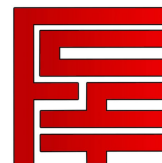


fancy

[page=1,level=-2]INICIO



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
ABRIL, 2017

8 de abril de 2017

Índice general

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:	4
1.6. Alcance	4
2. Marco Teorico	6
2.1. Aplicaciones Móviles	6
2.2. Single Page Application	7
2.3. REST API	10
2.4. Node JS	12
2.5. Express JS	12
2.6. Ember JS	13
2.7. Base de Datos	15
2.8. Geolocalización	17
2.9. Ruta Óptima dentro el campus Universitario	22
2.10. Metodologia de Desarrollo	25
2.11. Desarrollo del Proyecto usando XP	29
3. Caso Estudio: Campus Universitario	34
3.1. Generar el mapa con información geográfica	34
3.2. Backend	42
3.3. Crear el Frontend	51
3.4. Informe de los datos	55

4. Planning Game Ubikate UMSS	56
4.1. Historias de Usuario	56
4.2. Planeacion de Entregas	59
5. Iteration Planning Game Ubikate UMSS	60
5.1. Iteración 1	60
5.2. Iteración 2	63
5.3. Iteración 3	70
Bibliografía	73

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 m² 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 cuenta 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, de diferentes formas, por ejemplo si usamos movilidad, bicicleta, a pie, para lograr esto se toman en cuenta los distintos tipos de calles que existen y la dirección en el caso de movildades, Google Maps toma en cuenta la descripción de una locación o la referencia cartográfica en latitud y longitud de los puntos, y el cómo nos vamos 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, fotocopadoras, oficinas, etc. Esta información es inexistente o de difícil acceso lo cual genera malestar cuando se está buscando 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, este mapa sería de gran ayuda para desplazarse dentro del campus universitario, la falta de un mapa con estas características genera malestar entre la estudiantes o personas que quieren realizar trámites administrativos, ya que al no contar con una aplicación que muestre los puntos de interés geolocalizados se pierde tiempo al tratar de encontrarlos. En la figura 1.1, se puede apreciar al árbol de problemas.

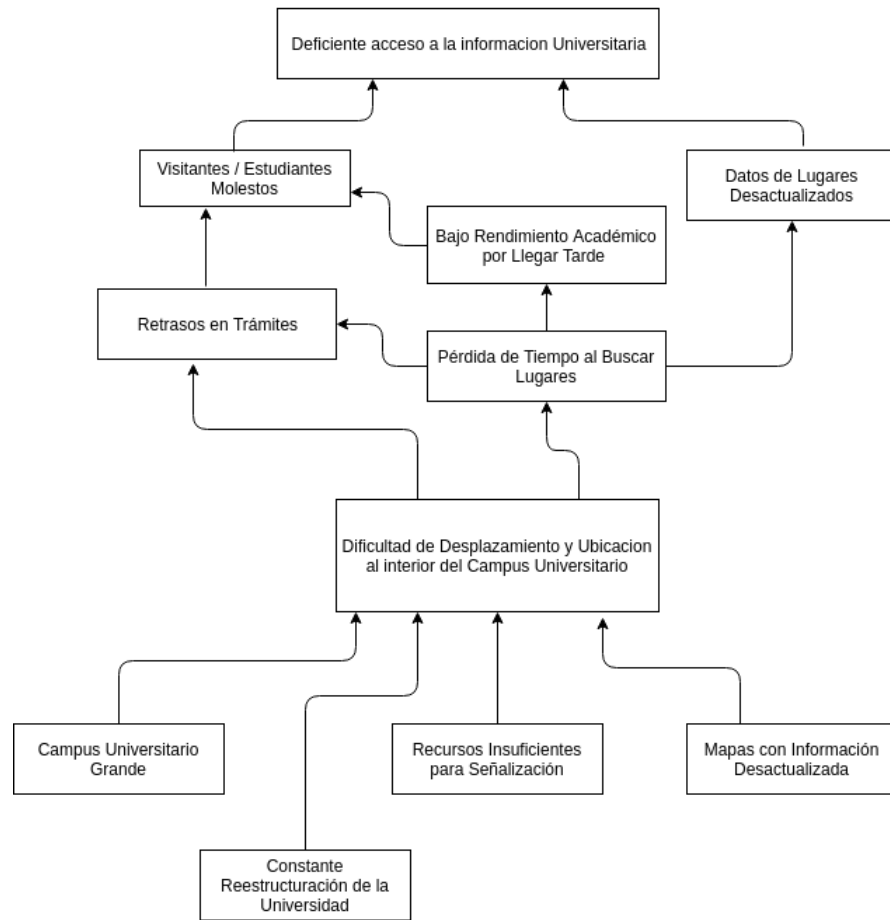


Figura 1.1: Diagrama Árbol de Problemas

Fuente: Elaboración propia

1.3. Objetivo general

Desarrollar una aplicación web móvil responsive para optimizar la ubicación 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.
- Gestionar 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 está en constante reestructuración, cada vez hay más aulas, las oficinas se mueven de lugar, etc. gracias a esto es que los mapas, que son escasos y están impresos sobre banners estáticos, son 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 trámites administrativos no encuentran con facilidad las oficinas a las que necesitan llegar.

Una aplicación que permita ubicar y encontrar locaciones además de proveer la ruta óptima dentro del campus de la Universidad Mayor de San Simón es de gran importancia para mejorar nuestra presentación 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 nosotros necesitamos apuntar a esta tendencia.

1.6. Alcance

1.6.1. Alcance Práctico

Una aplicación web móvil puede llegar a ser muy compleja, por ejemplo la seguridad de los datos de los usuario, ya que el API está expuesto en un servidor de acceso público y es susceptible de ataques maliciosos y malintencionados para lograr 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, como ser tarjetas de crédito pero la aplicación manejará información relacionada con la locación del usuario y esto podría ser usado para fines delictivos. Para una aplicación web la seguridad es muy importante, el presente proyecto implementara medidas de seguridad para asegurar la identidad del usuario que se está solicitando el ingreso (logueado) 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 Teorico

La aplicación a desarrollar estará enfocado a su uso en un celular inteligente (smartphone) por lo que hay que determinar el enfoque de desarrollo que se usará y las herramientas necesarias para construir esta aplicación.

2.1. Aplicaciones Móviles

El desarrollo de aplicaciones web se divide en 3 grupos de enfoques de desarrollo.

2.1.1. 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.

2.1.2. 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.

2.1.3. 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[4].

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[4].

Para la aplicación se escogió un desarrollo enfocado a tecnología Web diseñado para su uso en smartphones, o una aplicación web responsive. Para lograr este objetivo se usará, tecnologías aplicadas ampliamente en el desarrollo de aplicaciones web. Para implementar el backend de la aplicación se usará *NodeJS* con *ExpressJS*, la base de datos se construirá sobre *PostgreSQL* y *PostGIS* más *pgRouting*, estos complementos de PostgreSQL nos ayudarán a manejar los datos geoespaciales, para el desarrollo del frontend se usará *EmberJS* y para manejar las imágenes en la web *Cloudinary*.

2.2. Single Page Application

Un single-page application (SPA), o aplicación de página única es una aplicación web o es un sitio web que cabe en una sola página con el propósito de dar una experiencia más fluida a

los usuarios como una aplicación de escritorio. [20]

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 datos, generalmente en formato JSON, de esta forma actualiza porciones de la página con información nueva del servidor, sin recargar la pagina gracias a JavaScript. 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 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. Es básicamente una implementación del Patrón de Diseño *MVC*.

2.2.1. 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.[16]

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*.

Esta separación hace que diseñar cada capa de la aplicación como un ente distinto, sea sencillo. 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 viceversa.

La arquitectura de un SPA, como se puede ver en la figura 2.1, el *DOM* (*Document Object Model*, es la estructura básica de la página desplegada en el navegador.) 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, generalmente son usados los *cookies* pero cuando la aplicación se compleja 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 esté representado objetos definidos toda 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 lo cambios ocurridos en el *Modelo* y reflejar estos cambios en el *DOM*, el cual es una renderización de un *Template*.

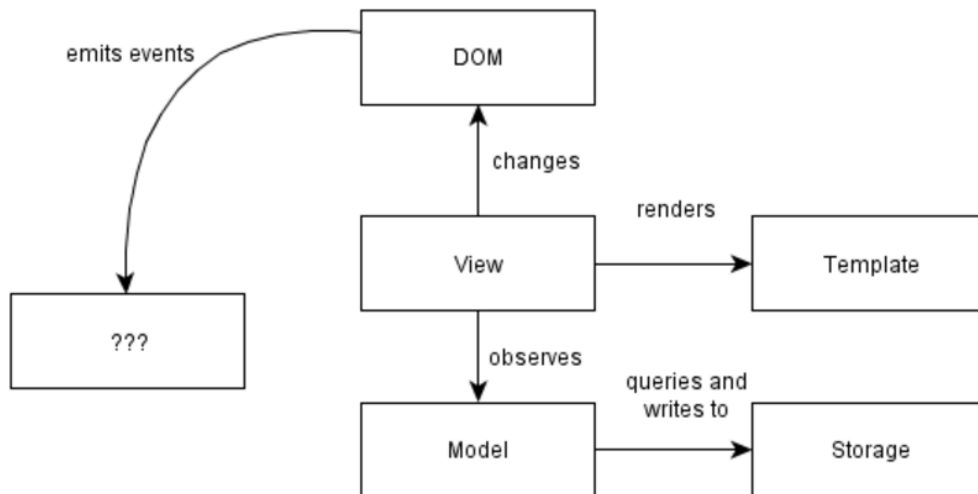


Figura 2.1: Arquitectura de una Aplicación Web Moderna

Fuente: Web [3]

Como se puede observar en la figura 2.1, 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 *Backbone.JS*, *AngularJS*, *EmberJS*, etc. Los cuales agilizan en gran medida la implementación de una aplicación de página única.

2.3. 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?

2.3.1. 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”.^[17]

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. 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 son de 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, etc.) para establecer las acciones u operaciones que se ejecutarán sobre los “recursos” que maneja el servicio web.

2.3.2. 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. [15]

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*). [19]

Los principios de diseño sobre los que se basa REST son:

- 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. [18]

Al implementar una aplicación *RESTful*¹ 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

¹Se denomina RESTful a los sistemas que siguen los principios REST

informacion va en el cuerpo del request **PUT** se usa para actualizar objetos, **DELETE** es usado para borrar objetos.

2.4. Node JS

Node.js 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.

Node 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.

Node.js 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, Node 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 *Express.js*.

2.5. Express JS

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

El Cliente (navegador web, aplicación móvil, etc) envía una petición web y el servidor web de Node.js maneja los protocolos web, leyendolos y enviandolos a una aplicación *Express* 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 Express no es necesario una gran función para manejar un request, Express contiene utilidades que permite escribir funciones más pequeñas para facilitar el manejo de las peticiones web, haciendo uso de “middleware” y “routing”.

2.5.1. Middleware

Node.js maneja una función para trabajar con una petición web, en cambio *Express* 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 denomina *Middleware functions* o Middleware.

2.5.2. Routing

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

Express.js 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 llamadas a la base de datos PostgreSQL se usó la librería *knex*.

```
database.any("SELECT * FROM users WHERE id = $1", [userId])
  .then(function (data) {
    response.send(data.name);
  });
```

2.6. 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 [5]:

- 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 *HTML-HandleBars* el cual es motor de plantillas se usa para separar el diseño HTML de Javascript, para así escribir código mucho más limpio, que permite embeber código enlazando o sincronizado con el Controlador. Esto significa que si actualizamos código en la Vista, este es actualizado en el Controlador y viceversa. [<http://handlebarsjs.com/>]

En Ember 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 guardarlo 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 addons 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 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 nos 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 de la imagen al celular sin hacer uso de nuestro backend o servidor, 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.

2.7. 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.

2.7.1. 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 documentado. 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.
- Funciones/procedimientos almacenados.
- Soporte a la especificación SQL.

Personalmente se escogió trabajar con PostgreSQL como DBMS cuenta con una extensa documentación, y gracias a su carácter “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 ó espaciales.

2.7.2. PostGIS

PostGIS es un módulo que añade soporte de objetos geográficos al DBMS PostgreSQL, convirtiéndola en una base de datos espacial para su utilización en un Sistema de Información Geográfica o *SIG*, es bastante común utilizar el acrónimo en Inglés, *Geographic Information System* o *GIS* y de ahí 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*, declarándola como software libre que lo protege de cualquier intento

de apropiación.

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 documentación bastante extensa y equipo de desarrollo que continuamente va sacando nuevas versiones, actualmente se encuentra la versión 2.2.2, pero para el desarrollo de la aplicación se hizo uso de la versión 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 están 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.

2.7.2.1. pgRouting

pgRouting es una extensión 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 sistema.

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 Salesperon Problem)
- Cálculo de ruta (Isolíneas)

2.8. Geolocalización

La Geolocalización o Georreferenciación es un termino 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.[13]

La Georreferenciación antiguamente era bastantemente usada en el ambito 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 comun, de hecho todos los smartphones actuales (celulares con Android, Iphones) traen integrados receptores GPS (Global Posicion 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 seria muy difícil sin la ayuda de los *Sistemas de Información Geografica*.

Un *SIG* es una herramienta que permite integrar, analizar, mostrar, interpretar y entender las relaciones, patrones y tendencias de la información geográficamente referenciada. 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, se se quisiera planificar la construcción de un colegio se podria integrar los datos del censo con un mapa, identificando los sectores con mayor porcentaje de niños y localizando los sectores mas 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, ayudaria en la evaciación de las personas del lugar.

2.8.1. Definiciones

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

Coordenada: Es una secuencia de n-numerós 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 como las coordenadas son asignadas a cada punto.

Punto: Es la representación de una posición, topológicamente 0-dimensional (no tiene volumen, area, longitud o cualquier otra unidad multi-dimensional).

Estas definiciones estan desarrolladas en la especificación **Simple Feature Access**, la cual es mantenida por la OGC (Open Geospatial Consortium). Esta especificacion define el conjunto de tipos de datos (puntos, linea, poligono, etc) y las operaciones o metodos necesarios para manejar estos datos.

2.8.2. Sistema de Coordenadas para datos Geográficos

Se podria pensar en un sistema de coordenadas como la forma de dar sentido a un *par de coordenadas*, por ejemplo cuando se ve una locación “POINT(-66.1457475 -17.3937285)”, como se interpretan estos números?. Podria ser la latitud y longitud del campus de la UMSS, o podria ser un sistema de años luz desde alguna estrella en el Universo. El sistema de coordenadas es lo que diferencia estos casos.

Una aplicacion que maneja datos geograficos, generalmente trabaja con sistemas de coordenadas relacionadas con la superficie terrestre, conocidas como coordenadas espaciales (coordenadas globales), que permiten representar la tierra en *3-Dimensiones* (3D), ya que esta es una Esfera (elipsoide oblato), o en una representacion de la superficie terrestre en *2-Dimensiones* (2D), se pueden nombrar los siguientes:

2.8.2.1. 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.

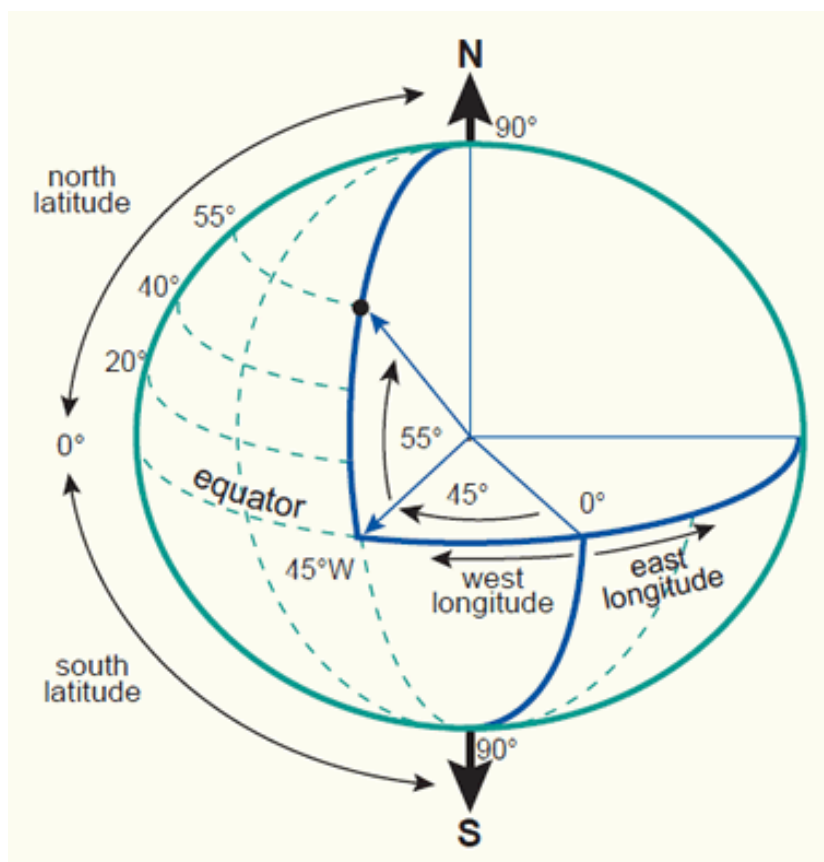


Figura 2.3: Sistema de coordenadas Geográficos

Fuente: Web

2.8.2.3. 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 *medida lineales* para las coordenadas, de forma que los cálculos de distancia y área se pueden realizar en términos de esas mismas unidades.[24]

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 que se quiere representar de la Tierra.

La proyección que usan *Google Maps* y *Open Street Maps* es la **Mercator Projection**[14], 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 2.4

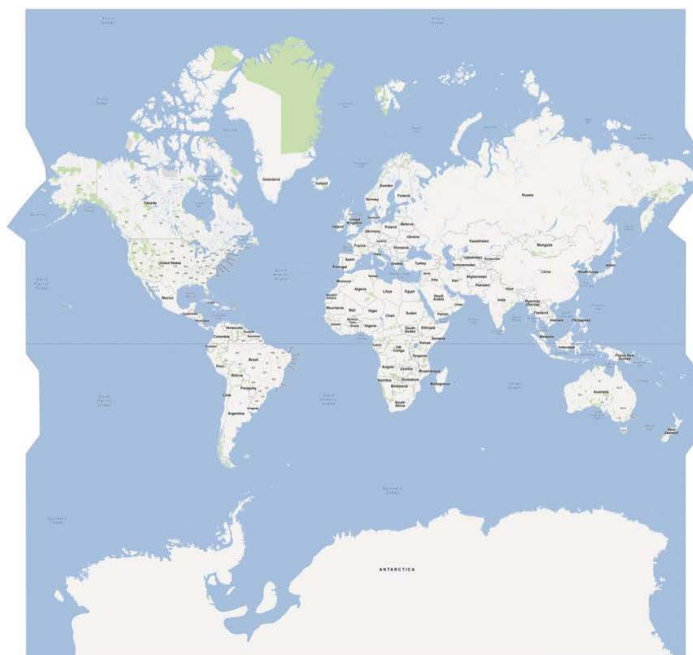


Figura 2.4: Sistema de coordenadas Proyectadas

Fuente: Web

Tal como se puede apreciar en la figura 2.4, la distorsión de esta proyección se hace evidente si se observa la zona de Groenlandia ya que parecería tan grande como Africa o America del Sur, cosa que no es cierta, ya que Groenlandia es casi 14 veces mas pequeño que Africa. A pesar de esta distorsión tan marcada, la **Proyección de Mercator** es una de las mas usadas.

Los Mapas son herramientas muy útiles a la hora de desplegar información pero realizar el mapa, crear las fórmulas matemáticas con las cuales se trabajará, determinar cómo se usarán estas fórmulas para una representación adecuada de la superficie terrestre, es una tarea muy compleja. Como programador la tarea más complicada fue determinar el tipo de mapa y el sistema de coordenadas más adecuado para el tipo proyecto que se necesita desarrollar.

Los términos de longitud y latitud son en un inicio, más fácilmente comprendidos que un sistema proyectado, pero no se puede tomar a la ligera una correcta comprensión del uso de los *sistemas de coordenadas* en una base de datos espacial, un mal uso de estos conceptos puede generar errores a la hora de manejar datos espaciales o en el resultado de las operaciones sobre estos datos, llegando a resultados no deseados y que pueden costar mas tiempo y dinero en una posterior corrección.

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 nos ayuda a movernos 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.

2.9. Ruta Óptima dentro el campus Universitario

Si se quiere ir de un punto a otro el mejor camino o el más óptimo siempre es aquel con la menor distancia entre los 2 puntos, pero como se puede definir que un camino es óptimo?, si se va en coche hay que tomar en cuenta la dirección de las calles, los cruces, etc. si se va a pie hay que ver las características del terreno, caminos cortados, distancias, etc.

Si se analiza el terreno que se va a cubrir con la aplicación (el campus de la UMSS ubicado entre las calles Oquendo, Sucre, Belzu y M. U. Lopez), se tiene que el camino óptimo es siempre el más corto o de menor longitud, ya que el metodo de desplazamiento que se tomara en cuenta para moverse dentro del campus Univeristario sera *a pie* y el terreno es plano.

La resolución de este problema es la se analizará en este capítulo.

2.9.1. Grafos

El problema se lo podria definir como; Encontrar la ruta más corta de un punto a otro punto, en donde los puntos están interconectados por una red de caminos. El problema descrito se lo puede resolver/describir como un caso específico de la teoría de grafos.

2.9.2. Definiciones

Primeramente es necesario aclararar alguno términos usados en la teoría de grafos.

Un **grafo** G consiste en un conjunto de vértices V y un conjunto de aristas A , y se lo representa con $G(V, A)$.

El **vértice** v es adyacente a u , o a un vecino de u , si y sólo si $(u, v) \in A$. Por lo tanto, en un grafo no dirigido, dado una arista (u, v) , v es adyacente de u , y simétricamente u es adyacente de v . Los vértices también son llamados nodos.

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. En un grafo no dirigido el par de vértices que representan la arista no tiene orden, por lo tanto la arista (u, v) y (v, u) representa la misma arista. En cambio en un grafo dirigido la arista (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*.

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 como 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 $(u, v) \in A$.

Un camino en un grafo es una secuencia de nodos v_1, v_2, \dots, v_n tal que $(v_1, v_2), (v_2, v_3), \dots, (v_{n-1}, v_n)$ son aristas.

2.9.3. Representacion de un Grafo

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.

2.9.4. 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 no es dirigido la matriz de adyacencias es simétrica. La matriz de adyacencias, que se puede observar en el cuadro ??, es la misma matriz de la relación A de V en V porque indica cuales vértices están relacionados (unidos por una arista).

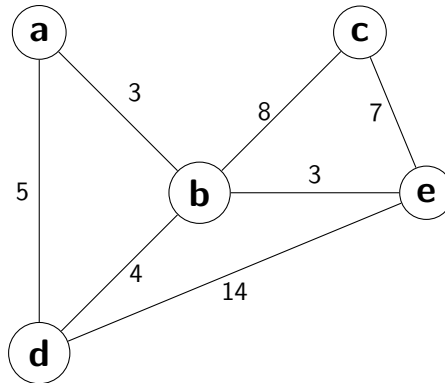


Figura 2.5: Grafo ponderado no-dirigido

Fuente: Elaboración propia

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

Matriz de adyacencias del grafo de la figura 2.5

2.9.5. El Problema de la ruta mas corta

Dados los vértices v_i y v_j de un grafo $G = (V, A)$ se llama trayectoria mínima o camino minimo de v_i a v_j al numero 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 minimo 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.

2.9.5.1. 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:

- Puede ser permanente: en este caso la distancia encontrada es a lo largo de la trayectoria, la más corta de todas las encontradas.
- 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. [26]

Existen numerosas soluciones para encontrar la ruta óptima, donde se toman en cuenta diferentes variables y heurísticas, en el que cada algoritmo presenta ventajas respecto a las demás. La teoría de grafos es un tema extenso y para fines prácticos solo se explicó el algoritmo de Dijkstra por ser el que se está usando en la aplicación desarrollada.

El algoritmo de Dijkstra puede ser una de las soluciones más sencillas y que requiere muchos más cálculos que las demás pero el grafo implementado al no ser extenso, por extenso se podría entender un grafo con millones de aristas y vertices como se puede dar en el caso de una Ciudad o un País, pero en el presente caso al ser los predios de campus Universitario no existe una razón de gran peso para implementar otra solución más eficiente en el manejo de recursos.

El problema de la ruta más corta es ampliamente usado por las empresas de transporte, correos, etc., que necesitan mejorar la eficiencia del trayecto y a la vez reducir el consumo de combustible, dentro del campus universitario, reducir el tiempo en el cual encontramos un aula o una oficina mejoraría en gran medida la presentación de la Universidad hacia gente externa que necesitan hacer uso o encontrar algún lugar en específico ya que lamentablemente esta información actualmente sólo se la pueden ofrecer las personas que conocen el lugar de antemano y aun en esos casos existe la posibilidad de no encontrar el lugar que se está buscando.

2.10. Metodología de Desarrollo

La metodología para el desarrollo de software nos 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 nos ayudara a organizarnos y a seguir un ritmo de trabajo.

Para este proyecto de grado se hará uso de una metodología Ágil. Para lo cual se definirá en que se basan las metodologías ágiles.

Para este proyecto de grado se va a ser uso de XP como metodología ágil.

2.10.1. 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, esta demostrado que este tipo de metodologías son muy rígidas y les falta flexibilidad a la ho-

ra 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 de manifiesto ágil:

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.[7]

Para alcanzar este objetivo es que las metodologías ágiles se basan en tres principios[8]:

- 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 metodo o tecnica de desarrollo, en cambio son un grupo de metodologías que implementan los principios ágiles. Entre los cuales se pueden apreciar las siguiente metodologías:

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

Por las siguientes características de la Metodología *Programación extrema*, la cual viene del ingles *eXtreme Programing* por lo cual generalmente nos referiremos a esta como *XP* es que es la escogio para implementar este proyecto de grado.

2.10.2. Programación Extrema

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)[9] 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)* en **Ford Motor Co**[9] se demostró que esta metodología de desarrollo es un método apropiado para llevar a buen termino el proyecto de desarrollo.

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 encontró que cuatro enunciados las cuales son la base de la filosofía de *XP*[9]:

- 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 cohesionandolo 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 programación extrema.

La programación extrema se caracteriza por las siguiente practicas:

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 día.

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.

2.10.2.1. 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.[10]

2.10.2.2. Proceso de desarrollo

La programación extrema identifica las siguientes fases en el proceso de desarrollo de software

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.[10]

2.10.2.3. 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.[11]

2.11. Desarrollo del Proyecto usando XP

Ya que para el desarrollo del proyecto se usará programación extrema, se va a definir el proceso de desarrollo general que se usará en este proyecto de grado.

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 el cliente es capaz de ofrecer feedback del producto.

En un proyecto que sigue la metodología XP los releases son frecuentes, esto para recibir feedback más seguido. Los releases son negociados en un Planning Game, donde los clientes definen qué se va a implementar en el release y los desarrolladores especifican el tiempo que necesitan para desarrollar las características deseadas.

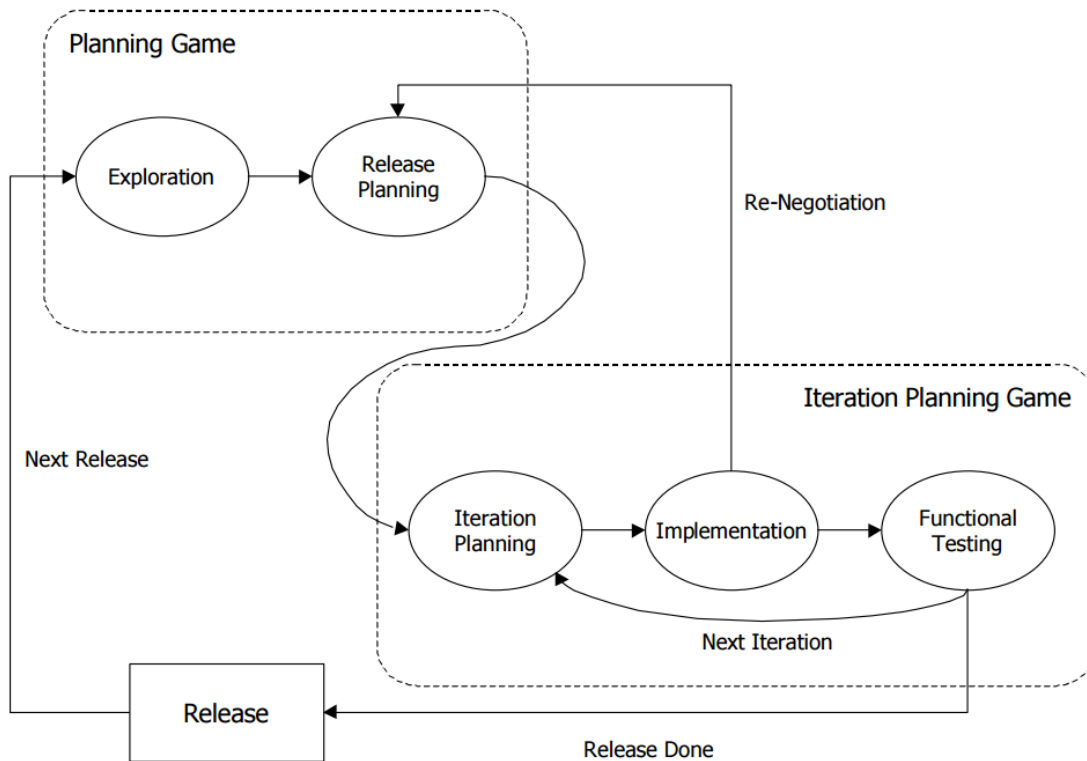


Figura 2.6: Diagrama del proceso XP

Fuente: Web

En la figura 2.6 se puede apreciar el proceso de desarrollo de la metodología XP, la cual se explicará a continuación.

2.11.1. Planning Game

La fase del planning game consiste de 3 etapas: exploración, planeación y dirección.

Durante la fase de la **exploración** los clientes definen lo que desean que tenga el sistema y los desarrolladores estiman el tiempo necesario que necesitan para realizar esas tareas.

Durante la **planeación** se negocia y se decide cuales de todas las características que los clientes quieren pueden llegar a realizarse en el tiempo diseñado para una iteración.

Después de la planeación sigue la fase de **dirección**, durante la cual se desarrolla y actualiza (cuando sea necesario) la planeación ya negociada según lo que se vaya aprendiendo a medida que avanza el desarrollo del proyecto.

2.11.1.1. Exploración

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.

Una vez que estas tarjetas están escritas, el equipo de desarrollo debe asignarles una estimación en términos del tiempo necesitado para el desarrollo y el riesgo para el producto que pueden llegar a tener las características detalladas en las historias del usuario.

Las historias del 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 cuando las historias están por ser implementadas en las “tareas de ingeniería”.

2.11.1.2. Planeación

Cuando se tienen las historias del usuario junto con las estimaciones de los desarrolladores, se está listo para una “negociación de aceptación” donde un “calendario de entregas” es negociado y donde se aceptan qué historias se van a desarrollar primero en cuanto tiempo se van a entregar resultados.

2.11.1.3. Dirección o Steering

Esta fase es básicamente comprende el resto del desarrollo del producto hasta que es liberado al mercado o el proyecto es cancelado.

Esta fase consiste en 4 “movimientos”:

Iteración: Es durante la iteración cuando se desarrolla el producto, el tiempo que se utiliza para esta fase es de generalmente de una o 2 semanas, dependiendo de la naturaleza del proyecto.

Recuperación: Si durante el desarrollo no se completan las características a desarrollar en el tiempo establecido, es durante la recuperación que se re-negocia con el cliente si quiere cambiar la fecha de entrega del release o modificar el alcance del desarrollo (menos historias de usuario).

Nuevas Historias: El cliente tiene el derecho de aumentar historias de usuario, las cuales se tienen que estimar y negociar si serán parte del actual desarrollo, en tal caso se tiene que renegociar las fechas de entrega.

Re-estimación: Si durante el desarrollo el equipo considera que el plan ya no es correcto, todas las historias que faltan hacer se tienen que re-estimar y el plan se tiene que re-negociar con el cliente.

2.11.2. Iteration Planning Game

La fase de Iteración en XP se lo denomina como Iteration Planning Game, esta al igual que el release planning game consiste en las fases de: Exploración, Planning e Implementación. Steering.

Hay que tomar en cuenta que la planeación de una iteración en particular es desarrollada al inicio de cada interacción, no se planifican iteraciones por adelantado.

Generalmente cada 3 iteraciones se actualiza el Calendario de Entregas “committed schedule” para reflejar los logros alcanzados por el equipo de desarrollo y el estado del proyecto, también sirve para identificar posibles riesgos.

2.11.2.1. Exploración

Durante esta fase el cliente escoge las historias de usuario que serán implementadas en la presente iteración, generalmente se escogen las que aportan más valor al producto o tienen más relevancia en la lógica de negocio del cliente, así como también cualquier historia que no se acabó en una iteración anterior.

Los desarrolladores dividen las historias en Tareas de Ingeniería, las cuales son más pequeñas que las historias, si se encuentra que una Tarea es casi tan grande como una historia es porque es una historia y debería ser dividida en Tareas. Una tarea puede estar relacionada a 2 o más historias o no estar relacionada con ninguna historia. Dentro de la metodología XP es una buena práctica el escribir las tareas en las “Index Cards”, similares a las historias, esto debido a que estas tarjetas son fáciles de manipular durante la fase de planeación.

2.11.2.2. Planeación

Durante esta fase un desarrollador acepta la responsabilidad de implementar una Tarea de acuerdo de su experiencia personal en el área y tecnologías que se usarán en el desarrollo. El desarrollador debe estimar el tiempo necesitado para completar la tarea en un *Ideal Engineering Time* Tiempo de Ingeniería Ideal, siempre hay que considerar que la tarea se deriva de la Historia de usuario que es escogido para la iteración actual, una regla de XP consiste en que no se debe realizar trabajo el cual no se va necesitar ahora, **YAGNI**².

La Tarea combinada con el nombre del desarrollador responsable, la estimación asignada son parte del *Iteration Schedule* ó *Plan de la Iteración*, con el cual el equipo de desarrollo es capaz de determinar si la iteración está *floja* o *cargada*, si está *floja* se pueden añadir otras historias a la iteración o si está *cargada* es necesario dividir historias. Finalmente cuando la carga de trabajo de la iteración está balanceada se procede con la siguiente fase, la *Implementación* de las tareas.

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

2.11.2.3. Implementación

Dentro de lo que es el ciclo de desarrollo de software, **XP** define el siguiente procedimiento:

- 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).

2.11.2.4. Registrar el Avance

XP define un rol en específico que se encarga de medir el progreso, el Tracker.

2.11.2.5. Verificación

Cada historia lleva asociado test funcionales, que están diseñados para verificar que los criterios de aceptación de cada historia están implementados.

Si durante esta fase las pruebas fallan, la historia de usuario relacionada se marca para volver a trabajar en ella en la siguiente iteración.

Capítulo 3

Caso Estudio: Campus Universitario

En el presente proyecto se implementó una aplicación web móvil, como ya se explico, este tipo de aplicaciones son básicamente páginas web, con amplia funcionalidad e implementadas específicamente para su uso en dispositivos móviles, tablets o smartphones, para lo cual se investigó las tecnologías, arquitecturas y los frameworks de desarrollo que actualmente usan este tipo de aplicaciones, una de los concepto que se estudió fue de las *aplicaciones de página única*, concepto que se empezó a discutir a principios de 2003, y se puso de moda con la aparición de frameworks JavaScript que agilizan y facilitan en gran medida la implementación de aplicaciones web, tales como *AngularJS*, *BackboneJS* y en el 2010 *EmberJS*.

Es necesario reconocer las diferentes partes con la que cuenta una aplicación web móvil como una aplicación de página única diseñada para dispositivos móviles, es necesario implementar un REST API que se encargue de hacer las consultas a la base de datos y que el cliente pueda consumir la información obtenida de la base de datos así como también el camino inverso, la base de datos necesita manejar informacion geoespacial y el cliente necesita que la aplicación sea optimizada para su uso en dispositivos móviles. Tomando en cuenta estos requerimientos se puede separar el desarrollo del proyecto en los siguientes apartados.

3.1. Generar el mapa con información geográfica

Para poder responder al problema de encontrar una ruta óptima entre 2 puntos dentro del campus universitario, se necesita de un mapa que contenga todas las rutas que existen dentro del campus.

En primer lugar fue necesario obtener un grafo ponderado no-dirigido que represente un mapa de los caminos que existen dentro del campus Universitario.

Para obtener este mapa se procedió a caminar a través del campus de la UMSS con un GPS Garmin Nuvi 1300, el cual es un dispositivo GPS básico pero cumple con la función de guardar información geográfica, los archivos generados tienen extensión *gpx*, que básicamente es un fichero XML estándar usado para compartir datos entre GPS's, se recorrieron 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, se utilizó el archivo *current.gpx* para exportar la información a un archivo *shapefile*, para esta tarea se utilizó QGis, con el cual se acabó editando las rutas recogidas por el GPS.

Este paso fue necesario porque el mapa extraído del GPS es una línea única, pero para que nos sirva para el objetivo de buscar una ruta óptima, es necesario que esta línea sea dividida o separada en muchas líneas, las cuales son las aristas y los extremos de las líneas serán los nodos o vértices del grafo.

Implementando el algoritmo de *Dijkstra* en el grafo resultante es lo que nos permitirá encontrar la ruta más corta dentro del campus Universitario, al tener una gran cantidad de información resultante de la obtención de datos mediante un dispositivo GPS se hace imprescindible usar una base de datos que nos ayude con esta tarea, para lo cual se usó la base de datos PostgreSQL añadiéndolo PostGIS y pgRouting, herramientas ampliamente utilizadas en el manejo de datos geo-espaciales.

Técnicamente esta línea única es representada como un *POLYLINE* el cual consiste en una o más partes. Una parte es una secuencia conectada de dos o más puntos. Las partes pueden o no estar conectadas entre sí. Las partes pueden o no intersectarse entre sí, para transformar este *POLYLINE* necesitamos separar todas sus partes y convertirlas en objetos *LINESTRING* únicos, y a este conjunto de *LINESTRINGs* es el que se va a usar en la base de datos como mapa de “rutas”.^[27]

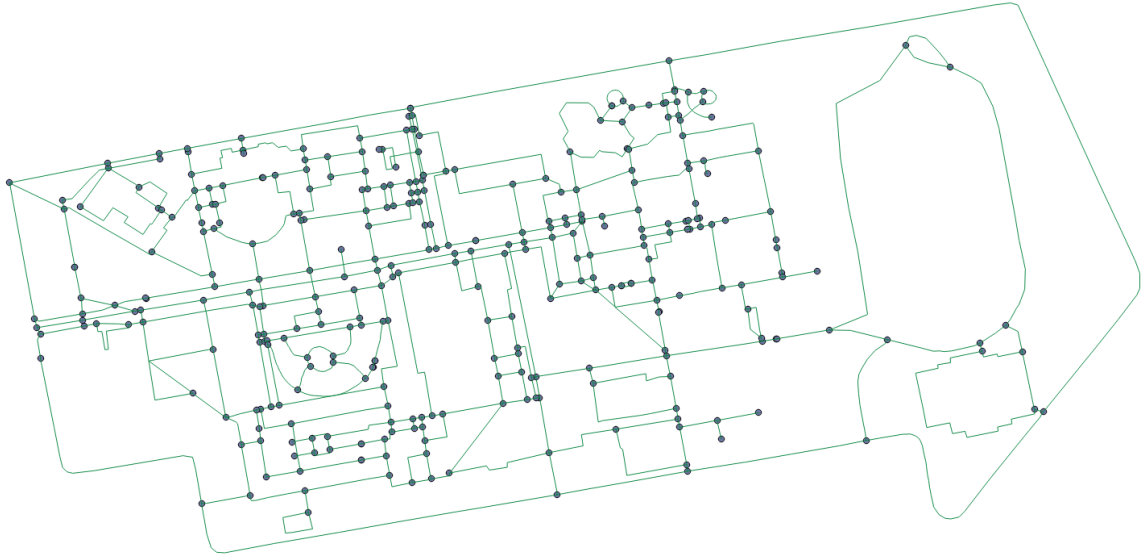


Figura 3.1: Shapefile del campus Universitario.

Fuente: Elaboración propia

En la figura 3.1 se puede apreciar el shapefile resultante de la división del POLYLINE original, las líneas que conforman el mapa de las rutas del campus Universitario, donde cada línea es una arista y los puntos son los nodos del grafo no-dirigido, que será usado para la resolución del problema de la ruta más corta en el presente proyecto de grado.

Para una mejor apreciación del grafo que consta de 1164 aristas y 1003 vértices, se lo puede ver en combinación o proyectada en un mapa de rutas del campus de la Universidad Mayor de San Simón ubicado entre las calles Oquendo, Sucre y Belzu de la ciudad de Cochabamba - Bolivia, se puede referir a la siguiente figura 3.2.

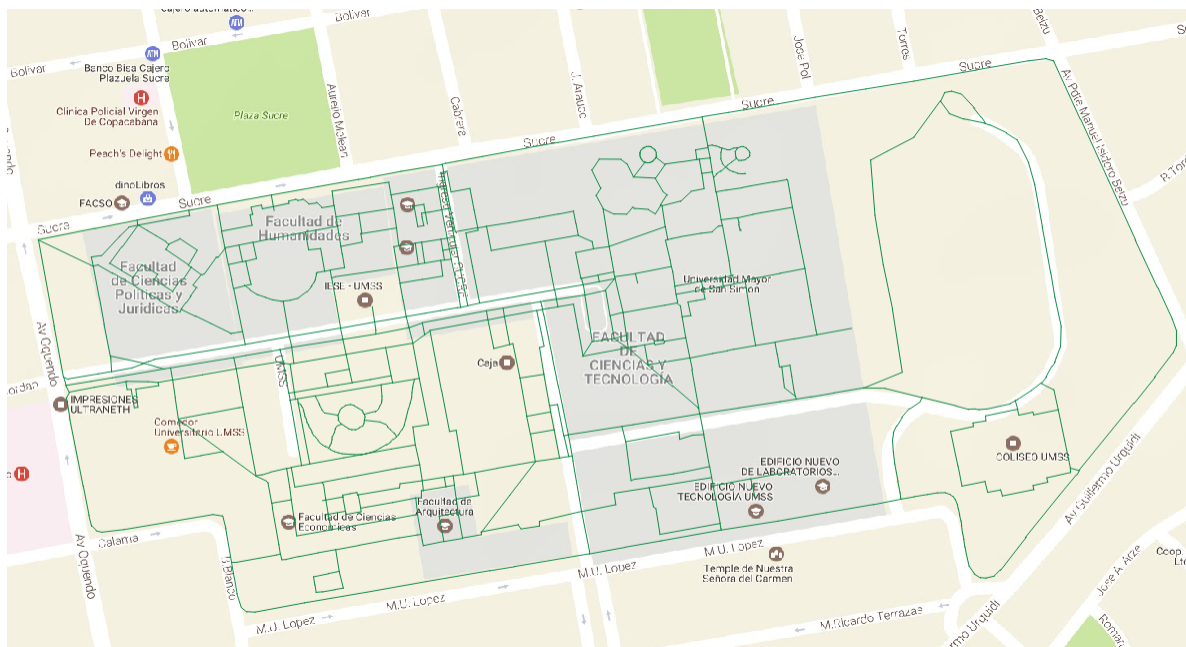


Figura 3.2: Shapefile sobre el campus Universitario de la UMSS.

Fuente: Elaboración propia

3.1.1. Facultad de Derecho

La facultad de Derecho cuenta con alrededor de 178 vértices y 88 aristas, está ubicada al nor-oeste del Campus Universitario, en la esquina de la calle Oquendo y Sucre, dentro del campus colinda con la facultad de Humanidades hacia el Nor-Este y hacia el Sur-Oeste está la facultad de Economía.

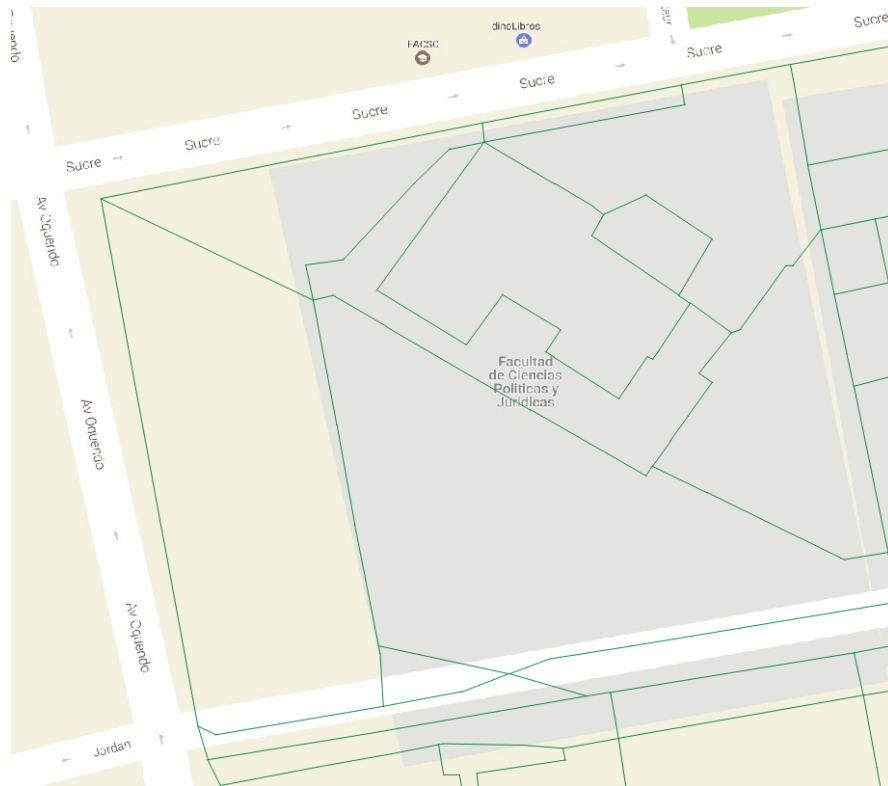


Figura 3.3: Facultad de Derecho - UMSS

Fuente: Elaboración propia

En la figura 3.3 se puede observar en la línea verde los caminos o rutas dentro de la facultad de derecho de la UMSS, proyectada sobre el mapa de Google Maps, para lograr esta representación se utilizó QGIS ya que la información geográfica de la ruta está contenida en un archivo shapefile y el mapa se lo obtiene usando el API de Google Maps gracias al plugin de QGIS, *QuickMapServices*.

3.1.2. Facultad de Economía

La facultad de Economía está compuesta de XX aristas y XXX vértices, está ubicada en el sector Sur-Oeste del campus Universitario, 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 3.4.

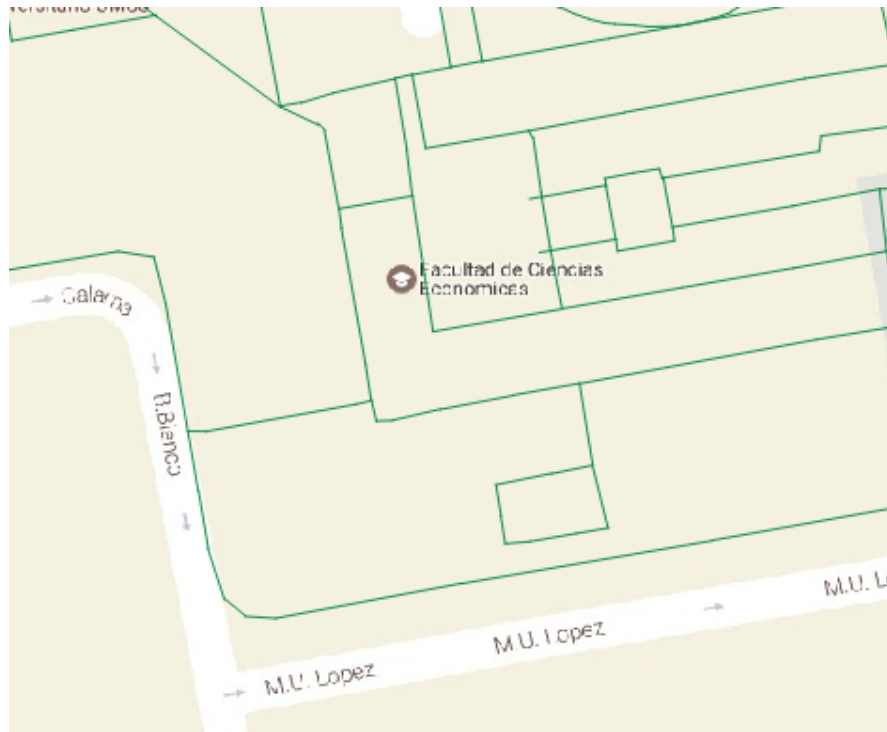


Figura 3.4: Facultad de Economía - UMSS

Fuente: Elaboración propia

3.1.3. Facultad de Humanidades

La facultad de Humanidades está compuesta por XX aristas y XXX vértices, colinda con la calle Sucre hacia el Nor-Oeste, dentro del campus Universitario se encuentra la facultad de Tecnología hacia el Nor-Este, hacia el Sur-Oeste está la Facultad de Derecho y al Sur-Este se encuentra la Facultad de Arquitectura, en la figura 3.5 se puede apreciar la facultad de Humanidades dentro del campus Universitario sobrepuesto con el mapa de rutas identificado por la línea verde.

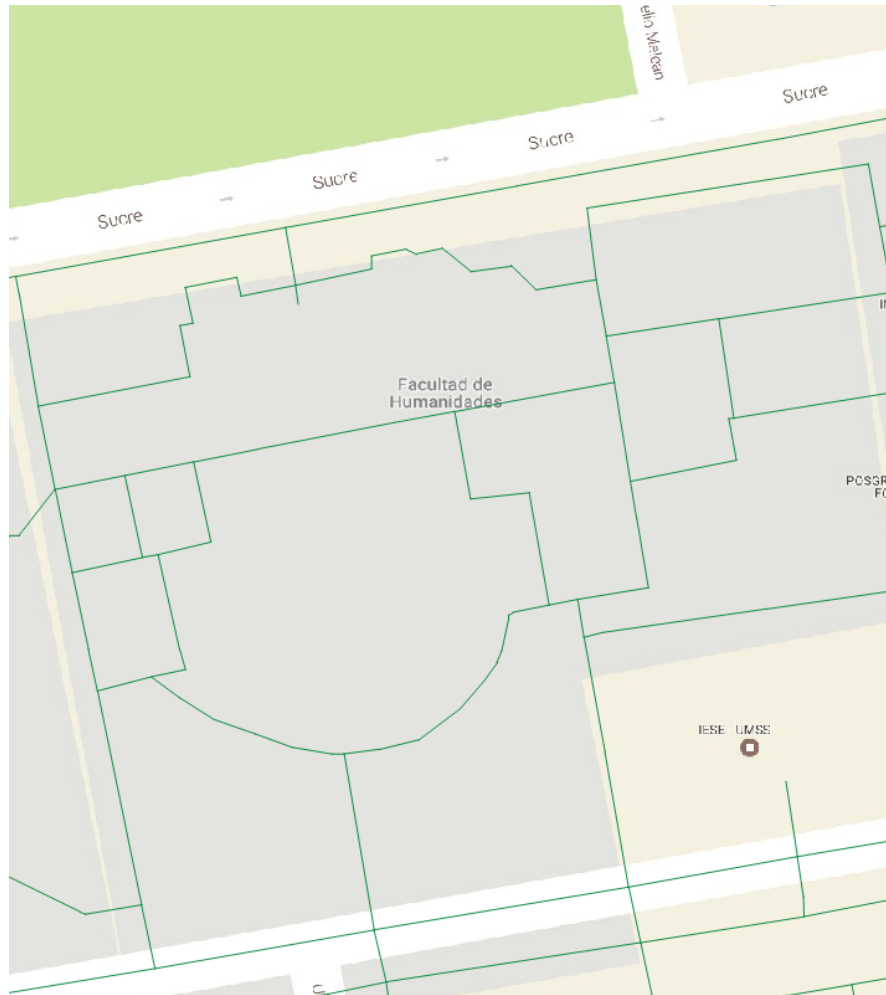


Figura 3.5: Facultad de Humanidades - UMSS

Fuente: Elaboración propia

3.1.4. Facultad de Tecnología

La facultad de Tecnología se encuentra en el extremo Nor-Este dentro del campus Universitario y cuenta con XX aristas y XXX vértices, la facultad de tecnología colinda hacia el Nor-Oeste con la calle Sucre, y hacia el Este con la calle Belzu, 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 3.6 se puede apreciar el mapa de rutas de la facultad de Tecnología.

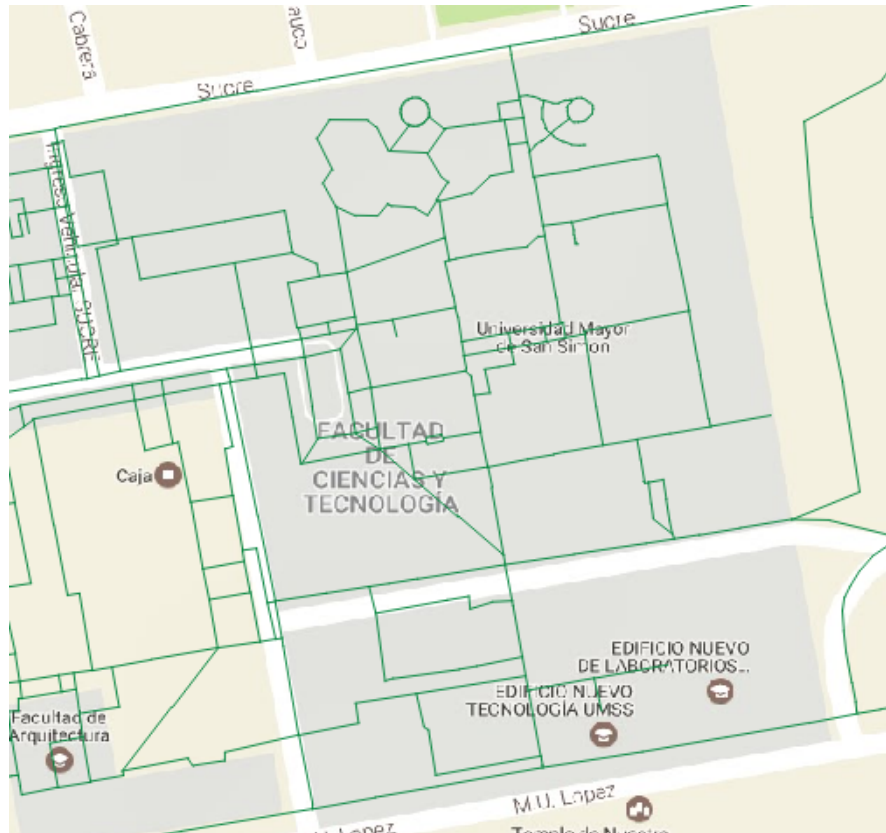


Figura 3.6: Facultad de Tecnología - UMSS

Fuente: Elaboración propia

3.1.5. Facultad de Arquitectura

El grafo correspondiente a la facultad de Arquitectura cuenta con XX aristas y XXX vértices, la facultad 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, el grafo se puede apreciar en la siguiente figura 3.7.

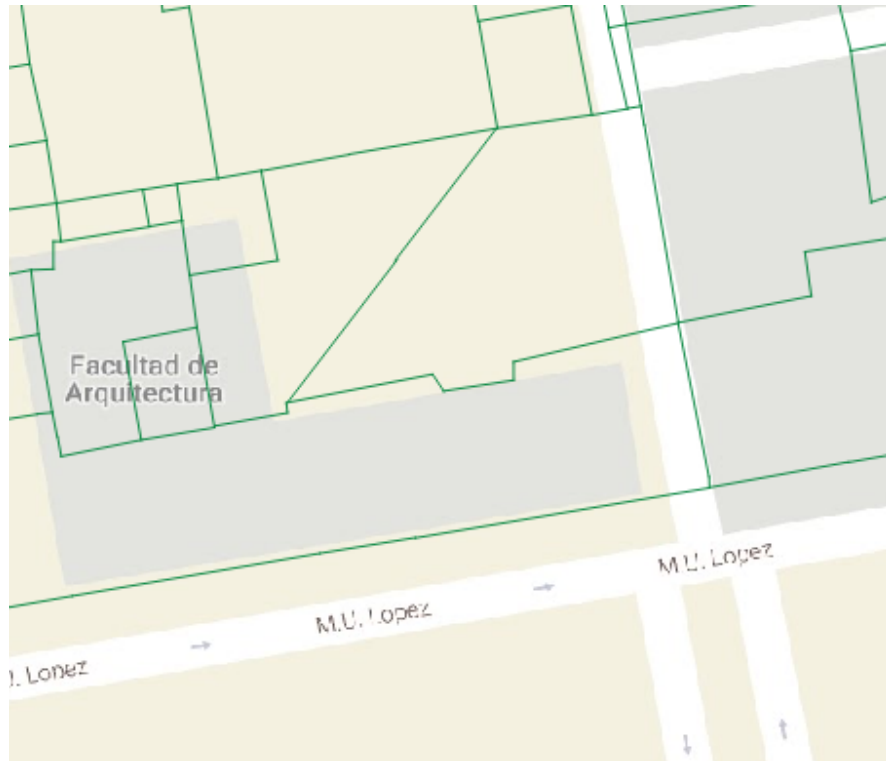


Figura 3.7: Facultad de Arquitectura - UMSS

Fuente: Elaboración propia

3.2. Backend

El backend de la aplicación comprende el manejo de la información y cómo es procesada para su consumo por parte del frontend y almacenamiento en la base de datos, por lo tanto se pueden observar dentro del backend, 2 secciones claramente diferenciadas: el servidor web y la base de datos.

3.2.1. Servidor Web

Un servidor web se puede referir a 2 cosas, la primera es la máquina en la cual se instalará y alojara toda la lógica de negocio que requiere la aplicación, la segunda se refiere a la capacidad de poder recibir y dar información al cliente del servidor, el segundo caso es también conocido como un *servicio web* el cual será discutido en esta sección.

El servidor web es el encargado de enviar la página HTML la primera vez que el cliente hace un request al servidor, una vez que la página está cargada el servidor solo se encarga de enviar y recibir información de la base de datos al cliente.

El servidor web será construido con *Express JS*, para lo cual primeramente se necesita instalar *Node JS*, *Express JS* cuenta con una herramienta para la consola, *express-generator*, entonces solo es necesario crear un proyecto Express y empezar a implementar la lógica de negocio.

El servidor se va a encargar de hablar con la base de datos y también necesita ofrecer una interfaz por la cual pueda recibir datos y entregar datos, en pocas palabras un *API*, implementar un *API* con *Express* es bastante sencillo, para trabajar con la base de datos se instaló una librería *BookshelfJS* para poder escribir las consultas y trabajar con la base de datos como si se tratara de objetos, este patrón se denomina *ORM* o *Object-Relational-Model*, básicamente mapea las tablas y las relaciones existentes como objetos dentro del paradigma *OOP* y el framework se encarga de traducirlo a lenguaje *SQL* que es el lenguaje que entienden las bases de datos, *BookshelfJS* también permite escribir las consultas en lenguaje *SQL* si fuera necesario, como en el presente caso ya que al trabajar con una herramienta específica, *pgRouting*, las consultas son personalizadas.

```
var getPlace = (req, res) => {
  var id = req.params.id;
  var raw = "SELECT " +
    " ST_AsGeoJSON(geom)::json As geometry," +
    " name," +
    " description," +
    " phone," +
    " level," +
    " gid As id " +
    " FROM place WHERE gid = " + id;
  Bookshelf.knex.raw(raw)
    .then((data) => {
      res.json(data.rows[0]);
    })
    .catch((error) => {
      console.log(error);
      res.send("Error");
    });
};
```

Como en la anterior consulta, se requiere obtener la información de un lugar, por lo tanto se necesita usar la forma *Raw SQL* o consultas en “SQL puro” ya que el fuerte de *BookshelfJS* es el manejo de las consultas en forma de objetos (ORM), lamentablemente actualmente no existe mucho soporte para manejar datos geoespaciales por parte de frameworks ORM.

Una vez que se obtiene la información del lugar; nombre, descripción, teléfono, el nivel o piso, pero lo importante de esta consulta es la obtención del “punto” geoespacial del lugar. Estos datos son importantes para implementar la lógica de negocio.

"POINT (-66.14857015827988 -17.394421906929086)"

Este atributo es de tipo *punto* o *point* el cual tiene un *SRID* o *Spatial Reference System Identifier*, el *SRID* corresponde a un sistema de referencia espacial basado en el elipsoide concreto usado para la creación de mapas de tierra plana o de tierra redonda.[25]

El *SRID 3857* también conocido como *EPSG 3857* es la implementada en la *Proyección de Mercator*, el *SRID* es la llave primaria de la tabla *spatial_ref_sys* que se crea cuando se inicializa *PostGis* en una base de datos para que esta soporte información geoespacial, esta tabla provee la información necesaria para interpretar y convertir correctamente todas las coordenadas existentes, el *SRID 3857* está definida en la tabla *spatial_ref_sys* como “Popular Visualisation CRS / Mercator”.

En resumen el servidor web será el encargado de recibir información y devolver una respuesta de acuerdo a los datos que recupere de la base de datos, para tal efecto se utilizan las direcciones URL, las cuales de acuerdo al protocolo HTTP se pueden definir de acuerdo a la acción que el servidor necesita procesar, para lograr esta comunicación es necesario definir nuestro API.

3.2.1.1. Implementación del REST API

El servidor necesita reconocer las peticiones que le llegan del cliente, para lo cual es necesario “mapear” un URI a una acción específica, las cuales ya están preparadas para comunicarse con la base de datos, no hay restricción en la declaración de las URIs pero para una mejor comprensión del API que se está desarrollando es necesario seguir convenciones que aseguran que cualquier desarrollador pueda comprender el API presentado y pueda ser fácilmente consumido por cualquier aplicación que requiera acceder a la información que disponible, un API REST es el que cumple con estas características. En primer lugar es necesario crear las URIs que serán “entendidas” por el servidor, esto se logra declarando en el servicio creado con *ExpressJS*, tal como se puede apreciar en el siguiente bloque de código, cada URI se lo relaciona a un modelo en específico de acuerdo a la acción que se requiere, tal como se puede observar en el cuadro 3.1 las URIs declaradas en el API cumplen con tal característica.

Code 3.1: 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);
```

El cuadro 3.1 muestra como se puede leer las peticiones al API de **places**, las acciones mostradas son las que se pueden encontrar en un API REST pero no es necesario declararlas

todas para considerar a que un API es restful.

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

Cuadro 3.1: 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 responde mostrando la información del lugar “1” y en cambio si se genera una petición PUT a la misma dirección */places/1* se ejecuta la acción **update** y se actualizan los datos del lugar “1”.

Siguiendo la convención de un API REST ayuda a entender el flujo que tiene un recurso, las URL son legibles y únicos para cada recurso. Por lo tanto la implementación de los recursos se hace de forma más limpia y ordenada, situaciones que son claves para el mantenimiento y la extensibilidad del sistema.

3.2.2. La Base de Datos

Para preparar la base de datos es importante entender las diferencias entre los distintos tipos de sistemas de coordenadas porque computacionalmente realizar operaciones sobre los sistemas de coordenadas tiene un costo.

Si se usara el sistema de coordenadas geográfico (WSG84) este es el más apropiado si se necesitaría usar grandes extensiones de la superficie terrestre, que al ser una estructura elipsoi-dal el costo computacional para realizar las operaciones matemáticas de calcular distancias, intersecciones, etc. es más elevado. En cambio el uso de un sistema de coordenadas proyectado (Mercator Projection) tiene un costo computacional más bajo, ya que se estaría trabajando con un sistema geométrico.

También hay tomar en cuenta la base de datos, ya que será esta la que se encargará de manejar los datos espaciales. Al estar usando PostGIS, se puede ver que en su documentación que claramente exhorta el uso de un sistema geométrico sobre el uso de un sistema geográfico si se va trabajar con datos que cubran una pequeña área geográfica. Tomando en cuenta esta reco-mendación y el tamaño del área de estudio (el campus de la UMSS), se procedió a implementar

en la base de datos el uso de la proyección Mercator. Se va usar Mercator sobre las otras proyecciones porque aparte de las ventajas que se mencionaron con anterioridad, Google Maps usa esta proyección y ya que se usará este mapa lo más correcto es trabajar con la misma proyección.

Toda la información geoespacial recolectada necesariamente debe ser almacenada, para lo cual se investigó las diferentes bases de datos disponibles y tras la tarea de investigar acerca de ese problema se procede a instalar *PostgreSQL 9.4.8* sobre Linux *Ubuntu 15.10*, y para manejar datos geoespaciales se necesitó instalar *PostGIS 2.1.8*.

3.2.2.1. Los lugares

En primer lugar se recolectó la información de los lugares, que la aplicación contendrá de forma inicial, al igual que para recolectar las rutas se hizo uso de un *GPS Garmin Nuvi 1300*, el cual cuenta con la opción de guardar locaciones como favoritos, entonces solo fue necesario estar cerca del lugar que se desea guardar y activar esa opción del GPS, este guarda la información en un archivo *.gpx* y con la ayuda de *QGIS* se generó el archivo shapefile correspondiente.

Posteriormente es necesario pasar la información geoespacial del shapefile a la base de datos, para esta tarea se hizo uso de una herramienta disponible para postgres, *shp2pgsql*, que permite la conversión de un archivo shapefile a un archivo sql.

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

Con el anterior comando se tiene como resultado un archivo *.sql*, el cual es ingresado en la base de datos ya configurada, de esta forma nuestra base de datos para a contener una tabla geoespacial con datos de tipo *POINT*, los cuales representan los lugares dentro del campus de la UMSS.

El archivo *sql* resultante es usado para popular la base de datos con la información inicial de los lugares que contiene el campus universitario, para tal tarea se usó el siguiente comando.

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

SQL Editor

Graphical Query Builder

Previous queries

1

2

SELECT * FROM place

Output pane

Data Output

Explain

Messages

History

	gid integer	name character varying(254)	elevation numeric	geom geometry(Point,4326)
1	1	607	2595.90999999999854	0101000020E61000003BE0BA62468950C0DB334B02D46431C0
2	3	642 No Se	2582.21000000000036	0101000020E6100000E46723D74D8950C037E0F3C3086531C0
3	4	Baño 1	2566.82999999999927	0101000020E6100000DC48D922698950C06B11514CDE6431C0
4	5	Baquita	2573.320000000000164	0101000020E6100000AAF1D24D628950C0876BB587BD6431C0
5	6	Biblioteca	2575.00000000000000	0101000020E6100000A7B393C1518950C06DAE9AE7886431C0
6	7	Bloque	2571.869999999999891	0101000020E6100000F9A23D5E488950C01152B7B3AF6431C0
7	8	Canchas	2589.179999999999836	0101000020E6100000E82E89B3228950C0EB025E66D86431C0
8	9	Carrera De Biologia	2595.90999999999854	0101000020E610000058C51B99478950C01041D5E8D56431C0
9	10	Centro De Tecn Agroindustrial	2583.409999999999854	0101000020E61000004A5E9D63408950C0B6D8EDB3CA6431C0
10	11	Centro Est Arquit	2569.949999999999818	0101000020E610000036C82423678950C0F06DFAB31F6531C0
11	12	Centro Sistemas	2568.5100000000000218	0101000020E61000007E1D3867448950C0CC2A6C06B86431C0
12	14	Comedor	2582.929999999999836	0101000020E6100000E0D74812848950C098F90E7EE26431C0
13	15	Decanato Y Direccion Academica	2583.1700000000000073	0101000020E61000001233FB3C468950C0800C1D3BA86431C0

Figura 3.8: Herramienta gráfica de PostgreSQL (*pgAdmin*).

Fuente: Elaboración propia

En la figura 3.8 se puede observar que la columna *Elevation* contiene datos que el GPS Garmin Nuvi 1300 genera al momento de guardar un punto, en el presente caso es irrelevante.

3.2.2.2. Las Rutas

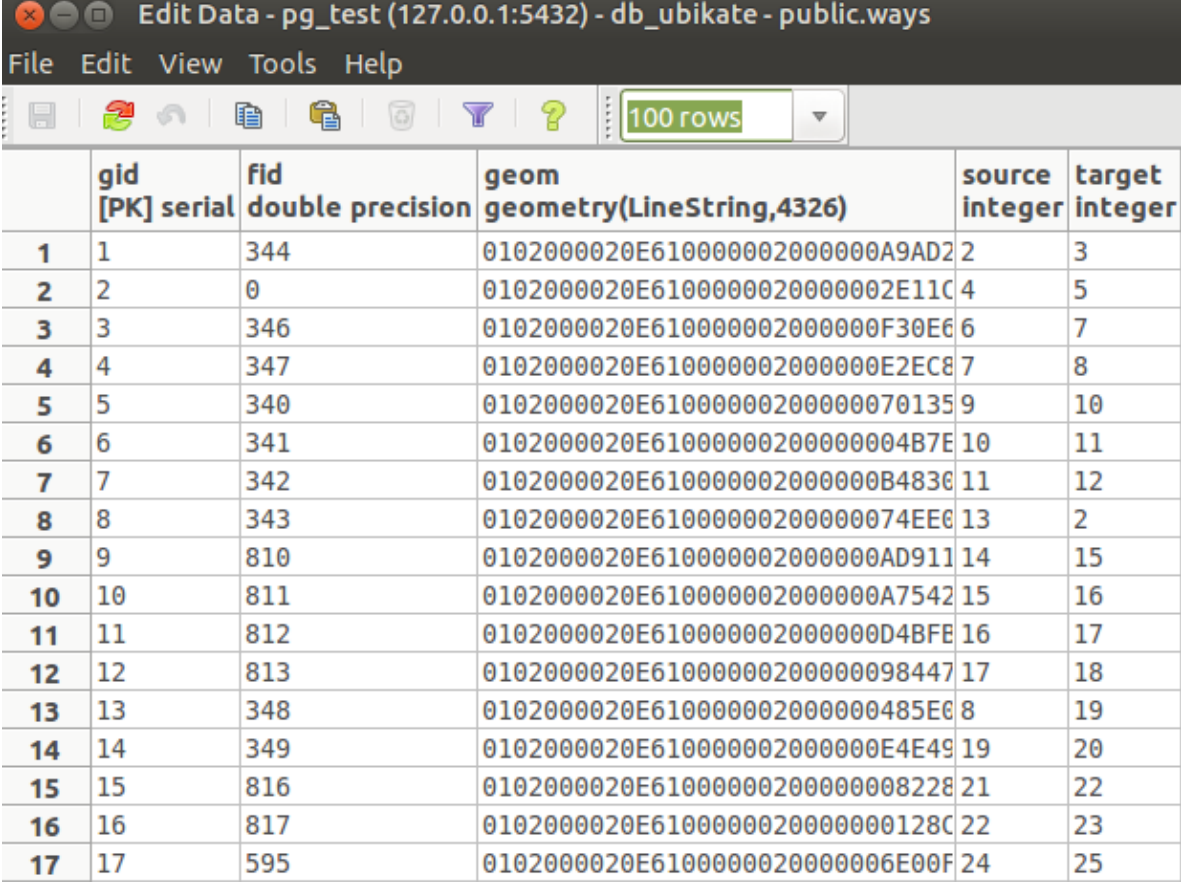
Después de generar un archivo shapefile en la sección 3.1, se procede a poblar la base de datos del mismo modo que se hizo con la información de los lugares. Para las rutas se genera una tabla nombrada *ways*.

Posteriormente se procede a preparar la tabla *ways* para que soporte las funciones instaladas por *pgRouting*. Es necesario ejecutar un query propio de *pgRouting* el cual 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* existe una aplicación que se lo conoce como *topología de red*. La *topología de red* representa las relaciones entre segmentos en una red lineal o una colección de segmentos de línea. [29]

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 3.9, las columnas *source* y *target* son pobladas con el análisis topológico y en la figura 3.10, se puede observar que la tabla *ways_vertices_pgr* es creada enteramente en la ejecución de la instrucción.



	gid [PK] serial	fid double precision	geom geometry(LineString,4326)	source integer	target integer
1	1	344	0102000020E610000002000000A9AD2	2	3
2	2	0	0102000020E6100000020000002E11C	4	5
3	3	346	0102000020E610000002000000F30E6	6	7
4	4	347	0102000020E610000002000000E2EC8	7	8
5	5	340	0102000020E61000000200000070135	9	10
6	6	341	0102000020E61000000200000004B7E	10	11
7	7	342	0102000020E610000002000000B4836	11	12
8	8	343	0102000020E61000000200000074EE6	13	2
9	9	810	0102000020E610000002000000AD911	14	15
10	10	811	0102000020E610000002000000A7542	15	16
11	11	812	0102000020E610000002000000D4BFE	16	17
12	12	813	0102000020E61000000200000098447	17	18
13	13	348	0102000020E610000002000000485E6	8	19
14	14	349	0102000020E610000002000000E4E49	19	20
15	15	816	0102000020E61000000200000008228	21	22
16	16	817	0102000020E6100000020000000128C	22	23
17	17	595	0102000020E6100000020000006E00F	24	25

Figura 3.9: Vista de la tabla *ways*.

Fuente: Elaboración propia

En la figura 3.9 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 que la primera línea tiene en sus extremos, la primera línea o fila está identificada por la columna *gid*.

En la siguiente figura 3.10 se observa la tabla *ways_vertices_pgr* que contiene los vértices creados a partir del análisis de los datos en la tabla *ways*.

The screenshot shows a PostgreSQL query editor window titled "Query - db_ubikate on db_admin@127.0.0.1:5432 *". The SQL Editor contains the query: `SELECT id, cnt, the_geom FROM ways_vertices_pgr;`. The Output pane shows the results of the query in a table format.

	id bigint	cnt integer	the_geom geometry(Point,4326)
1	1		0101000020E6100000F9832C156B8950C0901F762BA46431C0
2	2		0101000020E6100000A9AD269A468950C0689A75947F6431C0
3	3		0101000020E61000009C8F8A09478950C00C364BE8806431C0
4	4		0101000020E61000002E11CD72918950C0F71D633CA16431C0
5	5		0101000020E6100000A259CA818B8950C04B32FA98AC6431C0
6	6		0101000020E6100000F30E6A41478950C073B484B6816431C0
7	7		0101000020E6100000E2EC8FC2478950C00EB58397836431C0
8	8		0101000020E6100000485E0E41488950C06C98E6F2846431C0
9	9		0101000020E610000070135032468950C080B2E2457B6431C0
10	10		0101000020E61000004B7E306468950C04ABB39547A6431C0
11	11		0101000020E6100000B48302F4458950C08268BDC4796431C0
12	12		0101000020E6100000B48302F4458950C0B42A5388796431C0
13	13		0101000020E610000074EE0311468950C02E5190F97D6431C0
14	14		0101000020E6100000AD9113746C8950C0654D7AC3DF6431C0

The status bar at the bottom indicates "OK. Unix Ln 1, Col 49, Ch 49" and "1003 rows. 92 ms".

Figura 3.10: Vista de la tabla *ways_vertices_pgr*.

Fuente: Elaboración propia

Para entender los datos generados 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 del LINESTRING de la fila 1 son los vértices con **id** 2 y 3 respectivamente de la tabla *ways_vertices_pgr*.

Todo el conjunto de vértices y líneas de estas tablas se podría representar con una Matriz de adyacencias, explicada en 2.9.3, y usada en la resolución de la ruta mas corta, mas específicamente con el algoritmo de Dijkstra.

Code 3.2: Algoritmo de Dijkstra implementado en *pgRouting*

```
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);

```

La anterior consulta SQL es una llamada al método *pgr_dijkstra* implementado en *pgRouting* el cual solo puede ser utilizado una vez que la base de datos está preparada para tal efecto, es decir necesitamos que la tabla de rutas *ways* tenga la topología de red y también es necesario los ids de los nodos de destino y origen, *targetId* y *sourceId* respectivamente, estos dos últimos datos son obtenidos por una combinación de acciones ya que los nodos son propios del mapa de rutas y el punto destino es en realidad un lugar el cual está ubicado en otra tabla y el punto origen es donde se encuentra el cliente dentro del campus Universitario, por lo tanto una vez obtenido los punto geo-referenciados del lugar de la tabla *places* y el punto donde se encuentra parado el cliente se obtienen los nodos “más” cercanos a estos puntos, gracias a que *PostGIS* ya lo tiene implementado, encontrar el nodo más cercano es tan fácil como ejecutar la siguiente consulta SQL, donde *lon* y *lat* son la longitud y latitud del lugar.

```

SELECT id
FROM ways_vertices_pgr
ORDER BY the_geom <-> ST_GeometryFromText('POINT(lon lat)', 4326)
LIMIT 1

```

Una vez ejecutado el método *pgr_dijkstra* se obtiene un conjunto de líneas, que son un conjunto de latitudes y longitudes que representa la ruta más corta entre el punto origen y el punto destino, esta informacion realmente no dice nada a la persona que lo lee por lo tanto requiere ser procesada para poder ser consumida desde el navegador, este proceso es llevada a cabo en el servidor y entregada al cliente en formato GeoJSON.

Obtener la coordenada es el primer paso, seguidamente se debe mostrarlo sobre un mapa, en este caso *Open Street Maps*, como se puede apreciar en la figura 3.11, esta interfaz está implementada usando *ember-leaflet*, el cual está principalmente diseñada para ofrecer una mejor experiencia de usuario en celulares smartphones.

```

{{#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}}

```

Figura 3.11: *ember-leaflet* nos ayuda a desplegar un mapa y mostrar un *punto* o *lugar* con un *marcador* y dibuja una línea de color rojo sobre el mapa.



Fuente: Elaboración propia.

3.3. Crear el Frontend

El frontend como ya se mencionó es la vista de la aplicación, es donde el usuario interactúa con la aplicación, en el presente proyecto se implementó usando *Ember.JS*, el cual es un framework de desarrollo JavaScript orientado a crear Aplicaciones de Página Única, esto quiere decir que una vez que se cargue la pagina la primera vez ya no se recargara la pagina como tal, en cambio se actualizará de acuerdo como el usuario interactúa con la aplicación.

Ya que el proyecto se enfocara en la creación de una aplicación web optimizada para su uso en celulares, es primordial que el “look and feel” de la aplicación sea lo más parecido a una aplicación móvil nativa, para lograr este efecto se utilizó *ember-paper*, que ya trae implementado

componentes (botones, listas, links, menús de navegación, etc.) con un comportamiento que emula a una aplicación móvil nativa.

EmberJS está basado en “Convención sobre Configuración”, ya que la gran mayoría de los problemas con que uno se puede enfrentar a la hora de crear una aplicación web ya fueron resueltas por la comunidad de desarrollo, por lo tanto ya se llegó a un patrón o convención de cómo se debería resolver algún determinado problema, *EmberJS* implementa y sugiere soluciones a problemas comunes, para que de esa forma el desarrollo no se enfoque en resolver problemas ya resueltos (configuración) si no en implementar en la lógica del negocio de la aplicación, que al final es lo que le da valor al producto que queremos desarrollar y también para que si en un futuro otro desarrollador empieza a trabajar en el proyecto, sea capaz de entender la estructura y el diseño de la aplicación de forma sencilla. Por lo tanto nos enfocaremos en analizar cómo se resolvieron los problemas: de cómo mostrar los lugares, geolocalizar el lugar sobre un mapa, mostrar la ruta más corta dentro del campus Universitario, manejo de los usuario y como mostrar el reporte de los lugares más visitados.

3.3.1. Mostrar los Lugares

Durante la investigación de esta tarea se encontró *ember-leaflet*, una librería o plugin que contiene las herramientas para poder cargar y usar un servicio de mapas.

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

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

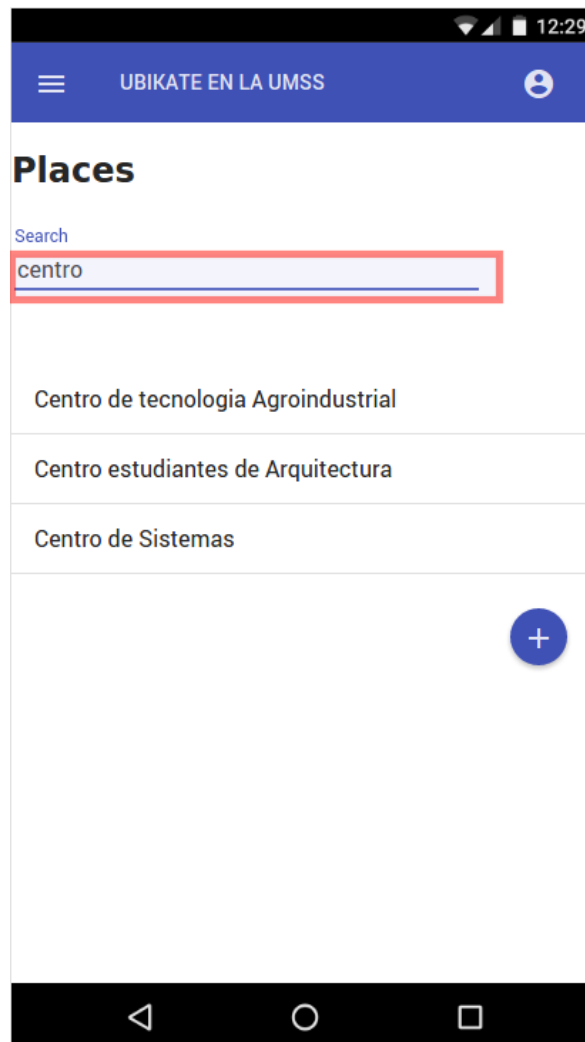
El resultado de la investigación puede apreciar en el marco teórico, en la sección que describe la librería, *ember-leaflet*. 2.6

Figura 3.12: Vista de la lista de Lugares registrados en el sistema.



Fuente: Elaboración propia

Figura 3.13: Vista de la búsqueda de lugares a través de un cajón de búsqueda.



Fuente: Elaboración propia

Para implementar esta funcionalidad del sistema fue necesario utilizar las funcionalidad de Ember JS.

```
{{#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}}
```

3.3.2. Mostrar los Rutas

3.3.3. Manejo de Usuarios

3.4. Informe de los datos

Capítulo 4

Planning Game Ubikate UMSS

Siguiendo la metodología XP, el presente proyecto de grado debe empezar por la primera etapa del Planning Game.

Exploración - Entrega y estimación de las historias de usuario

4.1. Historias de Usuario

Historia US01	Esfuerzo 8 puntos
Descripción	Yo como usuario Deseo registrarme en el sistema Para poder acceder al sistema
Criterios de Aceptación	Quiero ver fácilmente que no estoy registrado Quiero que el registro solamente me pida un nombre de usuario y un password Quiero ver que al estar registrado pueda acceder a los lugares

Cuadro 4.1: Historia de Usuario - US01

Historia US02	Esfuerzo 13 puntos
Descripción	Yo como usuario registrado Deseo buscar un lugar Para conocer su ubicación geográfica
Criterios de Aceptación	Quiero ver una lista de lugares Quiero escribir el nombre de un lugar y ver posibles lugares Quiero encontrar un lugar y poder ver su información

Cuadro 4.2: Historia de Usuario - US02

Historia US03	Esfuerzo 8 puntos
Descripción	Yo como usuario registrado Deseo leer 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

Cuadro 4.3: Historia de Usuario - US03

Historia US04	Esfuerzo 13 puntos
Descripción	Yo como usuario registrado Deseo ver la ruta que necesito seguir Para encontrar el lugar que estoy buscando
Criterios de Aceptación	Deseo ver sobre un mapa un punto referenciado el lugar actual donde me encuentro Deseo ver sobre un mapa un punto referenciado el lugar buscado a donde quiero ir Deseo ver una línea roja que muestre la ruta más corta para llegar de mi ubicación al lugar donde quiero ir Quiero ver un marcador sobre el lugar que estoy buscando con alguna información para asegurarme que es a donde quiero ir

Cuadro 4.4: Historia de Usuario - US04

Historia US05	Esfuerzo 8 puntos
Descripción	Yo como usuario administrador Deseo añadir más lugares Para aumentar las búsquedas de lugares
Criterios de Aceptación	Quiero que sea posible anadir un lugar si no lo encuentro en la lista de lugares Quiero pararme cerca o en el lugar que necesito añadir para geo-referenciarlo Al añadir un lugar necesito ingresar alguna descripción y/o teléfono si fuera necesario

Cuadro 4.5: Historia de Usuario - US05

Historia US06	Esfuerzo 8 puntos
Descripción	Yo como usuario administrador 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 icono 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

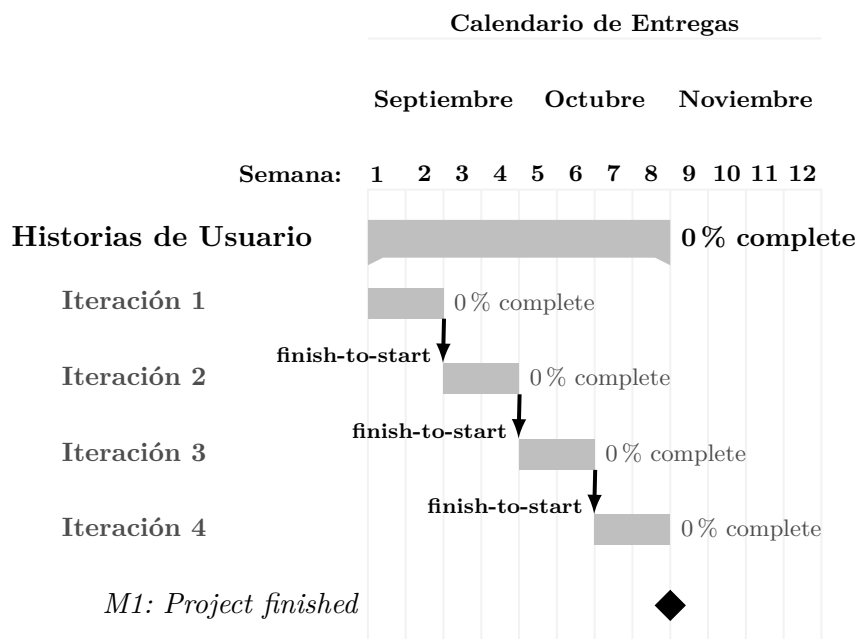
Cuadro 4.6: Historia de Usuario - US06

Historia US07	Esfuerzo 13 puntos
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 que se muestren la cantidad de veces que los usuarios buscan un lugar Quiero obtener esta información y poder guardarla

Cuadro 4.7: Historia de Usuario - US07

4.2. Planeacion de Entregas

A continuación el calendario de entregas del proyecto, el cual de acuerdo de las historias de usuario recogidas se estimó para unas 4 Iteraciones, y cada Iteración de 2 semanas.



Cuadro 4.8: Calendario de Entregas

El orden de implementación y la estimación del equipo de desarrollo para completar las Historias de Usuario se pueden apreciar en la tabla 4.9.

Iteración	Historia de Usuario	Estimación [dias]
Iteración 1	US02	6
	US03	4
Iteración 2	US04	10
Iteración 3	US05	5
	US06	5
Iteración 4	US01	4
	US07	6

Cuadro 4.9: Estimación de la implementación de las Historias de Usuario.

Capítulo 5

Iteration Planning Game Ubikate UMSS

5.1. Iteración 1

Para la primera iteración se implementaran las historias de usuario que tengan más relevancia dentro de la lógica de negocios para el cliente, generalmente son las que tienen mayor impacto en el sistema a desarrollar.

5.1.1. Iteration Planning Meeting

Tomando en cuenta que el equipo de desarrollo está compuesto solo por mi persona, para el desarrollo del presente proyecto de grado la fase de Exploración y Planeación se procedió a realizarlas en la misma fase.

5.1.1.1. Exploración y Planeación

Para la primera iteración se llevó a cabo una reunión para determinar las historias de usuario que se implementaran, y de acuerdo del impacto en el producto se determinó que las historias de usuario 2 y 3 serán las primeras en implementarse.

Posteriormente como tarea del desarrollador se procede a dividir las historias de usuario en Tareas de Ingeniería, en la tabla se determinaron las Tareas pertenecientes a la historia de usuario 2, dentro lo que es la planeación se debe repartir las tareas entre los desarrolladores, pero ya que el equipo de desarrollo se traduce a mi persona, todas las tareas recaen sobre mi responsabilidad, como parte de la planeación es necesario estimar las tareas, para lo cual se presenta la tabla 5.1.

5.1.2. Tareas del US02

Código	Tarea	Estimación [días]
RF001	Crear un archivo shapefile con información inicial de lugares principales dentro el campus de la UMSS.	1
RF002	Crear una base de datos que pueda manejar información geoespacial.	1
RF003	Popular la base de datos creada en RF002 con la información de RF001.	0.5
RF004	El usuario puede ver una lista con los lugares creados.	2
RF005	El usuario deberá poder ingresar el nombre de un lugar para filtrar los lugares existentes.	0.5
RF006	El usuario deberá poder ver la información de un lugar al hacer tap sobre el nombre del lugar en la lista.	1
TS001	Crear pruebas de funcionalidad del US02.	1
Total:		7

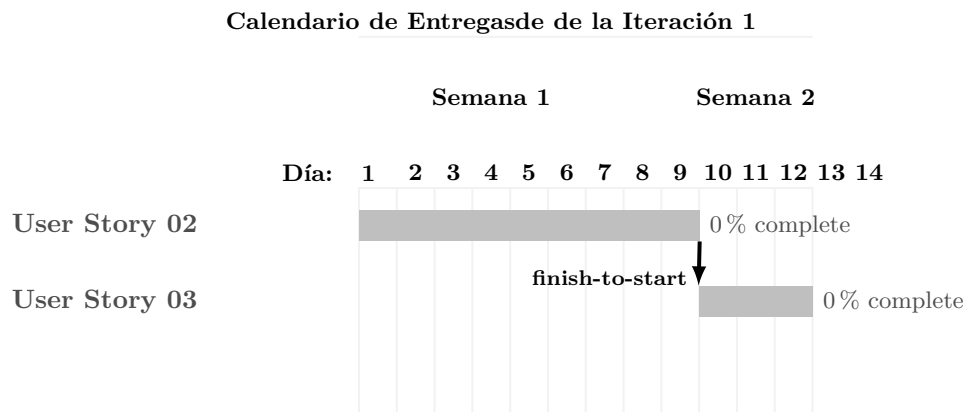
Cuadro 5.1: Tareas de la US02

5.1.3. Tareas del US03

Código	Tarea	Estimación [días]
RF007	El usuario puede ver la descripción del lugar.	0.5
RF008	El usuario puede ver el teléfono del lugar.	0.5
RF009	El usuario puede ver en qué piso se encuentra el lugar.	0.5
RF010	El usuario puede ver una imagen del lugar.	1
TS002	Crear pruebas de funcionalidad del US03.	0.5
Total:		3

Cuadro 5.2: Tareas de la US03

5.1.4. Calendario de Entregas



Cuadro 5.3: Calendario de Entregas de la Iteración 1

5.1.5. Implementación

A continuación se detalla los resultados de la implementación de cada tarea asignada.

5.1.5.1. RF001

5.1.5.2. RF002

5.1.5.3. RF003

5.1.5.4. RF004

5.1.5.5. RF008

5.1.5.6. RF009

5.1.5.7. RF010

El hecho de visualizar las imagenes dentro de la aplicacion se tiene q pensar en donde se van a guardar las imagenes Para realizar esta tarea se hizo uso de Cloudinary

5.1.5.8. TS001

Pruebas funcionales

5.1.5.9. TS002

Pruebas funcionales

5.1.6. Registrar el Avance

5.1.7. Verificación

5.2. Iteración 2

Después de que finalizó la primera iteración y ya que todas las pruebas pasaron exitosamente, se continúa con la segunda iteración.

5.2.1. Iteration Planning Meeting

Al igual que la primera iteración, las fases de exploración y planeación se realizan al mismo tiempo, en el sentido que no es necesario repartir las tareas resultantes de la exploración, por lo tanto al mismo tiempo en que se determinan las tareas se puede realizar la estimación de las mismas.

5.2.1.1. Exploración y Planeación

Para la segunda iteración se toman en cuenta todas las historias de usuario restantes, y de acuerdo al criterio de escoger las siguientes historias más relevantes y de mayor valor para el producto, se escogió la historia de usuario 4.

Como parte de la fase de Exploración se toma la historia de usuario 4 y la dividimos en las Tareas de Ingeniería, las cuales serán trabajadas en la fase de la Implementación.

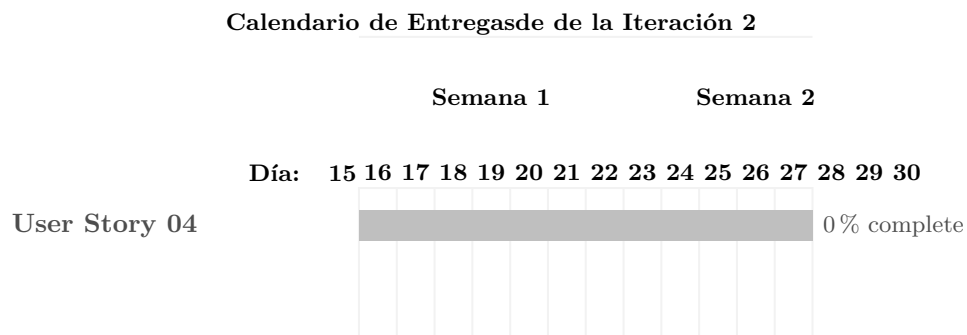
En la siguiente tabla se especificaran las tareas correspondientes a la historia de usuario 4 5.4.

5.2.2. Tareas del US04

Código	Tarea	Estimación [días]
RF011	Crear un archivo shapefile con información inicial de lugares principales dentro el campus de la UMSS.	2
RF012	Preparar la base de datos para manejar información geográfica de rutas.	1
RF013	Investigar e instalar una herramienta que permita usar un servicio de mapas.	1
RF014	El usuario puede ver un mapa usando un servicio del campus de la UMSS.	0.5
RF015	El usuario puede ver un marcador sobre el lugar.	0.5
RF016	El marcador tiene información básica del lugar, nombre, piso.	0.5
RF017	El usuario puede ver un marcador mostrando el lugar actual donde se encuentra (el usuario).	0.5
RF018	Desarrollar un módulo que encuentra la ruta más corta usando la base de datos con información geográfica ruteable de RF012.	2
RF019	El usuario puede ver una línea roja que une el marcador de la posición del usuario con el marcador del lugar.	1
TS003	Crear pruebas de funcionalidad del US04.	1
Total:		10

Cuadro 5.4: Tareas del US04

5.2.3. Calendario de Entregas



Cuadro 5.5: Calendario de Entregas de la Iteración 2

5.2.4. Implementación

Durante esta fase es donde se implementaran las tareas especificadas en la tabla 5.4.

5.2.4.1. RF011

5.2.4.2. RF012

5.2.4.3. RF013

5.2.4.4. RF014

Para completar esta tarea se hizo uso de la herramienta *ember-leaflet*, con la cual se puede desplegar un mapa en el browser y optimizada para dispositivos moviles.

```
{{#leaflet-map lat=lat lng=lng zoom=zoom}}
  {{tile-layer
    url="http://{s}.tile.openstreetmap.fr/hot/{z}/{x}/{y}.png"
  }}
{{/leaflet-map}}
```

Con la anterior instrucción se accede al servicio de *Open Street Maps*, de la cual obtenemos los datos necesarios para renderizar un mapa en el browser. Los atributos de *lat* y *lng* se acceden de la capa del controlador de la aplicacion, son la latitud y longitud respectivamente, la conbinacion de ambos datos es la locacion donde se va a ubicar el renderizado del mapa.

5.2.4.5. RF015

Para completar esta tarea se continuó usando la librería *ember-leaflet*, la cual permite que con la siguiente instrucción se despliegue un marcador sobre el mapa renderizado del API de *Open Street Maps*.

```
{{#marker-layer location=userLocation}} {{/marker-layer}}
```

El resultado de la tarea se puede observar en la figura 5.1.

5.2.4.6. RF016

Para poder mostrar la informacion del lugar sobre el marcador creado en RF015 se hizo uso de la librería *ember-leaflet*, al igual que dicha tarea, solo se necesito de una instruccion para poder desplegar la informacion necesaria.

```
{{#marker-layer location=location}}
```

```

h3>{{model.name}}</h3>
{{model.description}}
<strong>telf:</strong> {{model.phone}}
<strong>piso#</strong> {{model.level}}
{{/marker-layer}}

```

En la figura 5.1 se puede apreciar el marcador con la información desplegada del lugar “Baquita”.

Figura 5.1: Tooltip con la información de un lugar.



Fuente: Elaboración propia.

5.2.4.7. RF017

Para encontrar la locación del usuario se uso el API de geolocalización propio de HTML5, que en un smartphphone puede acceder y usar los recursos nativos de un smartphphone, es necesaria la aceptacion del usuario mediante un mensaje que el navegador despliega, la locacion es encontrada mediante la triangulacion de Coordenadas por GPS (el mas exacto a la hora de encontrar la locacion del dispositivo), Wi-Fi, GSM o CDMA. Solo es necesaria la ejecucion de la siguiente linea para ponder obetener la posición actual del usuario usando el API de geolocalización de HTML5.

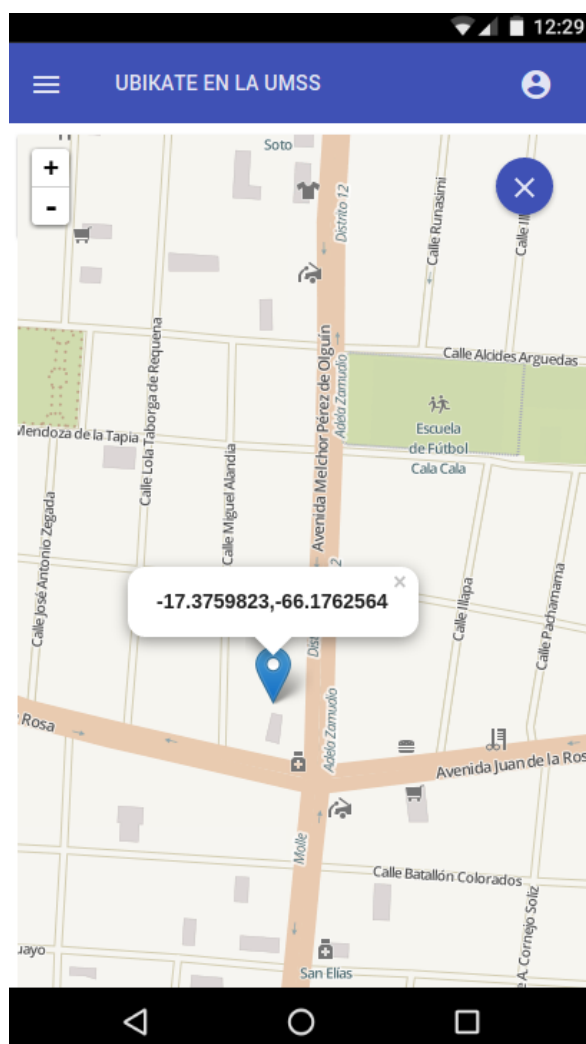
```

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

```

La *latitud* y *longitud* obtenidas es fácilmente trasladado al mapa usando *ember-leaflet* mediante un marcador, como se puede apreciar en la siguiente figura.

Figura 5.2: Tooltip con la latitud y longitud de la posición actual del usuario.



Fuente: Elaboración propia.

5.2.4.8. RF018

Durante esta tarea se investigó la mejor forma de encontrar la ruta más corta y se llegó a la conclusión de usar la combinación de *Postgres + Postgis + pgRouting*, esta investigación se puede apreciar en el marco teórico.

Para hallar la ruta más corta se necesita usar las características de la base de datos para poder analizar los datos geográficos almacenados en la Tarea RF012, como el análisis que se requiere hacer es caro o sea el costo de procesador para realizar los calculos necesarios es elevado, lo mas recomendable es que este trabajo sea realizado en el backend de la aplicacion por la base de datos.

Tambien hay que tomar en cuenta que la tierra no es plana y las líneas que en un mapa parecen líneas rectas, realmente no son rectas, ya que el planeta Tierra es un *esferoide oblato*¹ por lo que las líneas en apariencia rectas tienen la curvatura natural del planeta Tierra. En distancias largas esto tiene un gran impacto al manejar o utilizar mapas proyectados, pero tambien es cierto que para una área pequeña como es el campus de la Universidad de San Simón este problema no tiene un gran impacto pero no está demás en tomar en cuenta esta característica del análisis de datos geoespaciales, como se explico en el capitulo ??, para el presente proyecto se usara el proyeccion *SRID 3857*.

Una vez que se tienen en cuanta estas variables es necesaria la resolucio del problema de la ruta más corta, *pgRouting* tiene varios métodos implementados para el analisis de datos geo-espaciales en la resolucio de este problema, para el presente proyecto se usara el algoritmo de *Dijkstra*, explicado en el capítulo ??.

Tomando en cuenta los conceptos aprendidos y las herramientas investigadas es que se desarrollo el modulo que encuentra la ruta mas corta.

5.2.4.9. RF019

Como resultado de la tarea RF018 se tiene un conjunto de datos en formato de latitud y longitud que conforman líneas, las cuales representan la ruta más corta, pero al final es solo un montón de números, útiles pero para el usuario esta información es difícil de procesar, el usuario necesita información que sea fácil de entender y no existe mejor herramienta disponible para esta tarea que mostrar la *ruta* de forma visual, esto quiere decir que se necesita mostrar la ruta sobre un *mapa*, en la aplicación se usará *ember-leaflet* para desplegar el mapa ofrecido por los servicios de OpenStreetMaps y también para mostrar ruta más corta mediante una línea de color rojo.

Para resolver esta tarea se creó un servicio API usando ExpressJS, la cual se encarga obtener la información extraída de la base de datos y transformarla en un objeto JSON (GeoJSON), este objeto contiene la información geoespacial necesaria para “dibujar” la línea roja entre 2 puntos georeferenciados, uno de los cuales es el lugar al que se quiere llegar y el otro es la ubicación actual del usuario.

```
ENV.APP.API_HOST + '/api/v1/ways/route/' + sourceData.id + '/' + targetData.id;

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
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           %             %             Dload  Upload  Total  Spent    Left     Speed
100  3911  100  3911    0     0  161k      0  --:--:-- --:--:-- --:--:-- 166k
{
```

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

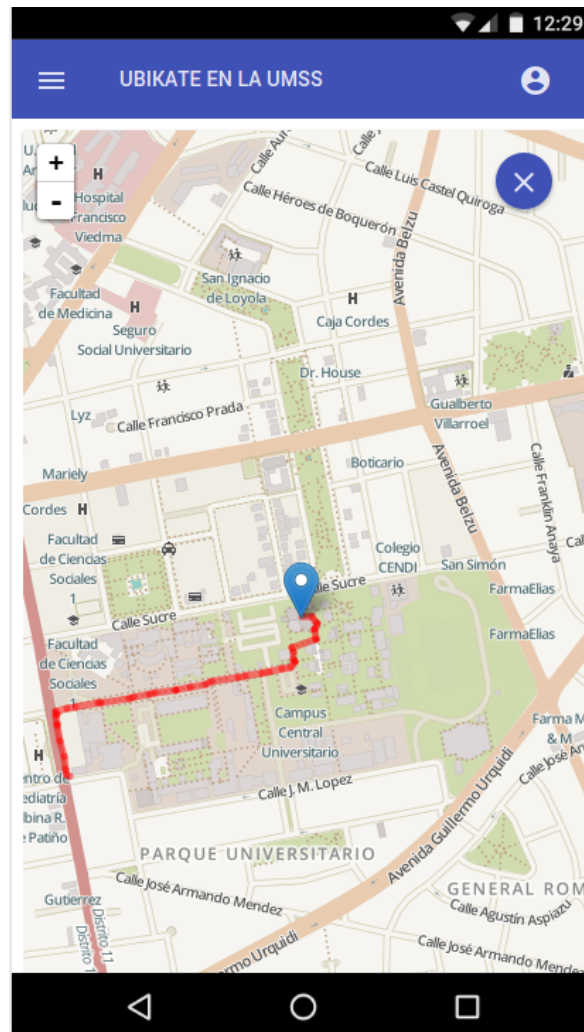
```

"features": [
  {
    "geometry": {
      "coordinates": [
        [
          -66.1467397848201,
          -17.3935321732846
        ],
        [
          -66.1467190789842,
          -17.3935294725234
        ]
      ],
      "type": "LineString"
    },
    "type": "Feature"
  },

```

Y se puede observar en el mapa una línea roja que representa la ruta más corta entre el punto donde se encuentra el usuario y el punto del lugar a buscar.

Figura 5.3: Ruta más corta dibujada con una línea roja.



Fuente: Elaboración propia.

5.2.4.10. TS004

Pruebas funcionales

5.2.5. Registrar el Avance

5.2.6. Verificación

5.3. Iteración 3

Al igual que al principio de la segunda iteración, se esperan los resultados de las pruebas realizadas para poder empezar con la planificación de la tercera iteración.

5.3.1. Iteration Planning Meeting

Los resultados de las pruebas realizadas se analizan para determinar si los criterios de aceptación, de las historias de usuario trabajadas en la segunda iteración, se cumplen para poder continuar con las historias que continúan sin ser desarrolladas, en caso que las pruebas fallen, es necesario continuar con la implementación de las historias inconclusas.

En el caso del presente proyecto, las pruebas pasaron exitosamente y se aceptaron los criterios de aceptación de las historias de usuario trabajadas, por lo tanto se procede con la primera fase del “Iteration Planning”.

5.3.2. Exploración y Planeación

Esta fase generalmente se realiza en 2 pasos pero será realizada al mismo tiempo, ya que en la exploración se definen las tareas a realizar y en la planeación se asigna estas tareas al equipo de desarrollo, el cual tiene que estimar las tareas, pero como el equipo de desarrollo está compuesto por mi persona, puedo definir las tareas y asignarles una estimación en el mismo paso.

Para la Iteración 3, se trabajarán las historias de usuario 5 y 6, de las cuales serán definidas sus tareas de desarrollo en las siguientes tablas.

5.3.3. Tareas del US05

Código	Tarea	Estimación [días]
RF020	El usuario podrá ver un link hacia el formulario para añadir más lugares desde la lista de lugares existentes.	0.5
RF021	El usuario podrá ver un formulario para añadir un lugar con información básica. por ejemplo, el nombre del lugar, descripción, teléfono.	1
RF022	El usuario deberá estar cerca del lugar que desea añadir para poder georeferenciarlo.	1
RF023	Un usuario no-administrador no debería poder ver el formulario para añadir lugares.	1
TS004	Crear pruebas de funcionalidad del US05.	0.5
Total:		4

Cuadro 5.6: Tareas del US05

5.3.4. Tareas del US06

Código	Tarea	Estimación [días]
RF024	El usuario podrá ver un link dentro del perfil del lugar hacia el formulario para editar el lugar.	0.5
RF025	El usuario debería poder ver la información actual del lugar en el formulario para editar el lugar.	1
RF026	El usuario no-administrador no debería poder ver el link hacia el formulario para editar el lugar.	1
TS005	Crear pruebas de funcionalidad del US06.	0.5
Total:		3

Cuadro 5.7: Tareas del US06

Bibliografía

- [1] <https://www.scrumalliance.org/why-scrum>
- [2] <http://www.extremeprogramming.org/>
- [3] Beck, K., Extreme Programming Explained: Embrace Change, Addison Wesley, 1999.
- [4] ftp://ftp.software.ibm.com/la/documents/gb/commons/27754_IBM_WP_Native_Web_or_hybrid_2846853.pdf
- [5] <http://brewhouse.io/blog/2015/05/13/emberjs-an-antidote-to-your-hype-fatigue.html>
- [6] <http://si.ua.es/es/documentacion/asp-net-mvc-3/1-dia/modelo-vista-controlador-mvc.html>
- [7] Hoffer, Jeffery A., George, Joey F., Valacich, Joseph S. Modern Systems Analysis and Design. New Jersey: Prentice Hall, 2005.
- [8] Steve Hayes, Martin Andrews, An Introduction to Agile Methods, <http://csis.pace.edu/~marchese/CS616/Agile/IntroToAgileMethods.pdf>
- [9] <http://www.extremeprogramming.org/donwells.html>
- [10] https://prezi.com/axxujrg_tyxw/xp/
- [11] <http://www.cyta.com.ar/ta0502/v5n2a1.htm>
- [12] Thomas Dudziak, eXtreme Programming An Overview, http://csis.pace.edu/~marchese/CS616/Agile/XP/XP_Overview.pdf
- [13] <https://sites.google.com/a/xtec.cat/curs-qr-ar-public-copia/modul-2/2-realidad-aumentada-georeferenciada-espira>
- [14] <http://openstreetmapdata.com/info/projections>
- [15] <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>
- [16] Applications Programming in Smalltalk-80 (TM): How to use Model-View-Controller (MVC) by Steve Burbeck, Ph.D http://www.dgp.toronto.edu/~dwigdor/teaching/csc2524/2012_F/papers/mvc.pdf
- [17] <https://www.w3.org/2002/ws/Activity>

- [18] Rafael Navarro Marset. ELP-DSIC-UPV Modelado, Diseño e Implementación de Servicios Web 2006-07 <http://users.dsic.upv.es/~rnavarro/NewWeb/docs/RestVsWebServices.pdf>
- [19] Uniform Resource Identifier (URI): Generic Syntax <http://www.ietf.org/rfc/rfc3986.txt>
- [20] https://es.wikipedia.org/wiki/Single-page_application
- [21] Mike Wasson, ASP.NET - Single-Page Applications: Build Modern, Responsive Web Apps with ASP.NET <https://msdn.microsoft.com/en-us/magazine/dn463786.aspx>
- [22] <http://kartoweb.itc.nl/geometrics/coordinate%20systems/coordsys.html>
- [23] Introduction to Spatial Coordinate Systems: Flat Maps for a Round Planet, [https://technet.microsoft.com/en-us/library/cc749633\(v=sql.100\).aspx](https://technet.microsoft.com/en-us/library/cc749633(v=sql.100).aspx)
- [24] University of West Hungary, Faculty of Geoinformatics, Ferenc Végső Data acquisition and integration 1. http://w3.geo.info.hu/~ng/tamop_jegyzet/pdf/DAI1.pdf
- [25] <http://msdn.microsoft.com/es-es/library/bb964707.aspx>
- [26] Introducción a la Teoría De Grafos, Alfredo Caicedo Barrero, ISBN: 978-958-99325-7-5
- [27] <https://www.esri.com/library/whitepapers/pdfs/shapefile.pdf> - Pag7
- [28] <http://desktop.arcgis.com/es/arcmap/10.3/manage-data/shapefiles/what-is-a-shapefile.htm>
- [29] Spatial Relationships in GIS – Geospatial Topology Basics, Landon Blake, OSGeo Journal, ISSN 1994-1897