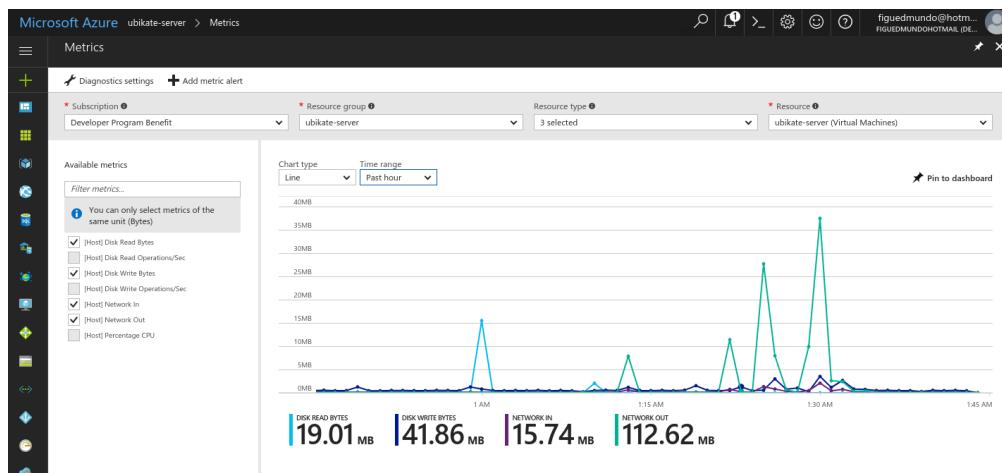


Figura 5.29: Reporte del servidor Azure.

Fuente: Elaboración propia.

De la ejecución de las pruebas de carga se puede concluir que es necesario analizar y refactorizar los módulos encargados de *obtener la lista de usuarios*, tarea que se puede realizar en una futura entrega del producto.

Tomando en cuenta que para una carga de 100 usuarios se observaron tiempos de respuesta aceptables por el cliente, se considera que la aplicación cumple con los objetivos de rendimiento trazados.

Capítulo 6

Conclusiones y Recomendaciones

6.1. Conclusiones

En relación al objetivo de “Generar un mapa con información geográfica de las rutas dentro del campus Universitario” se concluye que, es de gran importancia conocer y comprender los distintos tipos de *sistema de coordenadas*, por ejemplo al principio se intentó trabajar con el *sistema de coordenadas geográfico* ya que trabaja con los términos “latitud” y “longitud” que son conceptos más comúnmente utilizados y es de alguna forma más fácil empezar a usar este sistema de coordenadas, pero para encontrar la distancia entre puntos se notó que el cálculo es realizado sobre *unidades angulares* o *grados*, por lo que, para el cálculo del tamaño de una línea recta se necesita tomar en cuenta la forma esférica de la tierra y esto se traduce en muchos más cálculos matemáticos, que son fácilmente solventados usando un *sistema de coordenadas geométrico* porque trabaja con mapas proyectados o aplanados y el cálculo de distancias es realizado sobre líneas rectas y en *metros*, por lo que finalmente se terminó escogiendo este sistema de coordenadas, más específicamente la proyección *SRID 3857* o proyección *Mercator*.

En relación al objetivo “Administrar lugares geolocalizados dentro del campus Universitario” se observó que al registrar lugares dentro la base de datos es necesario prestar atención a la georeferenciación del *lugar*, ya que, el API de Geolocalización de HTML5 ofrece la posición del dispositivo en *latitud* y *longitud* (datos geográficos) es fácil corromper la base de datos si es que no se proyectan estos datos al momento de registrar un *lugar*, en este caso hay que recalcar el beneficio de trabajar con PostGIS, el cual implementa numerosas funciones para manipular datos geoespaciales.

En relación al objetivo “Mostrar en la aplicación los lugares geolocalizados desplegando la ruta óptima desde mi posición hasta el punto destino” se concluye que existen numerosas soluciones para encontrar la ruta óptima, donde se toman en cuenta diferentes variables y heurísticas, y su uso depende del problema pero para el presente proyecto se decidió usar el algoritmo de *Dijkstra* que aunque requiere muchos más cálculos que las demás soluciones es la solución más sencilla, ya que, tomando en cuenta el tamaño del grafo resultante, no existió una razón de gran peso para implementar otra solución más eficiente en el manejo de recursos.

En relación al objetivo de “Administrar usuarios en el sistema”, se concluye que al tratarse de información difícil de crear, es decir la información de los *lugares* y su posición georeferenciada, es necesario un registro y seguimiento personalizado de los usuarios por parte del administrador para detener el uso indebido o malintencionado de los usuarios registrados y de alguna forma asegurar que la información almacenada en la base de datos no sea corrompida y sea confiable.

En relación al objetivo de “Registrar las búsquedas sobre rutas realizadas por los usuarios en el sistema”, se concluye que guardar un registro de la cantidad de veces que los usuarios buscan algún *lugar* podría ayudar a la Universidad en la toma de decisiones en relación a los *lugares* dentro del campus Universitario, por ejemplo mejorar la infraestructura o la señalización de los lugares más visitados.

Al desarrollar la aplicación presentada en el proyecto de grado se llegó a la conclusión de que aunque es indudable que una aplicación móvil nativa tiene muchas más prestaciones que una aplicación web, también es cierto que existe una mayor fragmentación de dispositivos móviles y por lo tanto de tecnologías, por lo que con la implementación de una aplicación web móvil se puede llegar a muchas más personas agilizando en gran medida la captación de usuarios para la aplicación.

6.2. Recomendaciones

Para finalizar el presente proyecto de grado, se recomienda:

- Que para un futuro desarrollo de la aplicación se podría tomar en cuenta las facultades externas al campus Universitario “Las Cuadras”.
- También se podría aumentar la característica de que la aplicación pueda mostrar los puntos de interés cercanos a la ubicación del usuario.
- Se podría incluir códigos QR en los lugares de interés dentro de la Universidad para que la aplicación leyendo estos códigos pueda ofrecer información acerca del lugar.
- En el manejo de usuarios se recomienda un cuidado especial a la hora de registrar usuarios en el sistema, ya que siempre existen usuarios maliciosos que podrían dañar o corromper la información recolectada.
- Para una futura implementación de una aplicación web usando tecnologías basadas en JavaScript, como se *ExpressJS* o *EmberJS*, se recomienda un análisis cualificativo de los beneficios que se pueden encontrar en los diferentes *frameworks* de desarrollo, tomando en cuenta las características de la aplicación.

Bibliografía

- Abernethy, M. (2011). *¿simplemente qué es node.js?* <https://www.ibm.com/developerworks/ssaopensource/library/os-nodejs/index.html>. (Visitado Septiembre - 2016)
- Azuma, D. (2011). *Coordinate systems and projections.* <http://daniel-azuma.com/articles/georails/part-4>. (Visitado Septiembre - 2016)
- Barrero, A. C., de Garcia, G. W., y Parra, R. M. M. (2010). *Introducción a la teoría de grafos.* (ISBN: 978-958-99325-7-5)
- Bascopé, R. T. (2017). *Control de calidad: Test cases.* http://www.cs.umss.edu.bo/doc/material/mat_gral_130/Presentacion%20de%20Test%20Cases%201.7.pptx. (Visitado Julio - 2017)
- Beck, K. (1999). *Extreme programming explained: Embrace change.* Addison Wesley.
- Blake, L. (2007, Septiembre). Spatial relationships in gis – geospatial topology basics. *OSGeo Journal*, 2. (ISSN 1994-1897)
- Burbeck, S. (1992). *How to use model-view-controller (mvc)* (Inf. Téc.).
- Chan, G. (2015, Mayo). *Ember.js: An antidote to your hype fatigue.* <http://brewhouse.io/blog/2015/05/13/emberjs-an-antidote-to-your-hype-fatigue.html>.
- Dudziak, T. (2000). *extreme programming an overview* (Inf. Téc.). SEIDENBERG SCHOOL OF CSIS. http://csis.pace.edu/~marchese/CS616/Agile/XP/XP_Overview.pdf.
- Escudero, X., y Sosa, G. M. (2017). Glosario estándar de términos utilizados en pruebas software. *Spanish Software Testing Qualifications Board*. (Versión: 2.4.ES V001 E009)
- Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures* (Tesis Doctoral no publicada). UNIVERSITY OF CALIFORNIA, IRVINE.
- Hahn, E. (2014). *Understanding express.js.* <https://evanhahn.com/understanding-express/>. (Visitado Septiembre - 2016)
- Herring, J. R. (2011). Implementation standard for geographic information. *Open Geospatial Consortium Inc.* (OGC 06-103r4)
- Historia de la universidad mayor de san simón.* (2005). <http://www.umss.edu.bo/historia.php>. (Visitado Abril - 2017)

Hoffer, J. A., George, J. F., y Valacich, J. S. (2005). *Modern systems analysis and design*. Prentice-Hall.

Hutagalung, W., y Sauter, D. V. (2006). *Extreme programming*. <http://www.umsl.edu/~sauterv/analysis/f06Papers/Hutagalung/>. (Visitado Enero - 2017)

Knippers, R. (2009). *Coordinate systems*. <http://kartoweb.itc.nl/geometrics/coordinate%20systems/coordsys.html>. (Visitado Abril - 2017)

Lafon, Y. (2002). *Web services activity statement*. <https://www.w3.org/2002/ws/Activity>.

Mapa universitario. (2005). <http://www.umss.edu.bo/mapa.php>. (Visitado Abril - 2017)

Marset, R. N. (2007). *Rest vs web services* (Inf. Téc.). ELP-DSIC-UPV.

Offutt, J. (2002). Quality attributes of web software applications. *IEEE SOFTWARE*. (0740-7459/02)

Ponce, A. T. (2004, Marzo). *Xp metodología ágil*. https://prezi.com/axxujrg_tyxw/xp/.

Priolo, S. (2009). *Métodos ágiles*. (ISBN 978-987-1347-97-1)

Realidad aumentada georeferenciada. (2013).

<https://sites.google.com/a/xtec.cat/curs-qr-ar-public-copia/modul-2/2-realidad-aumentada-georeferenciada-espira>.

Single-page application. (2017). https://es.wikipedia.org/wiki/Single-page_application. (Visitado Abril - 2017)

Topf, J., y Hormann, C. (2015). *Projections/spatial reference systems*. <http://openstreetmapdata.com/info/projections>.

Uniform resource identifier (uri): Generic syntax. (2005, Enero). <http://www.ietf.org/rfc/rfc3986.txt>.

Végső, F. (2010). *The basics of positioning* (Inf. Téc.). University of West Hungary, Faculty of Geoinformatics.

WebSphere. (2012, Abril). *El desarrollo de aplicaciones móviles nativas, web o híbridas* (Inf. Téc. n.º WSW14182-USEN-01). IBM.

Wells, D. (2013, Octubre). *Extreme programming: A gentle introduction*. <http://www.extremeprogramming.org/>. (Visitado Abril - 2017)

What is gis? (2017).

<http://www.esri.com/what-is-gis>. (Visitado Abril - 2017)

Anexos

Anexo A

Manual Instalación

El Manual de instalación tomará en cuenta que todos los comandos y las herramientas son ejecutados sobre un servidor Linux Ubuntu 16.04 LTS.

A.1. Instalación de la base de datos

- **PostgreSQL:** Se usará la versión 9.4.8.

Code A.1: Comandos usados para instalar PostgreSQL.

```
$ sudo apt-get update  
$ sudo apt-get install -y postgresql postgresql-contrib
```

- **PostGIS:** Se usará la versión 2.1.

Code A.2: Comandos usados para instalar PostGIS.

```
$ sudo add-apt-repository ppa:ubuntugis/ubuntugis-unstable  
$ sudo apt-get install -y postgis postgresql-9.4-postgis-2.1
```

- **PgRouting:** Se usará la versión 2.2.

Code A.3: Comandos usados para instalar PgRouting.

```
$ sudo add-apt-repository ppa:georepublic/pgrouting-unstable  
$ sudo apt-get update  
$ sudo apt-get install postgresql-9.4-pgrouting
```

A.2. Configuración de la base de datos

- Habilitar *Admin pack* en PostgreSQL.

Code A.4: Habilitar Admin Pack.

```
$ sudo -u postgres psql
> CREATE EXTENSION adminpack;
> \q
```

- Crear el Usuario de la Base de Datos y crear la Base de Datos usando el usuario creado.

Code A.5: Comandos para crear el usuario y la base de datos.

```
$ sudo -u postgres createuser -d -E -i -l -P -r -s USER_NAME
$ sudo -u postgres createdb -O USER_NAME DATABASE_NAME
```

A.2.1. Añadir las funciones de PostGIS y pgRouting a la base de datos

PostGIS y pgRouting son extensiones a la base de datos PostgreSQL, y al instalar sus paquetes, todas las funciones que vienen con las extensiones están disponibles pero para utilizarlas hay que habilitarlas específicamente en la base de datos que las va a utilizar.

Ejecutar el siguiente comando para habilitar las funciones de PostGIS y pgRouting en la base de datos.

Code A.6: Habilitación de las Extensiones.

```
$ sudo -u postgres psql -c ''
CREATE EXTENSION adminpack; CREATE EXTENSION postgis;
CREATE EXTENSION postgis_topology;
CREATE EXTENSION pgrouting;
CREATE EXTENSION fuzzystrmatch;
CREATE EXTENSION postgis_tiger_geocoder;
'' DATABASE_NAME
```

- Se puede verificar que las extensiones instaladas están correctamente habilitadas en la base de datos.

Code A.7: Habilitación de las Extensiones.

```
$ psql -h localhost -U USER_NAME DATABASE_NAME
> SELECT postgis_full_version();
> SELECT * FROM pgr_version();
```

A.2.2. Habilitar el acceso a la base de datos

PostgreSQL por defecto viene con la configuración habilitada para conectarse a la base de datos usando conexiones seguras. Pero para el presente proyecto se debe ampliar los permisos de acceso a la base de datos para que el servicio REST API desarrollado no presenta problemas en la conexión a la base de datos.

- Editar el archivo *pg_hba.conf* ubicado en el directorio de instalación de PostgreSQL. La última sección del archivo debe quedar como en la siguiente tabla.

hostssl	all	all	0.0.0.0/0	md5
local	all	postgres		trust
local	all	all		trust
host	all	all	127.0.0.1/32	trust
host	all	all	::1/128	trust

- Reiniciar el servicio de PostgreSQL para que los cambios a la configuración tomen efecto.

Code A.8: Comando para reiniciar el servicio PostgreSQL.

```
$ sudo service postgresql restart
```

A.3. Configuration de la aplicación UBIKATE UMSS

Para el presente proyecto se utilizaron los siguientes datos para configurar la base de datos.

- USER_NAME: db_admin
- DATABASE_NAME: db_ubikate

A.3.1. Instalación de las dependencias

Instalar los siguientes paquetes para la correcta ejecución de la aplicación.

- **NodeJS:** Se usará la versión 6.2.2 o superior.

Code A.9: Comando para instalar NodeJS.

```
$ sudo apt-get install nodejs
```

- **Knex:** Instalar la última versión.

Code A.10: Commando para instalar Knex.

```
$ npm install knex -g
```

A.3.2. Copiar los archivos del proyecto UBIKATE UMSS

Los archivos de proyecto irán dentro de una carpeta, localizarla en alguna ubicación donde el usuario tenga permisos de lectura y escritura.

A.3.3. Configurar el mapa topológico del campus Universitario

El *shapefile* con las rutas están ubicadas dentro de la carpeta *resources* ubicada en la raíz del proyecto.

Para configurar la base de datos con las rutas se debe seguir los siguientes pasos:

1. Popular la tabla correspondiente a las rutas.

Code A.11: Comando popular la base de datos.

```
$ psql -d db_ubikate -U db_admin -f resources/ways.sql
```

2. Añadir las columnas *source* y *target* a la tabla *ways*, necesarias para agregar la topología.

Code A.12: Agregar las columnas *source* y *target*.

```
$ psql -h localhost -U db_admin db_ubikate
> alter table ways add column "source" integer;
> alter table ways add column "target" integer;
```

3. Crear la topología usando pgRouting.

Code A.13: Añadir la topología.

```
> select pgr_createTopology('ways', 0.00000001, 'geom', 'gid');
```

A.3.4. Instalar las dependencias del proyecto UBIKATE UMSS

Una vez los archivos hayan sido copiados, es necesario instalar todas las dependencias utilizadas por el proyecto.

Code A.14: Instalar dependencias de la aplicación.

```
$ npm install
```

A.3.5. Iniciar el servidor *ExpressJS*

Finalmente se inicia el servidor *ExpressJS* y la aplicación es mandada al cliente visitando la dirección del servidor.

Code A.15: Iniciar el servidor.

```
$ npm start
```


Anexo B

Manual Usuario

B.1. Página de Inicio

Es la primera vista, ver figura B.1, que se despliega cuando accedemos a la aplicación, se puede ver una descripción del sistema implementado, se pueden apreciar los siguientes elementos:

1. Icono del Menú.
2. Icono del Login o Registro del Usuario.
3. Botones de Registro de Usuario y Login respectivamente.

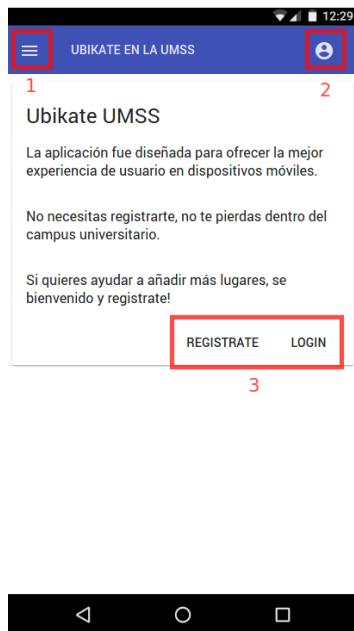


Figura B.1: Página de Inicio.

Fuente: Elaboración propia.

B.2. Menú de la Aplicación

Está ubicado en el sección izquierda de la pantalla, se accede a ella haciendo un tap sobre el *ícono del Menú*, ver la figura B.2. Una vez que se selecciona cualquier opción del menú, este se oculta. De esta forma se aprovecha el espacio reducido que dispone un dispositivo móvil.

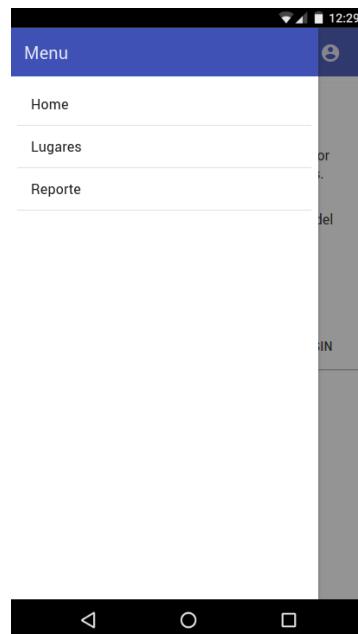


Figura B.2: Menú de la aplicación.

Fuente: Elaboración propia.

B.3. Lista de Lugares

Se accede a la *lista de lugares*, ver figura B.3, seleccionando la opción *Lugares* del Menú.

En la *lista de lugares* se puede ver los siguiente elementos:

1. Cajón de Búsqueda.
2. Lista de Lugares.
3. Botón de Registro de un Lugar (ver figura B.4).

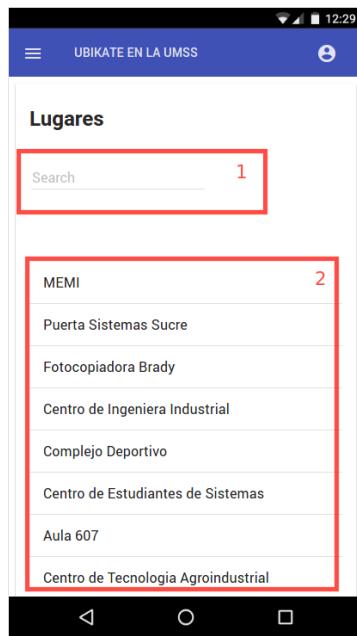


Figura B.3: Lista de lugares.

Fuente: Elaboración propia.

- **Cajón de Búsqueda:** La lista de lugares es filtrada de acuerdo al nombre de un lugar o parte del nombre, que es ingresada en el *cajón de búsqueda*. Como se puede apreciar en la figura B.4.

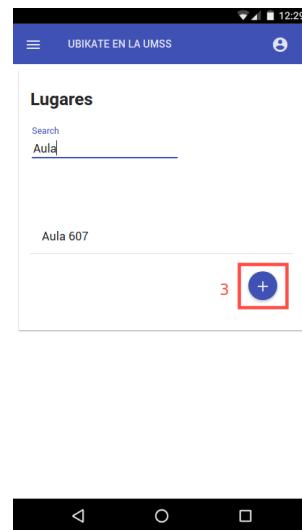


Figura B.4: Cajón de Búsqueda.

Fuente: Elaboración propia.

- **Lista de Lugares:** Es la lista de lugares propiamente dicha, contiene todos los lugares

registrados en la aplicación o la lista filtrada mediante el *cajón de búsqueda*.

- **Botón de Registro de un Lugar:** Ubicada al final de la *lista de lugares*.

B.4. Formulario de registro de un Lugar

El *formulario de registro de un lugar* contiene la siguiente información:

1. El *Nombre* del lugar, esta información es requerida.
2. La *Descripción* del lugar, en caso de que tuviera una, no es requerida.
3. El *Teléfono* del lugar, en caso de que tuviera una.
4. El *Nivel* o *Piso* de donde se encuentra lugar, si no es especifica un piso, el sistema deduce que el lugar está en el *nivel 0* o *Planta Baja*.

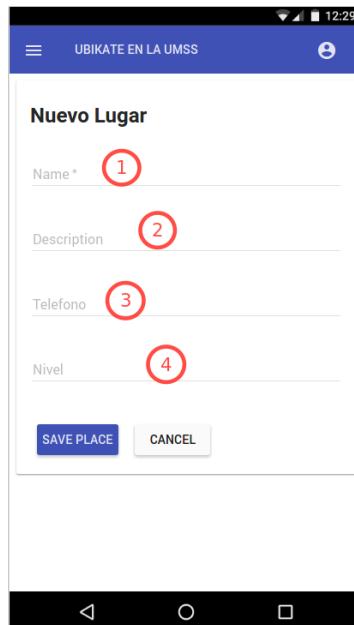


Figura B.5: Formulario de Registro de un Lugar.

Fuente: Elaboración propia.

B.5. Información del Lugar

La *información de un lugar*, ver la figura B.6, contiene los siguientes datos:

1. La *Foto* o imagen del lugar.
2. La *Información* del lugar, todos los datos ingresados en el *formulario de registro*.

3. El *Botón para agregar imágenes* del lugar, las imágenes agregadas serán desplegadas en la parte superior de la *información del lugar*.
4. El *Botón para encontrar la Ruta óptima al lugar*, al seleccionar esta opción se despliega el mapa del campus Universitario. Ver la figura B.7.
5. El *Botón para Editar la Información del lugar*, al seleccionar esta opción se despliega el *Formulario de edición de un Lugar*.



Figura B.6: Información de un Lugar.

Fuente: Elaboración propia.

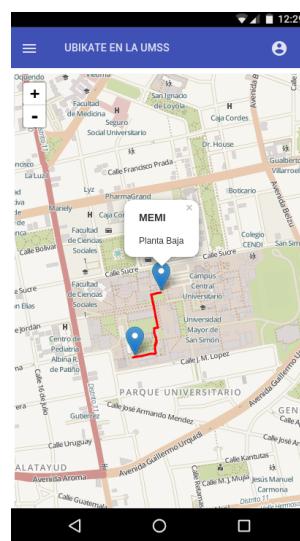


Figura B.7: Vista del camino o ruta óptima al lugar.

Fuente: Elaboración propia.

B.6. Formulario de edición de un Lugar

El *formulario de edición de un lugar* contiene la misma información que el *formulario de registro de un lugar*, pero en el formulario de edición está desplegada la información del lugar para ser edita, ver la figura B.8.



Figura B.8: Formulario de edición de un lugar.

Fuente: Elaboración propia.

B.7. Formulario de registro de un Usuario

Dentro del *formulario de registro de un usuario* todos los campos son necesario, en caso de no ingresar un dato, la aplicación muestra un mensaje indicando que el campo es requerido. Como se puede ver en la figura B.9, dentro de este formulario se pueden ver los siguientes campos:

1. El *Nombre* del usuario.
2. El *Email* del usuario.
3. La *Razón* o el *Porque* del registro.

The screenshot shows a mobile application interface for user registration. At the top, there is a header bar with the text "UBIKATE EN LA UMSS" and a battery icon showing 12:29. Below the header, the title "Registrate" is displayed, followed by a descriptive message: "Anota tu nombre, email y la razón para registrarte, un administrador se pondrá en contacto para completar el proceso de registro. Gracias!". The form contains three fields: "Nombre*" with the value "Luis Gomez" (circled in red as point 1), "Email*" with the placeholder "This is required." (circled in red as point 2), and a checkbox labeled "Quiero recorrer la Universidad y conocer a fondo todos las facultades." (circled in red as point 3). At the bottom are two buttons: "ENVIAR" and "CANCELAR".

Figura B.9: Formulario de registro de un Usuario.

Fuente: Elaboración propia.

B.8. Formulario de ingreso al sistema

El *formulario de ingreso al sistema* o *Login*, ver figura B.10, nos permite acceder al sistema con permisos de *Administrador* o de *Usuario Registrado*.

The screenshot shows a mobile application interface for user login. At the top, there is a header bar with the text "UBIKATE EN LA UMSS" and a battery icon showing 12:29. Below the header, the title "Log In" is displayed. The form contains two fields: "Username*" and "Password*". At the bottom are two buttons: a blue "INICIAR SESIÓN" button and a white "CANCEL" button.

Figura B.10: Formulario de ingreso al sistema.

Fuente: Elaboración propia.

B.9. Reporte

El *Reporte* mostrado en la figura B.11, muestra los lugares más visitados por los usuarios de la aplicación, para acceder a él es necesario seleccionar la opción *Reporte* del Menú.



Figura B.11: Lugares más visitados.

Fuente: Elaboración propia.

Índice de figuras

1.1.	Diagrama Árbol de Problemas	2
2.1.	Sistema de coordenadas Geocéntricas	7
2.2.	Sistema de coordenadas Geográficos	8
2.3.	Sistema de coordenadas Proyectadas	9
2.4.	Grafo ponderado no-dirigido	11
2.5.	Flujo de una Aplicación de Página Única.	14
2.6.	Arquitectura de una Aplicación Web Moderna	16
3.1.	Diagrama del proceso XP	29
4.1.	Mapa universitario	33
4.2.	Facultades dentro del Campus	34
4.3.	Facultad de Arquitectura - UMSS	35
4.4.	Facultad de Derecho - UMSS	37
4.5.	Facultad de Economía - UMSS	38
4.6.	Facultad de Humanidades - UMSS	40
4.7.	Facultad de Tecnología - UMSS	42
4.8.	Multiacademico - UMSS	44
4.9.	Complejo Deportivo - UMSS	46
5.1.	Calendario de Entregas	52
5.2.	Diagrama ER: Lugares	56
5.3.	Diagrama de Secuencia: Lista e Información de Lugares	56
5.4.	Diagrama de Clases: Lugares	57
5.5.	Herramienta gráfica de PostgreSQL (<i>pgAdmin</i>).	59
5.6.	Lista de Lugares	61

5.7.	Vista de la Información de un Lugar.	62
5.8.	Diagrama ER: Caminos	70
5.9.	Diagrama de Secuencia: Ruta Óptima	70
5.10.	Diagrama de Clases: Caminos	71
5.11.	Mapa mostrado con la ayuda de <i>ember-leaflet</i>	73
5.12.	Marcador con la información de un lugar.	74
5.13.	Marcador sobre la posición actual del usuario.	75
5.14.	Mapa de rutas del campus Universitario.	76
5.15.	Vista de la tabla <i>ways</i>	77
5.16.	Vista de la tabla <i>ways_vertices_pgr</i>	78
5.17.	Ruta más corta dibujada con una línea roja.	80
5.18.	Diagrama de Secuencia: Guardar Lugar	87
5.19.	Diagrama de Clases: Guardar y Editar Lugar	88
5.20.	Formulario para añadir un nuevo <i>lugar</i>	89
5.21.	Formulario para editar un <i>lugar</i>	91
5.22.	Diagrama ER: Ubikate UMSS	99
5.23.	Diagrama de Secuencia: Registrar Usuarios	100
5.24.	Diagrama de Secuencia: Reporte	101
5.25.	Diagrama de Clases: Registro de Usuario	102
5.26.	Formulario para Registro de Usuario	103
5.27.	Reporte de Frecuencia	104
5.28.	Resultados obtenidos de JMeter.	109
5.29.	Reporte del servidor Azure.	109
B.1.	Página de Inicio.	123
B.2.	Menú de la aplicación.	124
B.3.	Lista de lugares.	125
B.4.	Cajón de Búsqueda.	125
B.5.	Formulario de Registro de un Lugar.	126
B.6.	Información de un Lugar.	127
B.7.	Vista del camino o ruta óptima al lugar.	127
B.8.	Formulario de edición de un lugar.	128
B.9.	Formulario de registro de un Usuario.	129
B.10.	Formulario de ingreso al sistema.	129

B.11.Lugares más visitados.	130
-------------------------------------	-----

Índice de tablas

2.1. REST URIs para los lugares	19
4.1. Locaciones de la Fac. Arquitectura	36
4.2. Locaciones de la Fac. Derecho	37
4.3. Locaciones de la Fac. de Economía	39
4.4. Locaciones de la Fac. Humanidades	41
4.5. Locaciones de la Fac. Tecnología	44
4.6. Locaciones del Multiacadémico	45
4.7. Locaciones del Complejo Deportivo	46
5.1. Requerimientos Funcionales	50
5.2. Historias de Usuario	51
5.3. Estimación de las historias de usuario	51
5.4. Estimación de la implementación de las Historias de Usuario.	52
5.5. Historia de Usuario - US01	53
5.6. Tarea de Ingeniería - T001	53
5.7. Tarea de Ingeniería - T002	53
5.8. Tarea de Ingeniería - T003	54
5.9. Tarea de Ingeniería - T004	54
5.10. Tarea de Ingeniería - T005	54
5.11. Historia de Usuario - US02	54
5.12. Tarea de Ingeniería - T006	55
5.13. Tarea de Ingeniería - T007	55
5.14. Tarea de Ingeniería - T008	55
5.15. Tarea de Ingeniería - T009	55
5.16. Prueba de Aceptación - CP001	63

5.17. Prueba de Aceptación - CP002	63
5.18. Prueba de Aceptación - CP003	64
5.19. Prueba de Aceptación - CP004	64
5.20. Prueba de Aceptación - CP005	65
5.21. Prueba de Aceptación - CP006	65
5.22. Pruebas de regresión de la Iteración 1	66
5.23. Historia de Usuario - US03	66
5.24. Tarea de Ingeniería - T010	67
5.25. Tarea de Ingeniería - T011	67
5.26. Tarea de Ingeniería - T012	67
5.27. Tarea de Ingeniería - T013	67
5.28. Historia de Usuario - US04	68
5.29. Tarea de Ingeniería - T014	68
5.30. Tarea de Ingeniería - T015	68
5.31. Tarea de Ingeniería - T016	69
5.32. Tarea de Ingeniería - T017	69
5.33. Tarea de Ingeniería - T018	69
5.34. Prueba de Aceptación - CP007	81
5.35. Prueba de Aceptación - CP008	81
5.36. Prueba de Aceptación - CP009	82
5.37. Prueba de Aceptación - CP010	82
5.38. Prueba de Aceptación - CP011	83
5.39. Pruebas de regresión de la Iteración 2	83
5.40. Historia de Usuario - US06	84
5.41. Tarea de Ingeniería - T019	84
5.42. Tarea de Ingeniería - T020	85
5.43. Tarea de Ingeniería - T021	85
5.44. Tarea de Ingeniería - T022	85
5.45. Tarea de Ingeniería - T023	85
5.46. Historia de Usuario - US07	86
5.47. Tarea de Ingeniería - T024	86
5.48. Tarea de Ingeniería - T025	86
5.49. Tarea de Ingeniería - T026	86

5.50. Tarea de Ingeniería - T027	87
5.51. Prueba de Aceptación - CP012	92
5.52. Prueba de Aceptación - CP013	93
5.53. Prueba de Aceptación - CP014	93
5.54. Prueba de Aceptación - CP015	94
5.55. Pruebas de regresión de la Iteración 3	95
5.56. Historia de Usuario - US05	95
5.57. Tarea de Ingeniería - T028	95
5.58. Tarea de Ingeniería - T029	96
5.59. Tarea de Ingeniería - T030	96
5.60. Historia de Usuario - US08	96
5.61. Tarea de Ingeniería - T031	97
5.62. Tarea de Ingeniería - T032	97
5.63. Tarea de Ingeniería - T033	97
5.64. Tarea de Ingeniería - T034	97
5.65. Tarea de Ingeniería - T035	98
5.66. Tarea de Ingeniería - T036	98
5.67. Historia de Usuario - US09	98
5.68. Tarea de Ingeniería - T037	98
5.69. Tarea de Ingeniería - T038	99
5.70. Tarea de Ingeniería - T039	99
5.71. Caso de Prueba - CP016	105
5.72. Caso de Prueba - CP017	106
5.73. Caso de Prueba - CP018	106
5.74. Caso de Prueba - CP019	107
5.75. Pruebas de regresión de la Iteración 4	108
5.76. Pruebas de rendimiento	108