UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Representación geográfica y visualización cartográfica con ayuda de programación orientada a objetos

TESIS

Que para obtener el título de

Ingeniero Geomático

P R E S E N T A

Miguel Iván Hernández Martínez

DIRECTORA DE TESIS

M.C.T. María Elena Osorio Tai

Ciudad Universitaria, Cd. Mx., 2024

Agradecimientos y dedicatorias.

Gaby gracias por tu compañía en las muy buenas, las buenas, las malas y las horribles, desde que regresaste a mi vida siempre me has motivado a hacer de mi una mejor pareja, esposo y padre, no alcanzan las palabras para poder expresarte lo mucho que te amo y solo te diré “Ya se que puedo ofrecerte que nadie más puede, total y completa dependencia, mírame llevo un día fuera de la casa y ya estoy todo zarrapastroso”. Te amo para siempre mi corazón.

Gaby bebé, siempre te lo digo y siempre te lo voy a decir, eres el motor que siempre me impulsa y cada paso que de lo voy a hacer pensando en ti siempre, recuerda que aunque no puedas verme siempre voy a estar contigo. Te amo mi nena hermosa. “Do it for her”

Papá, tu sabiduría, guía y apoyo siempre me ha hecho un buen ciudadano y siempre trato de ser un gran ser humano debido siguiendo las enseñanzas que siempre me das.

Mamá, tu amor, confianza y cariño, además de tus porras siempre me ayudan a seguir adelante incluso cuando siento que no puedo más siempre estás ahí para ayudarme a seguir adelante.

Mis hermanos, siempre me apoyan y toleran en todo momento, no queda más que decirles que los quiero mucho y siempre voy a estar ahí.

Familia Maldonado, todos ustedes siempre son un ejemplo a seguir, gracias por siempre estar ahí para mí de manera incondicional.

Este trabajo es para y por todos ustedes gracias por jamás perder la fe en mi

A la Ing. Beatriz Cervantes García y la M.C María Elena Osorio Tai por todo su apoyo para el desarrollo del proyecto, les debo mucho no creo poder ser capaz de poder pagarles nunca mil gracias.

“Siempre dudé de la existencia de dios, ahora sé que existe soy yo” Homer J Simpson.

Índice

Introducción.

La cartografía desempeña una funcionalidad importante a lo largo de la vida civil y militar, la función principal es representar de manera gráfica puntos de interés en un mapa para así poder obtener información sin necesidad de estar en el lugar, esta ha cambiado con el avance de la tecnología, tanto para mejorar la precisión como la disponibilidad facilitando la disponibilidad que tienen los usuarios para poder utilizarla para características específicas.

A lo largo de este documento se describe la forma de poder realizar cartografía tanto estática como dinámica utilizando la programación como herramienta para facilitar la visualización de los datos.

El proyecto se divide en dos fases, la primera fase fue desarrollada en PHP y Java script para la parte web y Java y XML para la aplicación, esta fase no está actualizada, por lo que puede ser mejorada con otro tipo de tecnologías más actualizadas, sin embargo, en su momento fue lo más eficiente para el desarrollo de la aplicación y la segunda fase incluye la actualización del documento integrando nuevas tecnologías Go, ReactJs para web y para móvil Kotlin y Jetpack Compose.

Algunas de las herramientas utilizadas son con costo, sin embargo, la mayoría de ellas son desarrolladas de manera propia.

Los objetivos buscados con el desarrollo de esta tesis son mostrar las diferencias y ventajas de utilizar los diferentes tipos de visualizadores, así como mostrar algunas de las maneras de desarrollar estas herramientas.

Este trabajo surge de la necesidad de mostrar información en dispositivos móviles, para poder verificar información precargada de manera sencilla y accesible en campo, aunque esta funcionaba solo para visualización por lo cual fue modificada debido a que podría ser mejorada y adaptada para cumplir las características principales de los SIGs las cuales son:

- Accesibilidad.
- Claridad.

- Precisión.

Esta tesis mostrará métodos de representación cartográfica dinámica y colaborativa, donde el objetivo principal surge de la necesidad de representar dentro de dispositivos móviles información relacionada la localización de la información.

I Antecedentes

I.1 Primeros Visualizadores de Mapas

Cuando se habla de visualizadores cartográficos o geográficos no se puede únicamente mencionar a los visualizadores de última generación; antes de adentrarnos en los temas de visualizadores modernos se citarán a los primeros visualizadores en la historia; los mapas, como lo dijo Harley en su artículo: Los mapas y el desarrollo de la historia de la cartografía, "… los mapas son herramientas fundamentales ayudando en el pensamiento humano a dar sentido a su universo en varias escalas." (Harley, 1987, p 1).

El primer cartograma del que se tienen registros es la tablilla de Babilonia, esta data del año 2800 A.C, sin embargo, no se puede asegurar que este sea la primera representación puesto que el humano es un cartógrafo por naturaleza que posee un sentido que lo lleva a tratar de ubicarse dentro de su entorno, lo tomaremos como punto de partida debido a que es el primer registro impreso. (INEGI, 2006, p4).

En la antigua Grecia se propone la teoría de que la tierra es esférica y se desarrollan sistemas de coordenadas curvilíneos.

Mientras que los romanos habían alcanzado su apogeo tanto cultural como económico, su cartografía en esencia se caracteriza por tener a Roma al centro de sus mapas y el territorio conquistado o por conquistar en los alrededores. Este ejemplo es representado por la imagen I.1.1 mandado a dibujar por el emperador Augusto y diseñado por Agrippa quien tenía grandes conocimientos de geografía.

Imagen I.1.1 *Orbis Terrarum, N.-S.*

En la edad media se publicaron los principios geográficos de Ptolomeo, quien establece los principios a seguir durante la construcción de cartas topográficas, sin embargo, debido a los grandes descubrimientos geográficos surgidos durante esta época, la cartografía "… llegará a tener una imagen más exacta y verdadera de la extensión de la tierra. Con ello, la cartografía general ganará en exactitud y precisión." (Errázuriz, González, Henríquez & Rioseco, 1988, p 17).

Estos descubrimientos no solo se ven reflejados en la cartografía también se aprecian en la forma de dibujar, durante esta época se pasa de tener mapas T en O de la Edad Media (ver Imagen I.2) a tener mapas con gran representación como lo es el ejemplo de la Carta de navegación de portulano (ver Imagen I.3).

Imagen I.1.2 *Mapa T en O de la edad media.*

Imagen I.1.3 *Carta de navegación de portulano.*

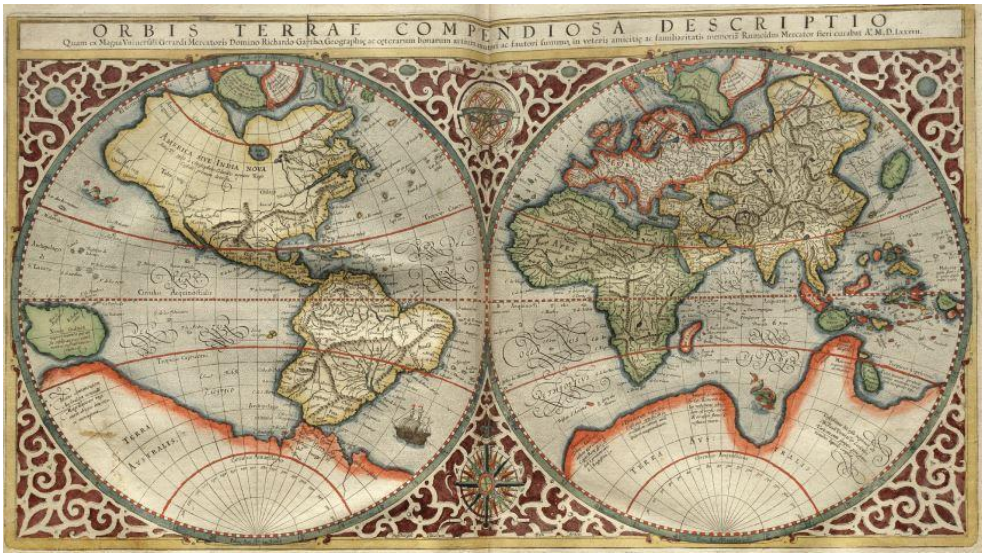
Al terminar la edad media, durante el Renacimiento se desarrolla cartografía de mayor precisión debido a que durante esta época hay descubrimientos importantes en el área de la topografía como: el método de la intersección y la información sobre la triangulación, además de la invención de la brújula.

Durante este periodo se tiene un conocimiento más amplio de la geografía más allá de solo el conocimiento de los 3 viejos continentes, ponen en marcha las expediciones hacia las colonias americanas españolas, inglesas y francesas.

Gerhard Kremer Mercator hijo, publica un año después de la muerte de su padre “Atlas sive cosmographicae meditationes de fabrica mundi et fabricati figura”, en este atlas se puede apreciar que incluye distintos mapas que fueron trazados con los nuevos métodos

en uso antes mencionados, aunque ya se tenía conocimiento e incluso había colonias en américa, no se tiene registro en este atlas de cartografía individual. En esta compilación hay dos apariciones de América, la primera es en el “Orbis Terrae Compendiosa Descriptio” como se ilustra en la imagen I.4 y la segunda es en el mapa “America” Imagen I.5.

Imagen I.1.4 *Orbis Terrae Compendiosa Descriptio.*



Con el paso del tiempo la cartografía fue desarrollando nuevos métodos de dibujo debido a nuevos mecanismos de medición, orientación y captura de información. Como ejemplo tenemos el desarrollo de la fotogrametría actual, esta forma de realizar levantamientos topográficos y cartografía surge por primera vez en el año 1893 por el alemán Albercht Meydenbauer, aunque los procesos de desarrollo de esta técnica surgieron antes y fueron utilizados por quien es considerado como “El padre de la fotogrametría” Aimé Laussedat.

Imagen I.1.5 *America.*

Las técnicas fotogramétricas desde finales del siglo XIX junto con los instrumentos y nuevas técnicas topográficas ayudaron a realizar mapas más realistas y precisos, estas técnicas de apoyo aéreo y terrestre son los predecesores de la teledetección o percepción remota la cual es utilizada actualmente para realizar estudios en regiones de grandes áreas.

Con el desarrollo tecnológico surge un nuevo concepto Sistema de Información Geográfica (SIG), el primer sistema formalmente desarrollado surge por primera vez en Canadá en la década de los 60’s elaborado por Roger Tomlinson, este sistema era una herramienta para el manejo de datos de inventarios y gestión del territorio rural canadiense, debido a este trabajo a Tomlinson se le conoce como el padre de los Sistemas de Información Geográficos.

Actualmente los Sistemas de Información Geográficos son utilizados para diferentes propósitos, gracias al gran avance en la capacidad de almacenamiento de las computadoras la cartografía utilizada con SIGs puede mostrar y almacenar grandes cantidades de información, la cual es utilizada para estudiar diferentes fenómenos.

I.2 Desarrollo y distintas proyecciones cartográficas.

La tierra no es completamente esférica, por lo cual existe la necesidad de crear representaciones aproximadas, la proyección que se utiliza o se realiza depende la necesidad que se tenga de representar el territorio.

Existen tres tipos principales de proyecciones.

1. Azimutal o cenital. Imagen I.2.1
2. Cónica. Imagen I.2.2
3. Cilíndrica. Imagen I.2.3

La proyección azimutal es generada al proyectar la esfera en un plano con un punto de contacto, debido a esto las deformaciones que se verán serán circularmente simétricas, esta representación es comúnmente utilizada para representar los cascos polares debido a que en estas posiciones se genera menos distorsión.

Imagen I.2.1 *Proyección Azimutal.*

Con la proyección cónica se consigue colocar como superficie de contacto un cono, esta proyección presenta menos distorsiones hacia el foco del cono. Es utilizada principalmente cuando se quieren mostrar grandes regiones de latitud y longitud, un ejemplo conocido, es la proyección conforme de Lambert, se utiliza para representar la República Mexicana en un mapa.

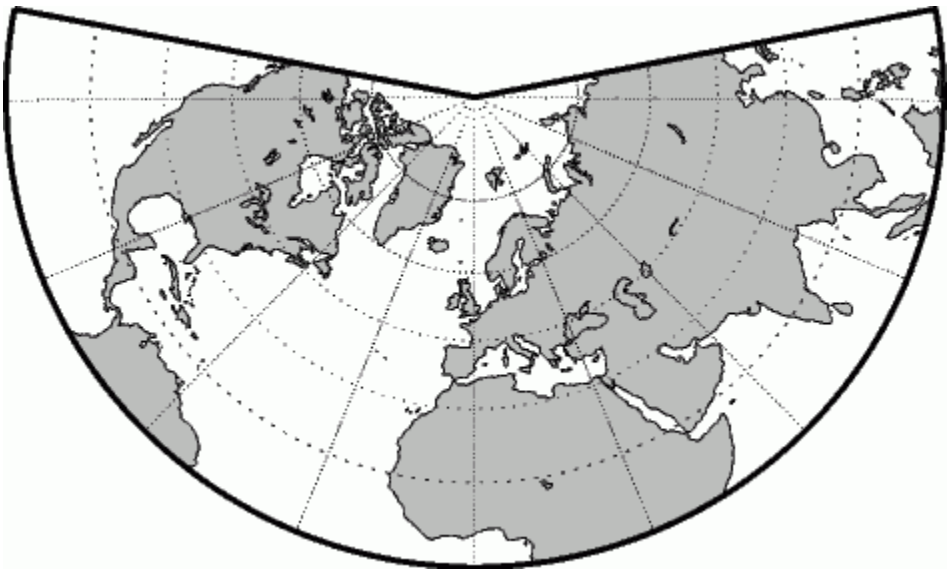


Imagen I.2.2 *Proyección Cónica.*

La proyección cilíndrica, es de las proyecciones más empleadas, ésta se consigue al envolver la forma de la tierra en un cilindro; desarrollada dentro de este tipo se encuentran las proyecciones de Mercator las cuales son las más utilizadas actualmente debido a que estas muestran factores de distorsión pequeños y además cuenta con diferentes tipos de proyecciones.

La proyección de más uso actualmente en México es la Proyección Transversal de Mercator (UTM), esto es debido a que en esta proyección se conserva la escala en cualquier dirección y la distorsión en las áreas es minimizada, por lo tanto, es utilizada en aplicaciones catastrales y estudio de áreas naturales.

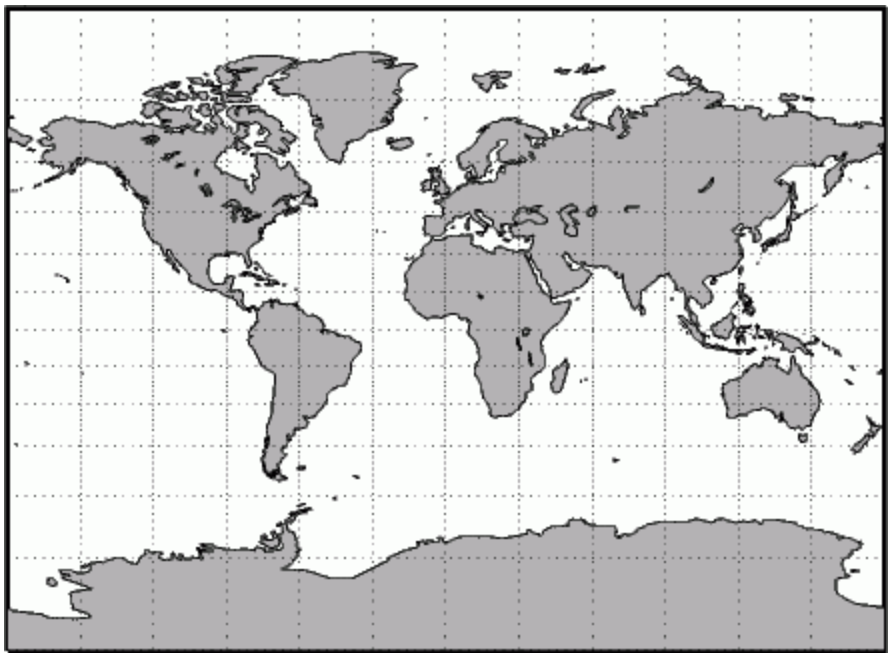


Imagen I.2.3 *Proyección Cilíndrica.*

Un ejemplo de esto son los mapas de Google que actualmente ocupan este tipo de proyecciones lo cual hace que el manejo de los datos sea más sencillo.

A pesar de que estas proyecciones son representaciones precisas de la forma de la tierra es imposible pensar que se reproducen sin distorsiones, la distancia, el área, la escala, etc., se verán afectadas por la representación puesto que como se mencionó anteriormente la tierra no es completamente esférica y las proyecciones son modelos matemáticos adaptados a una forma parcial del geoide.

I.3 Usos Principales de la cartografía antigua y moderna.

El concepto de cartografía es extenso y existen autores que mencionan “Se considera a la cartografía como el arte, ciencia y técnica de hacer mapas y el estudio de estos como documentos científicos y obras de arte.” (INEGI, 2006, p3).

Existen otro tipo de clasificaciones de cartografía como se analiza en la presentación del libro “Cartografía Temática” donde mencionan “…la cartografía se ha vinculado a la

ciencia geográfica, dado que el objetivo de esta requiere de un modo de representación de los fenómenos en el espacio.” (Errazuriz & Gonzales, 1988, p11).

Para estudiar la cartografía podemos hacerla como lo han hecho Errazuriz y Gonzales mediante dos líneas de tendencia: La cartografía temática utilizada en la publicidad y algunos negocios donde la precisión no es prioritario y la Topográfica se refiere a la forma y características del terreno, haciendo énfasis en la precisión de los mapas.

Una definición más de cartografía es la que se da en la 17a. Asamblea General de la Asociación Cartográfica Internacional, la cual tuvo lugar en Barcelona en el año de 1995 donde se determinó: La cartografía es la disciplina que trata sobre la concepción, producción, difusión y estudio de los mapas. Se puede decir que la cartografía es el conjunto de técnicas y ciencias para mostrar un modelo de la tierra y sus fenómenos mediante una representación artística, la cual puede ser subclasificada en cartografía topográfica y cartografía temática, esta subclasificación dará diferentes niveles de importancia a las características cartográficas, en el caso de la cartografía topográfica se le da mayor importancia al aspecto de las ciencias, seguido por las técnicas y terminando por la representación artística, debido a que estos mapas tienen una gran precisión son utilizados para las áreas de la construcción, geodesia, geología, etc.. Por otro lado, la representación temática tiene un mayor interés en el aspecto artístico, seguido por la parte técnica y terminando en la cuestión científica, estos mapas son los utilizados en su mayoría en las ciencias sociales de corto tiempo por ejemplo la publicidad. Como se ejemplifica en el Imagen I.3.1

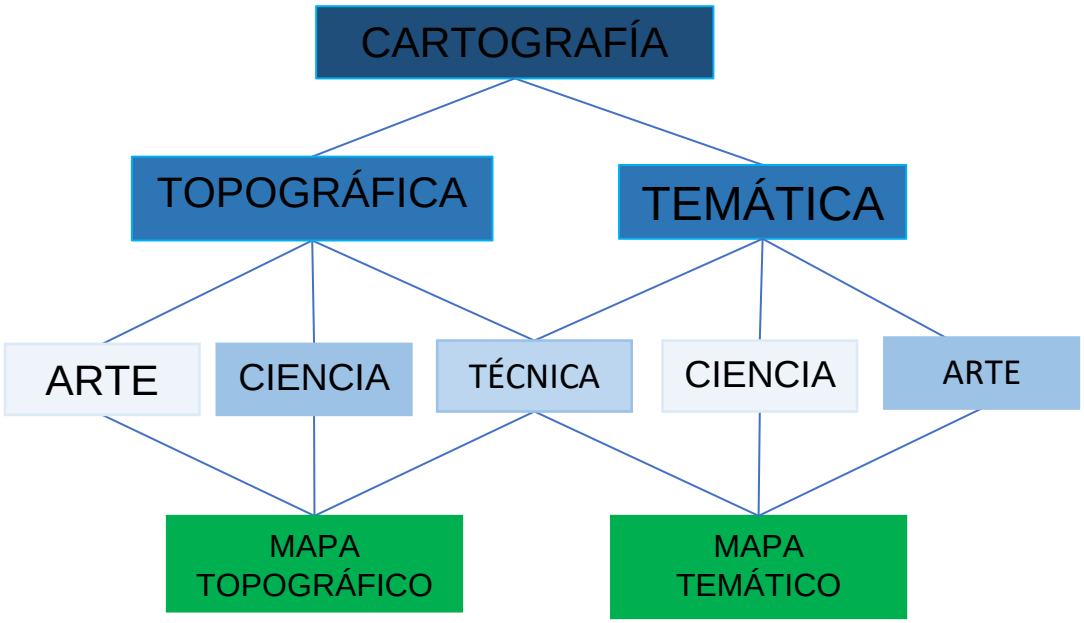


Imagen I.3.1 *Diagrama de ilustración de las cartografía.*

La cartografía a su vez es clasificada en dos tipos, cartografía estática y cartografía dinámica, la primera se refiere a un evento ocurrido en un tiempo definido, por otro lado, la cartografía dinámica es la que describe un evento en un tiempo "t" determinado, la representación en gráficas se encuentra en la Imagen I.3.2 y Tabla I.3.3

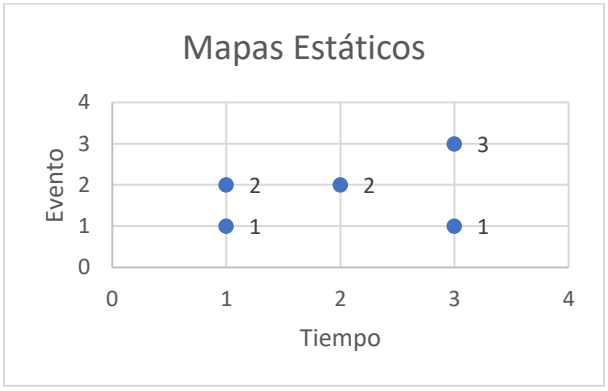


Imagen I.3.2 *Mapas estáticos Evento vs Tiempo.*

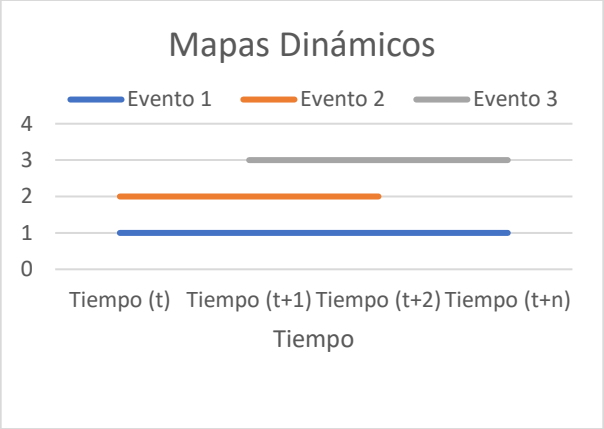


Imagen I.3.3 *Mapas Dinámicos Evento vs Tiempo.*

La cartografía estática como se ve en la imagen I.3.2 sólo está presente en un determinado tiempo y después de esto es descartada o inutilizada, a diferencia de la cartografía dinámica la cual estudia un evento en un intervalo de tiempo definido.

La cartografía dinámica representada en la gráfica I.3.3, es el tipo de cartografía más utilizada por los mapas actuales en donde un usuario tiene interacción con una base de datos y estos son mostrados mediante algún servidor, estos mapas son considerados dinámicos ya que un usuario se encarga de actualizar su base de datos y estos son mostrados en un mapa actualizado.

La actualización de los datos se puede realizar mediante dispositivos móviles apoyados con los Sistemas de Posicionamiento Global (GPS por sus siglas en inglés) integrados a estos dispositivos y con la conexión a internet estos son capaces de actualizar la información en tiempo real.

I.4 Influencia de los mapas en el desarrollo económico-social.

La cartografía es una herramienta fundamental en la mentalidad humana para percibir el universo en diferentes escalas, esta surge de la necesidad de ubicarse en el espacio de manera gráfica.

Los primeros mapas con enfoque científico son los realizados por los griegos, los cuales están construidos por un sistema de paralelos y meridianos, mostrando una forma aproximada de la tierra.

Con el paso del tiempo, en el apogeo del imperio Romano, su cartografía se caracteriza por tener orientación utilitaria es decir su cartografía es desarrollada para analizar la expansión del imperio.

Durante la Edad Media se registra un retroceso, está es una época catalogada como oscurantismo, los mapas en esta época tenían una representación teológica donde la tierra es plana, (son descartadas las teorías antiguas que proponían otro tipo de formas) en la cual se representaba un paraíso terrenal dentro de los tres continentes conocidos. En esta época a pesar del retroceso cartográfico se tiene un gran avance tecnológico debido a la invención de la brújula, en esta época se utilizaban las cartas ya no solo para la representación de los territorios cercanos o territorios por conquistar, también son utilizados con propósitos comerciales, los mercaderes consideraron la necesidad de mostrar las rutas comerciales.

En general la cartografía hasta la época del Renacimiento se enfoca principalmente en temas mercantiles y militares, adicionalmente la cartografía tiene características artísticas, esta no es la excepción en los mapas tienen con una representación artística, en ellos se ilustra además los linajes reales y las batallas importantes, durante esta época la cartografía es utilizada por personal de la milicia y la nobleza.

En la edad Moderna se han creado métodos cartográficos más precisos y confiables, además del invento de la imprenta los mapas son más accesibles al público en general, los intereses militares siguen siendo factor en el desarrollo de las nuevas tecnologías para localizar y controlar las fronteras de las diferentes administraciones territoriales.

Con el surgimiento de la fotogrametría en el siglo XIX y la computación en el siglo XX, los levantamientos cartográficos se realizan con mayor velocidad y precisión, los métodos de ortorectificación, aunque son anticuados son puestos a prueba.

En la creación de nuevas tecnologías de localización satelital, la cartografía tiene un nuevo giro pues ahora tiene altas precisiones y funciona con triangulaciones lo cual

permite tener acceso a lugares que anteriormente eran inaccesibles con equipos topográficos.

Actualmente la cartografía tiene gran impacto en la vida cotidiana, la mayoría de los teléfonos celulares cuentan con dispositivos de localización, los cuales ayudan a realizar cartografía dinámica apoyado de programas o paquetes especializados.

Los Sistemas de Información Geográficos ocupan tecnología GPS y tienen gran influencia en la vida cotidiana, ya que mantienen actualizada la información de localización de los usuarios, este tipo de sistemas son sistemas dinámicos, cuyos mapas base en general son mapas de Google con costo u OpenStreetMap licencia libre. Estos sistemas son utilizados en todo tipo de dispositivos y para distintas tareas, las cuales van desde encontrar algún lugar cercano a la posición para comer hasta localizar a algún contacto mediante su posición a horas de distancia determinando la ruta óptima. Esta se mantiene actualizada debido a la gran base de datos contenida en estos mapas, la cual es actualizada con información proporcionada por los usuarios, llamada cartografía colaborativa.

La cartografía colaborativa es generada gracias a la participación de los usuarios con un servidor, el cual provee la información al mapa, se clasifica como cartografía dinámica y su actualización se realiza constantemente, es utilizada con información que se proporciona desde vías remotas y está a su vez actualiza la información, a pesar de facilitar los procesos de levantado y actualización de la información, esta no siempre es de confiar, por lo cual siempre será necesario un proceso de validación para corroborar la veracidad y confiabilidad de los datos mediante un proceso de selección y descarte de información.

II Lenguajes de programación aplicados a los visualizadores.

II.1 Lenguajes Web

Los lenguajes de programación web surgen desde la creación del internet, debido a la necesidad de visualizar información en los navegadores. Estos lenguajes no necesitan un codificación especial o compilación previa puesto que los exploradores reconocen la mayoría de los comandos en los diferentes lenguajes.

Existen diferentes tipos de lenguajes de programación e interpretadores, para la creación de software, los más utilizados actualmente son HTML5 con CSS por parte de los interpretadores y JavaScript y PHP del lado de los lenguajes de programación.

II.1.1 HTML 5 y CSS.

El Lenguaje HTML es una serie de comandos y nodos utilizado por los navegadores de internet, para representar las diferentes formas de páginas web, desafortunadamente HTML cuenta con características gráficas muy simples, funciones básicas de consulta y almacenamiento de datos. Además de que al ser un interpretador marcado, este deja de tener características dinámicas lo que imposibilita realizar modificaciones una vez interpretado el código por el explorador. Debido a esto los desarrolladores se apoyan en diferentes lenguajes de programación y hojas de estilo como CSS, los que facilitan distintas funciones y estilos para mostrar al usuario final una página amigable y adecuada a las necesidades fundamentales de la misma.

CSS es utilizado para dar forma y estilo a los elementos dentro del código HTML, en este se especifican tamaños, formas, animaciones, colores, en general todo lo relacionado a con la visualización de la página web.

En este proyecto se ocupan en conjunto para dar forma a la visualización en navegadores web y el CSS como auxiliar en los estilos de visualización de HTML.

HTML es complementado por CSS es muy útil y puede ser modificado, personalizado o adaptado dependiendo las necesidades del proyecto o sitio en cuestión.
Hay que resaltar que toda tecnología nueva o de legado, su base para poder generar un visualizador o cualquier sitio web utilizará los principios de HTML para existir.

II.1.2 JavaScript

Este lenguaje fue desarrollado por Netscape a principios de los años 90´s debido a la necesidad de ejecutar tareas a mayor velocidad y desde el explorador del usuario, sin necesidad de obtener respuestas de los servidores, de esta forma si el usuario comete algún error al rellenar algún formulario la respuesta de este será casi inmediata, aunque el nombre JavaScript es considerado hasta que Netscape y Sun firman un acuerdo colaborativo del lenguaje.

JavaScript es un lenguaje que funciona de la mano con el explorador, por lo que este es capaz de integrar las funciones HTML, como ya se mencionó todos los lenguajes que tengan que ser visualizados en un explorador parten de los paradigmas que plantea HTML.

En el proyecto existe la necesidad de trabajar con este lenguaje dado que la carga de información dentro de los mapas y puntos de Google Maps se realiza mediante estos protocolos.

Este lenguaje se utiliza para poder programar la interpretación de los datos dentro del visualizador web. Este lenguaje se utiliza con la finalidad de poder mostrar los mapas y realizar un análisis correcto de los datos dentro de las bases de datos y servicios.

JavaScript se encargará de realizar todas las interpretaciones de datos y su traducción a HTML para poder ser visualizada dentro del proyecto, considerando que se realizan conexiones a bases de datos.

JavaScript será nuestra herramienta para poder generar nuevos puntos, realizar la captura de datos para nuevas inserciones puntos, así como el que determinará la simbología para poder obtener un visualizador de fácil interpretación.

II.1.3 PHP

PHP es utilizado para incrustar contenido dinámico, conectar con bases de datos y escribir scripts de JavaScript, el uso principal de este lenguaje es realizar funciones del lado del servidor.

Con la ayuda de un servidor, se puede procesar el código PHP para actualizar el contenido dentro de las páginas o funciones previamente mencionadas (HTML, JavaScript). El servidor procesa el código en PHP para mostrar un resultado en HTML o cualquier otro lenguaje.

La utilidad del lenguaje PHP en este proyecto es apoyar a los accesos y mediante diferentes usuarios, además de que este lenguaje también es capaz de ser modificado con aplicaciones móviles.

En la imagen II.1.3.1 se ilustra el diagrama de funcionamiento del sistema.

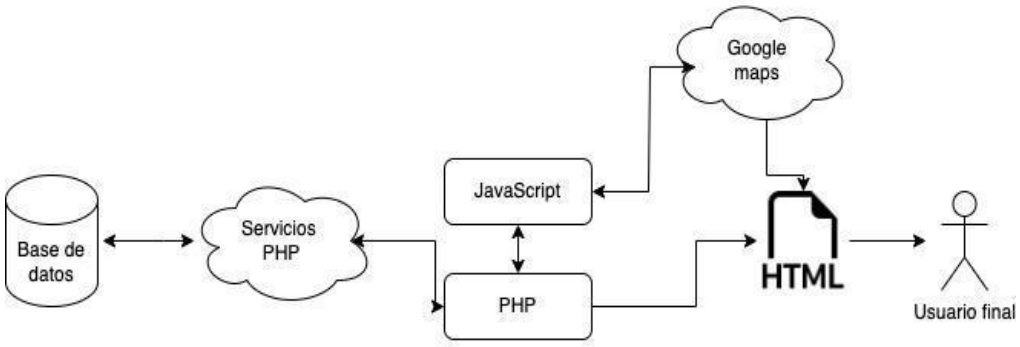


Imagen II.1.3.1 *Diagrama de arquitectura de usando servicios PHP con JavaScript*

II.1.4 Golang y ReactJS

La arquitectura de los sistemas web, ha sido mejorado a lo largo del tiempo, los servidores físicos o incluso servidores en nube ya no están siendo utilizados, estos están siendo reemplazados por lambdas los cuales son interpretadores que solo son ejecutados cuando son llamados por lo cual tienen la ventaja de no tener la necesidad

de estar encendidos todo el tiempo, sin embargo, es necesario levantar el lambda para cada petición lo que hace necesario contar con un lenguaje mucho más eficiente para el arranque del sistema.

El procesamiento de la información del lado de la nube puede ser desarrollada de una forma diferente a nivel de arquitectura, más eficiente para el consumo de servicios y procesamiento del portal web, además de que hay que resaltar los avances en la seguridad de los nuevos lenguajes de programación.

El lenguaje de programación Golang (GO), es utilizado actualmente para la creación de microservicios y procesamiento en la nube, este al ser un lenguaje que se ejecuta sin una máquina virtual, es altamente eficiente por lo cual los costos, rendimiento y procesamiento de la información es optimizado por el mismo lenguaje, en otras palabras, es más rápido en el procesamiento y ejecución de los servicios.

Este lenguaje y lambdas hacen más sencilla la arquitectura, procesamiento y mantenimiento de los sistemas.

Complementado en la parte visual con ReactJS nos facilita la arquitectura del sistema para así tener un canal bidireccional sin necesidad de utilizar lenguajes auxiliares como en la arquitectura anterior.

El lenguaje de Golang está integrado del lado de la nube por lo que es necesario realizar otro tipo de implementaciones, para poder realizar las visualizaciones adecuadas, para este escenario se utiliza ReactJS que es un avance de JavaScript y es compatible con este, por lo cual puede ser utilizado para la implementación de Google Maps.

La forma de visualizar este tipo de arquitectura se muestra en la imagen II.1.4.1, en este caso no es necesario la intervención de otro lenguaje y se tienen mejor separadas las clases y funcionalidades.

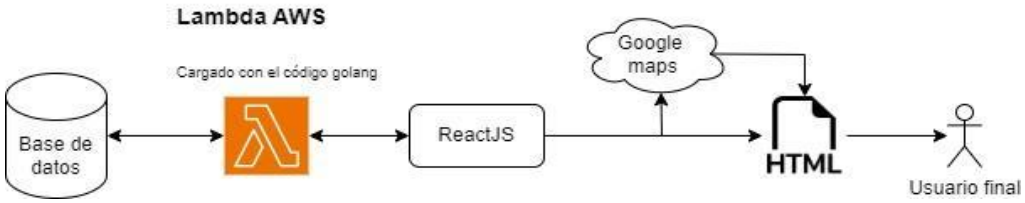


Imagen II.1.4.1 *Diagrama de estructura de microservicios Golang y ReactJS*

II.2 Lenguajes para aplicaciones con sistema operativo Android

Aunque durante el desarrollo de la primera versión de la aplicación las aplicaciones de Android deberían tener un peso menor a 64K hoy en día es importante mencionar que con el avance computacional de los dispositivo esta limitante ha sido removida hoy podemos tener aplicaciones de cualquier tamaño en los dispositivos, sin embargo, siempre es importante considerar que un dispositivo móvil por más memoria que este tenga esta va a ser finita, por lo que es importante hacer que nuestras aplicaciones no sean tan pesadas.

Siempre hay que tener en consideración la capacidad computacional de los dispositivos para así tener una mejor experiencia con estos.

Hoy en día el sistema operativo Android cuenta con varias herramientas que nos son de utilidad para este proyecto, el más importante es el GPS, este nos será auxiliar en la alimentación de los datos para la cartografía dinámica que buscamos conseguir.

Aprovechando este hardware buscamos poder alimentar la base de datos, la que nos brindará posiciones e información relativa al sistema. La arquitectura para el proyecto será la comunicación de la aplicación directo con la lambda de AWS como se enseña en la imagen II.2.1

Imagen II.2.1 *Diagrama de estructura de microservicios Golang y Aplicación Android*

Esta arquitectura nos permite realizar y actualizar en tiempo real los visualizadores tanto web como en otros dispositivos.

II.2.1 Java y XML.

El lenguaje Java es un fue desarrollado y creado por Green Team el cual existe en forma desde 1995 gracias ellos y al navegador Netscape que es el primero en implementar la tecnología Java hoy es un lenguaje muy popular para el desarrollo de aplicaciones.

Actualmente Java está mucho más presente en nuestras vidas y ya no solo existe para aplicaciones de internet, también lo podemos encontrar en diferentes dispositivos con distintos sistemas operativos, como es este caso, una aplicación para tableta con sistema operativo Android.

Java es un lenguaje de programación que después de ser compilado necesita ser interpretado por una máquina virtual, lo que hace que los dispositivos puedan ejecutar las funciones programadas, la importancia de esta máquina virtual es de vital importancia, pues hoy en día las aplicaciones ya no son desarrolladas nativamente en Java, estas solo utilizan su máquina virtual para realizar la interpretación del código.

Java fue utilizado para toda la parte lógica de la aplicación y XML fue utilizada para la parte visual, el pseudo lenguaje XML es utilizado en Android para hacer la declaración de los atributos visuales dentro de la aplicación, aunque al ser un ente externo al lenguaje este tiene que ser procesado durante su compilación y este no podrá ser modificado en el tiempo de ejecución.

Esta característica particular hace que las aplicaciones están sobrecargadas por elementos innecesarios.

II.2.2 Kotlin y Compose.

En la actualidad el lenguaje utilizado para desarrollar aplicaciones en Android es Kotlin que, aunque este corre sobre la máquina virtual de Java, este lenguaje ofrece varias funcionalidades que Java no podría ofrecernos o costaría mucho código y esfuerzo poder desarrollarlas un ejemplo típico se muestra en el fragmento de código II.2.1.1, donde el generar una lista implica una declaración dentro del ciclo para recorrer los arreglos.

```
fun getElementsOnList(){
        val pointsList = arrayListOf(element1, element2, element3)
        pointList.forEach{ element ->
                transformacion(element)
        }
}
```

```
private void getElementsOnList(){
        ArrayList<Elements> pointsList = arrayList(element1, element2, element3);
        for ( Elements element : pointsList){
        transformacion(element);
        }
}
```

Como se puede apreciar en el fragmento de código II.2.1.1 al trabajar con Kotlin nos reduce mucho el trabajo al desarrollar una aplicación, en el desempeño de la aplicación nos va a dar mejores resultados desarrollar aplicaciones en Kotlin, haciendo de estas más fáciles de leer y de mantener.

Otra ventaja que nos ofrece el lenguaje es que al ser ejecutado dentro de la misma máquina virtual que Java, es la compatibilidad de lenguaje, podemos llamar a los

archivos .java y .kt sin problema cuando estamos trabajando en las aplicaciones para así lograr migraciones seccionadas y moduladas.

Adicional Kotlin nos ofrece la facilidad de poder trabajar, las vistas sin necesidad de agregar otro lenguaje como XML las visuales las podemos trabajar directamente en clases de Kotlin utilizando Compose, para el caso específico de Android tenemos Jetpack Compose, estas clases al entrar al compilador pasan como código nativo, por lo que no es necesario generar clases intermedias que podrían ocasionar errores y así ayudar a mejorar el desempeño de la app para lograr una navegación y utilización más eficiente en las aplicaciones Android.

II.3 Representación con mapas de Google con datos precargados.

Para desarrollar la aplicación se utilizó el lenguaje de programación Java y XML, en su primera versión en la siguiente versión se utilizó Kotlin y mapa base se utiliza Google Maps.

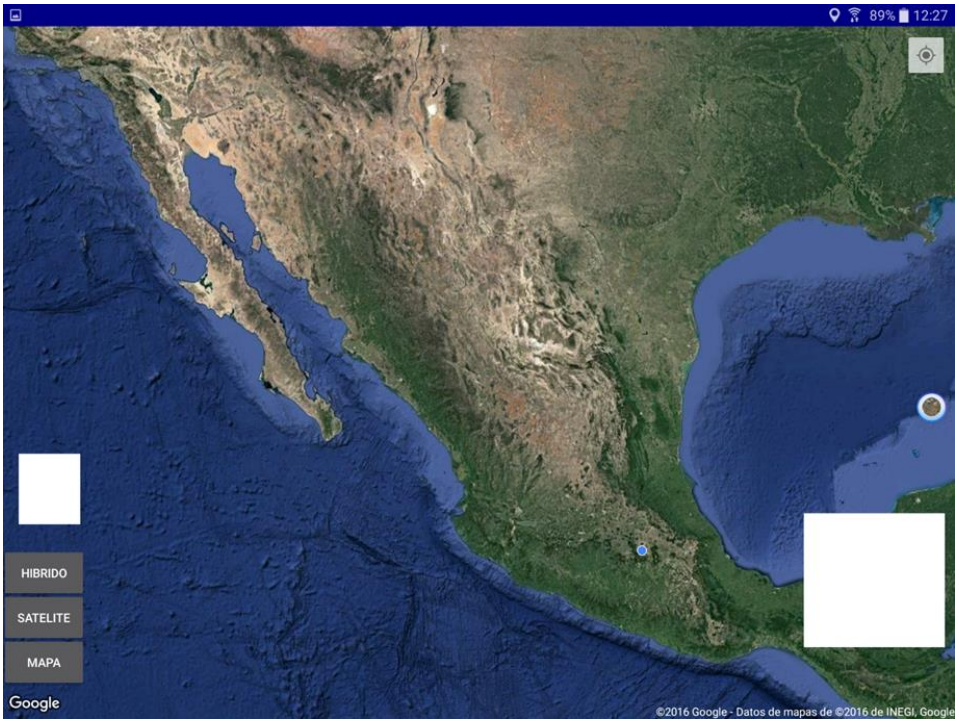
El primer paso es agregar las librerías y mapas correspondientes a cada operación

Como se ve en la imagen II.3.1 este solo es un mapa sin información o algún dato que visualizar debido a que esta es la información que se tiene que cargar mediante marcas de posición, líneas y polígonos.

Para poder realizar estas cargas de datos es necesario seguir los siguientes pasos, para generar, procesar, corregir y visualizar la información relacionada con los puntos:

- Proceso y análisis de capas y datos.
- Obtención de la información y visualización en el equipo de escritorio.
- Creación y validación de shapes.
- Creación de archivos XML
- Corrección de XML
- Validación de la información (Opcional)

- Visualización de archivos XML dentro de los mapas base.



Mapa II.3.1. Mapa de México mostrado dentro de la aplicación.

Preproceso y análisis de capas y datos.

Para poder mostrar la información necesaria dentro de las aplicaciones móviles esta debe llevar un proceso de validación y verificación de información. Ilustrado en la imagen II.3.2

El proceso consiste en 5 pasos los cuales son:

1. Obtención de la información y visualización en el equipo de escritorio.
2. Creación y validación de shapes.
3. Creación de archivos XML
4. Corrección de XML.
5. Validación final de la información.

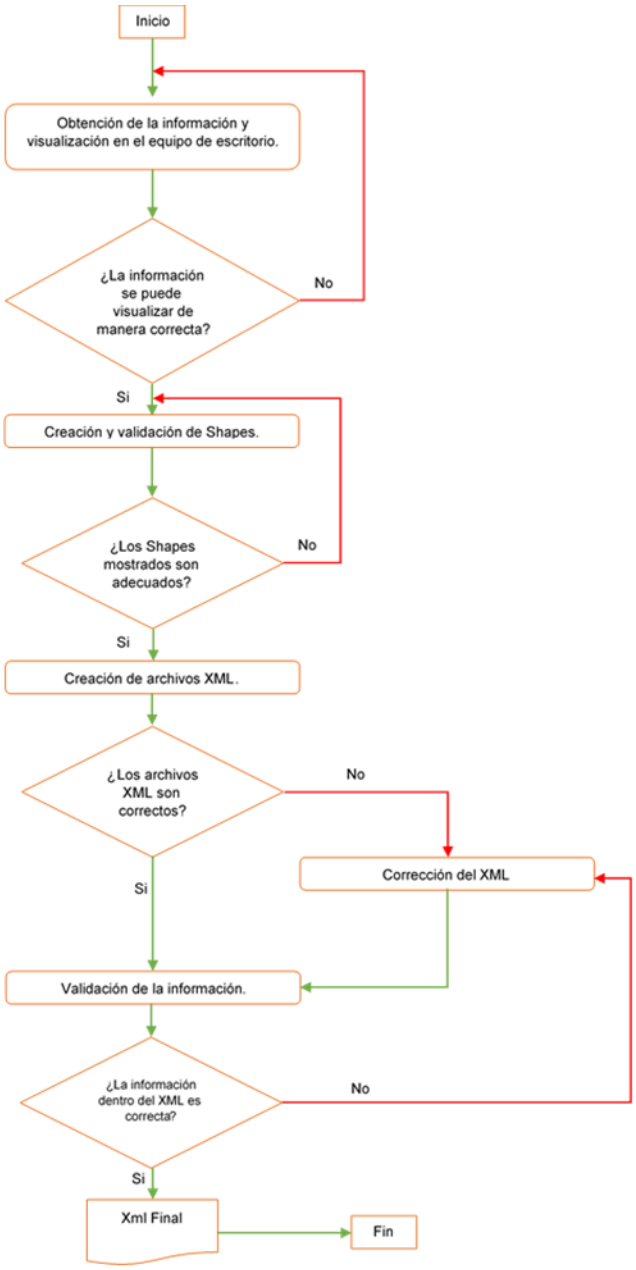


imagen II.3.2 *Diagrama de flujo para la corrección de XML*

Obtención de la información y visualización en el equipo de escritorio.

La información para mostrar es recabada por personal de campo y es presentada en tablas de Excel sin procesar, cuentan con información que no es necesaria para la visualización esta información debe ser eliminada para presentar la carga de versión final de las tablas.

Además de la información complementaria a mostrar, las tablas deben de contener un campo de identificación o id y la ubicación espacial “X” y “Y” o “λ” y “Φ”. Estos son utilizados para poder referenciar la información y ubicarla en un plano o en este caso el visualizador, parte de las validaciones que se realizan en este proceso es el verificar que contengan la información espacial, de lo contrario esta no podrá ser visualizada de ninguna manera.

Para este caso como software auxiliar para el tratamiento y visualización de la información se utiliza el software ArcMap el cual con la información espacial puede mostrar los puntos y con ayuda del mapa base se puede verificar que la información se encuentre bien colocada.

Creación y validación de shapes.

Validadas las tablas y comprobada la correcta localización de los puntos se realiza la creación de archivos shape, para poder ser procesados con el mismo software, con la finalidad de encontrar datos corruptos o eliminados, el realizar esta importación facilita la lectura de la información que se visualizará al final del proceso en la aplicación.

Creado el archivo shape se le tiene que asignar simbología ya que nos permite visualizar los puntos y una forma fácil de diferenciarlos unos con otros según sus características.

Por último, se edita la tabla visual de cada archivo, eliminando información que no es significativa para la visualización dentro de la tableta.

Creación de archivos XML

Este archivo es el más importante del visualizador pues contiene toda la información necesaria para trabajar dentro del visualizador como ya se ha mencionado anteriormente.

El archivo es generado a partir de la información dentro del archivo shape y contiene la información que el visualizador necesita para funcionar de manera adecuada como: ubicación, descripción, comentarios, entre otros tipos de datos de utilidad. Es generado en ArcGIS con la herramienta de conversión a KML, al realizar esta conversión obtenemos un resultado XML que puede ser utilizado dentro del visualizador.

Corrección de XML

Como se mencionó anteriormente los archivos sin procesar resultantes de este proceso contienen mucha información valiosa pero también incluyen información basura y vacíos como la línea de comando, un ejemplo es la línea `<Sniplet></Sniplet>`. Las líneas vacías, aunque no afectan la lectura del archivo dentro del visualizador, ocupan memoria valiosa y reduce el desempeño de la aplicación, por lo que parte de este proceso de corrección es eliminar esos nodos.

En este paso es procesada la información que no se ha corregido en las fases anteriores ya sea por omisiones, como no arreglar algún nombre de los campos, o por algún descuido, como las faltas de ortografía, es donde se da la edición final y los ajustes necesarios para hacer que la tabla de información y la localización de los puntos sea adecuada y con la información requerida.

Validación de la información. (Opcional)

Terminadas las correcciones se puede visualizar la información creada en Google Earth para verificar la existencia de las tablas y comprobar que el formato dado sea el indicado.

Al ser creado el archivo KMZ se puede verificar también el contenido de las tablas dentro de su archivo KML en el nodo `<description></description>` al estar escrito en lenguaje HTML se le pueden hacer las correcciones directamente desde el código o volviendo al paso “Corrección de XML”.

Visualización de archivos XML dentro de los mapas base.

Teniendo el archivo XML y el visualizador del mapa funcionando es entonces cuando nos dirigimos a la aplicación.

Agregar las marcas en el visualizador implica que tienen que leer el archivo XML previamente generado dentro de la aplicación, con el fin de representar la ubicación y leer la información dentro de la tabla de atributos asociada a cada punto.

Colocar las marcas de posición se realiza con las herramientas que nos proporciona el mapa, las marcas y líneas. Como ya se tiene generado el archivo XML y el visualizador ya cuenta con el mapa precargado la visualización se realiza leyendo el archivo dentro del visualizador y transformándolo en un arreglo mutable, para así con un bucle poder mostrar las marcar y descripciones para cada punto en el archivo XML.

Una vez recorrido todo el arreglo la lista y agregados los puntos con esta el visualizador podrán mostrar los puntos como se ve en el imagen II.3.3

Aunque esta es un procesamiento de datos que podría llegar a pensar que es muy manual, existen muchas opciones que pueden ser más automatizadas, como la carga de archivos JSON para alimentar la base de datos, la carga de bases de datos directamente dentro de la aplicación, la precarga de listas durante la creación del visualizador, todas estas opciones son similares y tienen la misma desventaja, estos siempre dependen de un archivo externo para poder procesar y visualizar la información.

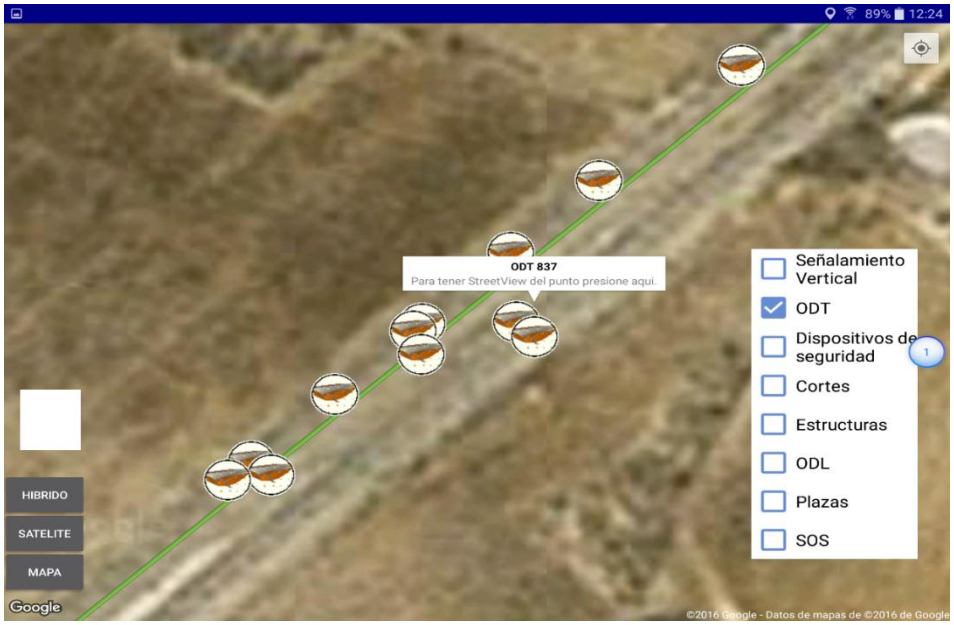


Imagen II.3.3 *Imagen de visualización con ODT seleccionado para mostrar*

II.4 Representación con mapas de Google con datos de una base de datos remota.

Tal como en el caso anterior los pasos para visualizar el mapa son los mismos, la diferencia principal es que no se precargarán los datos debido a que será un sistema dinámico el cual tendrá que tomar la información desde una base de datos almacenada dentro de un servidor remoto.

Este tipo de sistema es un sistema integral, el cual por su misma estructura puede ser visualizado tanto en sitios web como en la aplicación.

Crear este sistema necesita un orden y una serie de pasos los cuales se describen en la imagen II.3.4

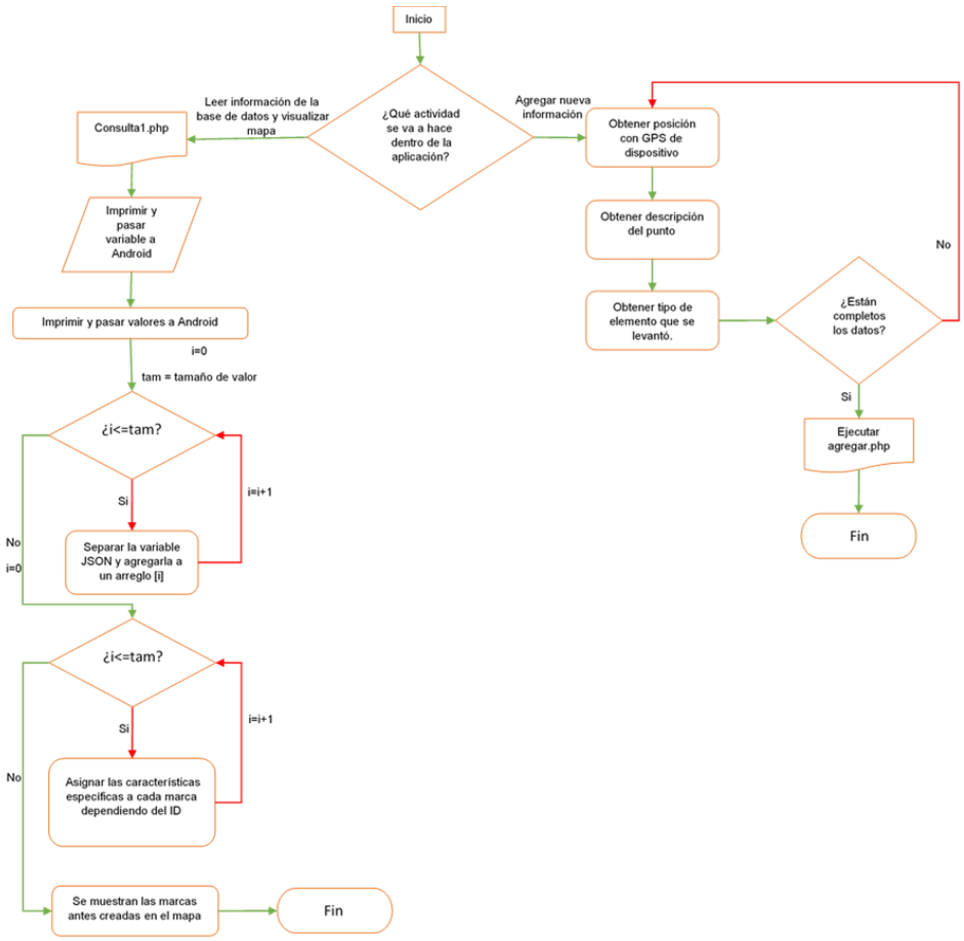


Imagen II.3.4 *Diagrama de flujo descriptivo de los pasos dentro de la aplicación*

El crear el visualizador, requiere de diversos procesos, los cuales serán detallados para entender mejor el funcionamiento.

Estos procesos serán divididos en tres diferentes tipos para facilitar el estudio.

1. Creación, revisión y descripción de la base de datos.
2. Conexión y manipulación de la base de datos de manera remota
3. Agregar nuevos inventarios.
4. Leer y mostrar los inventarios existentes.

Creación, revisión y descripción de la base de datos.

Para poder visualizar los datos es importante la estructura de la base de datos ya que son los que serán mostrados en cada marca de posición.

Los campos que debe contener toda base de datos espacial son tres: id o identificador único, latitud y longitud; esto es necesario para poder identificar cada uno de los puntos y representarlos en el mapa. Adicional es importante considerar la información adicional o de interés para agregar dentro de la base de datos como: descripción, tiempo, url relacionadas al punto, entre otros. Estos campos pueden ser personalizados dependiendo del interés del usuario final.

La base de datos cuenta con información necesaria para cada uno de los puntos, en este caso se utiliza una columna de descripción y una de tipo, el objetivo de estas columnas adicionales es para poder realizar la clasificación por tipo de inventario y proporcionar la información de la ventana de descripción, sin embargo, esta ventana puede contener todo tipo de información.

Durante la creación de la base se debe de tener claro qué tipo de información necesita ser mostrada en los visualizadores para así evitar el almacenamiento de información que no es de interés.

Conexión y manipulación de la base de datos de manera remota.

Para poder conectar y manipular la base de datos, es necesario contar con un protocolo de comunicación entre los visualizadores y los servidores; esto se realiza con un API REST, la cual con los verbos GET y POST funcionará como mensajero para detonar las funciones del lado del servidor.

Esta comunicación puede ser representada como se muestra en la imagen II.3.5

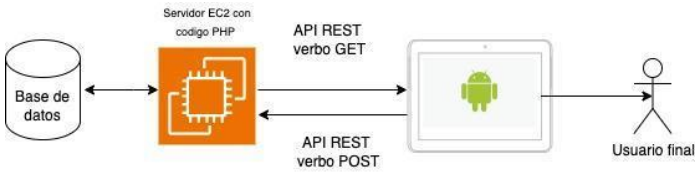


Imagen II.3.5 *Comunicación entre servidor y aplicación utilizando servicios REST*

Una vez creada la API y conectada con la aplicación, el servidor y la base de datos estarán disponibles para poder trabajar dentro de los visualizadores.

Agregar nuevos inventarios.

La API deberá tener un servicio, el cual será el encargado de agregar los puntos a la base de datos, el verbo POST, este servicio es el encargado de recibir la información desde los visualizadores en el cuerpo de la petición.

Esta APIs tendrá que recibir un cuerpo en formato JSON, para poder almacenar de manera correcta el punto guardado con el dispositivo.

Ejemplo de petición.

```
{
        “punto”: {
                                “lat”:19.000,
                                 “long”:-104.00
                                },
        “tipo”: “escuela”,
        “descripción”: “Esta es una escuela bonita”
}
```

Leer y mostrar los inventarios existentes.

Cuando se consumen los servicios de PHP desde el visualizador se utiliza el verbo GET, lo que quiere decir que para poder leer la información mostrada no es obligatorio enviar un cuerpo en la petición, sin embargo, este si nos regresará la lista de datos guardados en la base de datos, al igual que cuando se realiza la petición anterior este responderá una lista con los puntos asociados.

Ejemplo de la respuesta:

```
[
        {
                        “punto”: {
                                        “lat”:19.000,
                                         “long”:-104.00
                                        },
                        “tipo”: “escuela”,
                        “descripción”: “Esta es una escuela bonita”
        },
        {
                        “punto”: {
                                        “lat”:19.500,
                                         “long”:-104.50
                                        },
                        “tipo”: “escuela”,
                        “descripción”: “Esta es una escuela fea”
        }
]
```

Leer los valores enviados por el API dentro de la aplicación en Android se consigue deserializando la respuesta y creando una lista; la cual deberá de ser interpretada como el caso de visualizadores estáticos y dibujada con un ciclo for.

III Bases de datos y acceso a ellas.

III.1 Bases de datos en servidores.

En los visualizadores es importante la transmisión, procesamiento y almacenamiento de datos, dado esto se debe prestar especial atención a este tema.

Aunque ya se ha mencionado anteriormente algunos aspectos importantes es necesario profundizar en la alimentación y gestión de datos del sistema debido a que es la parte medular en los visualizadores.

Para poder comunicar la aplicación con la base de datos central es necesario contar con una conexión a internet estable al momento del levantado de los puntos, como se comentó en el capítulo anterior la aplicación comunicará información de interés y requerida para el almacenamiento de los datos dentro del servidor.

Las bases de datos están estructuradas por tablas las cuales contienen diferente información organizada, en este caso la base de datos no está correlacionada sin embargo contiene dos tablas.

La primera es la tabla de inventarios esta fue agregada debido a la importancia de los elementos que contiene, en ella se almacenan los tipos de elementos para clasificar los puntos y se asigna la simbología correspondiente a cada elemento. (Para fines prácticos la tabla ha sido limitada a cuatro elementos exclusivamente, aunque puede ser de cualquier tamaño y cantidad de inventarios como sea necesaria)

Otra tabla utilizada dentro de la base de datos es la encargada de almacenar los datos correspondientes a los puntos, esta es la tabla que contiene información relevante la cual es mostrada para cada tipo de elemento, además de un ID único para identificar cada elemento tiene campos como latitud, longitud, descripción y tipo (referente a el tipo de inventario que contiene).

Las bases de datos son de suma importancia para el almacenamiento de información geográfica y gracias a ellas podemos visualizar mediante comandos de programación esta información en un mapa.

III.2 Bases de datos en dispositivos móviles.

Estas bases son parecidas a las bases dentro de servidores sin embargo para poder acceder a ellas no es necesaria una conexión a internet, estas ocupan memoria dentro del dispositivo.

Las bases de datos locales debido a que no tienen ningún respaldo pueden ser eliminadas fácilmente de los dispositivos, por lo tanto, no son recomendadas para almacenar datos indispensables de la aplicación, para el caso del levantamiento de puntos pueden ser utilizadas.

Caso 1.

El usuario no tiene acceso a internet en campo sin embargo necesita realizar el levantamiento de los puntos del día, podrá finalizar la jornada ya que durante el trabajo de campo estos serán respaldados en el servidor.

En este caso la base de datos almacenará la información levantada por el usuario en el dispositivo de manera temporal, hasta que el dispositivo tenga una conexión a internet, una vez respaldada la información del levantamiento esta puede ser eliminada de manera automática.

Caso 2.

El usuario necesita revisar los puntos previamente levantados, ya sea de días anteriores o de levantamientos anteriores, sin embargo, la infraestructura de la red no brinda acceso a internet, por lo tanto, almacena los datos de manera local para poder visualizar la información en el momento que sea necesario, esta información es almacenada y borrada a disposición del usuario.

Así como estos dos casos se podrán encontrar distintos escenarios para el almacenamiento de datos dentro de manera local.

Para almacenar la información de manera local, Android proporciona muchas herramientas para facilitar la administración entre las más populares se encuentran:

- SQLite
- ORM
- MySQL
- ROOM
- Mongo DB

Estas herramientas son las más populares, pero no las únicas disponibles, además de ser utilizadas para facilitar el procesamiento y la gestión de los datos.

III.3 Importancia de las bases de datos y estructuración de la información en los visualizadores.

Siendo reiterativos en el tema, la información almacenada dentro de las bases de datos es la que le va a dar utilidad a los visualizadores. Cuanto mayor es la información se tenga sobre los puntos, un mejor análisis de los datos se podrá realizar.

Pero tan importante es la información que se almacena como la estructuración de la información, es decir, no es recomendable almacenar toda la información en una sola tabla, como se ha mencionado anteriormente, las tablas deben contener información relacionada a un tema en específico, no se debe mezclar la información de geolocalización con descripciones.

Para poder relacionar estas tablas se tendrán que realizar querys los cuales resultaron en una mejor consulta.

El tener la información organizada nos dará la posibilidad de realizar análisis más exhaustivos con la información capturada con la aplicación.

IV Conclusiones y utilidades de los visualizadores.

IV.1 Discusión y utilidades

IV.1.1 Sistemas Web.

Los visualizadores web como ya se demostró son capaces visualizar los datos en una computadora mediante el uso de internet, sin necesidad de previamente tener instalado ningún tipo de programa especializado, estos son utilizados particularmente de forma demostrativa y ya se ha mencionado que también son ser dinámicos, este tipo de características requieren de un nivel de programación mayor.

Los sistemas web son recomendados para un nivel gerencial ya que estos al estar en un servidor pueden ser visitados en cualquier momento o lugar con acceso a internet.

Dado que toda la información se encuentra en bases de datos el almacenamiento de esta es de baja densidad y de fácil acceso, gracias a este tipo de almacenamiento es necesario de personal para el mantenimiento de la base de datos, así como desarrolladores para nuevas funciones dentro de la página web, los elementos especializados para el análisis de los componentes lo que representa un costo para el proyecto.

Este tipo de visualizadores en esencia son útiles cuando el usuario tiene un perfil de no especialista en SIG o en sistemas, sin embargo, no dejan de ser llamativos y útiles. Estos visualizadores al ser programados a la medida pueden tener las herramientas y características exclusivamente necesarias para el perfil del usuario o negocio.

Además de ser un visualizador a la medida estos no requieren de actualizaciones automatizadas ya que al ser un diseño web y funcional bastará con refrescar la vista de la página para mostrar las últimas actualizaciones.

Una limitante de este tipo de sistemas debido a su naturaleza es necesario la conexión a internet ya que sin esta no se podrá visualizar ningún tipo de datos.

IV.1.1.2 Visualizadores móviles.

Estos visualizadores están dentro de un dispositivo móvil, los cuales son útiles cuando se necesita llevar la información a campo, pues gracias a su fácil acceso y portabilidad se puede verificar e incluso levantar información al momento.

Su programación es compleja por lo tanto se requiere de un programador especializado para poder desarrollar un sistema que funcione en este tipo de plataformas de manera eficaz.

Para mantener la aplicación actualizada es necesaria una conexión a internet estable, sin embargo, los visualizadores móviles pueden prescindir de esta, para eso se ocupa memoria del dispositivo cuando no se tiene conexión.

Una vez desarrollada la aplicación para el dispositivo, esta debe ser fácil de utilizar para el usuario, ya que este muchas veces no contará con capacitación especializada, por lo tanto, es importante tener en cuenta la experiencia del usuario.

Las aplicaciones móviles, aunque pueden llegar a ser muy poderosas, se ven limitadas por el tipo de dispositivos en donde son instaladas y a diferencia del sistema web los cuales tienen una afectación directa con la eficiencia del sistema.

Debido a los grandes avances dentro de la tecnología y a la gran necesidad de mantener actualizada la información cartográfica, es necesario que pueda ser renovada con mayor facilidad y con mayor eficiencia, aunque por un lado se tiene que ser consciente que esto puede representar también pérdidas, una importante es la precisión milimétrica sin embargo para realizar cartografía temática este tipo de sistemas pueden llegar a ser de gran utilidad; por lo que es de suma importancia el desarrollo y entendimiento de los mismos.

IV.1.1.3 Visualizadores estáticos.

Los visualizadores estáticos nos ofrecen una disponibilidad de la información sin importar la situación en la que se encuentre el dispositivo, nos ayudan en tareas muy específicas y que no requieren tener la información actualizada.

Estos son utilizados para revisar datos en campo sin necesidad de tener conexión a internet ya que la comunicación y actualización no es necesaria cuando se utilizan este tipo de visualizadores.

Contar con estos visualizadores reduce los costos de mantenimiento tanto de servidores como de desarrolladores dado que la información es inmutable y su mantenimiento es mínimo.

Las desventajas que tienen es que al ser un visualizador que no actualiza la información esta suele estar desactualizada y no tienen la capacidad de realizar modificaciones en caso de ser necesario.

Los casos de uso probables para este tipo de visualizadores son visualización de monumentos históricos, eventos históricos, espacios arqueológicos o incluso estudios geológicos dado que la información de estos no cambia o cambia muy poco con el paso del tiempo. Haciendo así que no sea necesario consumir o actualizar los datos en el corto tiempo.

IV.1.1.4 Visualizadores dinámicos.

La contraparte como se ha estado explicando a lo largo del documento son los visualizadores dinámicos estos tienen la ventaja de estar siempre actualizados en tiempo real proporcionando información actualizada y veraz en tiempo real.

En estos se tiene que estar realizando mantenimientos constantes, por lo que su costo es mayor a los visualizadores estáticos, estos necesitan estar mejorando la experiencia de usuario y protocolos de comunicación para tener un mejor desempeño.

Aunado a esto otra desventaja que nos arroja este tipo de visualizadores es que son dependientes totalmente del uso del internet y sin la conexión dejan de cumplir con la función principal, aunque existen medios para mitigar este tipo de fallos, no deja de presentarse como una desventaja en los visualizadores dinámicos, aunque estos pueden almacenar la información de manera temporal y después pueden ser sincronizados con los servidores se estaría perdiendo la esencia de tener un visualizador

IV.1.1.5 Tablas comparativas entre tipos de visualizadores

Con este estudio se puede determinar algunas relaciones para comparar entre los diferentes tipos de visualizadores tenemos la comparativa entre visualizador web y móvil que, aunque estos son complementarios, dado que el visualizador web depende de la información que proporciona el dispositivo estos también pueden ser utilizados por separado.

Como se aprecia en la tabla, cada visualizador tiene sus ventajas y desventajas y estos deberán ser utilizados dependiendo de la situación o necesidad del usuario final, para así lograr una mejor explotación de los datos.

	Visualizador Web	**Visualizador móvil**
Almacenamiento local	No	Si
Almacenamiento remoto	Si	Si
Tipo de visualizador	Dinámico	Estático / Dinámico
Portabilidad	No	Si
Necesidad de internet	Si	Solo en el caso dinámico
Necesidad de actualizaciones	No	Si

Tabla IV.5.1 *Diferencias entre visualizador web y visualizador móvil.*

Por otro lado, tenemos la comparativa más importante, esta es entre visualizadores dinámicos y estáticos, que como su nombre indica uno es mutable y el otro no y aunque sus usos son diferentes hay que resaltar las diferencias entre ellos, así como las ventajas y desventajas entre ellos

	Visualizador Estático	Visualizador Dinámico
Necesidad de internet	No	Si
Información actualizada	No	Si
Uso de servidores	No	Si
Desarrollo de apis	No	Si
Corrección de puntos	Si	No
Uso de memoria del dispositivo	Si	No

Tabla IV.5.2 *Diferencia entre visualizador estático y visualizador dinámico.*

IV.2 Conclusiones

Durante el desarrollo de este trabajo se mostraron principales diferencias entre los visualizadores para así dar razón y directrices sobre el tipo de visualizadores que se pueden utilizar, además de ilustrar las diferentes arquitecturas de software que pueden ser utilizadas para el desarrollo de visualizadores personalizados. También se explicaron las principales diferencias entre los tipos de visualizadores para así ayudar a la selección de la mejor opción de visualizador de acuerdo a la necesidad del proyecto, con lo cual se cumplen los objetivos buscados para realizar el desarrollo de visualizadores o SIGs más especializados.

En el presente trabajo se ha mostrado también que los SIGs pueden ser accesibles desde cualquier plataforma sin necesidad de software especializado sacrificando la precisión logrando así cumplir con la accesibilidad y disponibilidad de la información

Bibliografía.

1. INEGI. (Agosto 2006). Antecedentes de la Cartografía. Enero 2016, de Instituto Nacional de Estadística y Geografía Sitio web: http://www.inegi.org.mx/inegi/SPC/doc/internet/antecedentescartografia.pdf

2. Harley J.B. & Woodward D. 1897. The map and the Development of the History of Cartography. *The history of cartography* (622, pp.1-5) Chicago & Londres: The university of Chicago press.

3. Errázuriz A.M., Gonzalez I. & Rioseco R. (1988). Cartografía Temática. Santiago de Chile: Ediciones Universidad Católica de Chile

4. Schmitz, P., Cooper, A. & Krygsman, S.. (Diciembre 2006). Cartografía Móvil. ICA News, No. 47, (26, pp21-23).

5. Sánchez, M. (abril, 2008). Orbis Terrarum (El círculo de la Tierra). Enero,06,2017, de Galicia Romana Sitio web: http://galiciaromana.blogspot.mx/2008/04/orbis-terrarum-el-crculo-de-la-tierra.html

6. Hermosilla. V (noviembre, 2014). El mapa portulano y la navegación moderna. Enero, 06 2017. De Témpora Magazine de Historia Sitio web: http://www.temporamagazine.com/el-mapa-potulano-y-la-navegacion-en-la-edad-moderna/

Imágenes.

1. Imagen I.1.1 Orbis Terrarum http://galiciaromana.blogspot.mx/2008/04/orbis-terrarum-el-crculo-de-la-tierra.html

2. Image. I.1.2 Mapa T en O de la edad media http://www.lablaa.org/blaavirtual/exhibiciones/historia-natural-politica/hnp-04.html

3. Imagen. I.1.3 Carta de navegación de portulano http://www.temporamagazine.com/el-mapa-potulano-y-la-navegacion-en-la-edad-moderna/

4. Imagen I.1.3 Orbis Terrae Compendiosa Descriptio http://www.letraherido.com/170203historiacartografia.htm#3

5. Imagen I.2.1 Proyección azimutal

https://arquimedes.matem.unam.mx/puemac/PUEMAC_2008/mapas/imagenes/otros/estereo.gif

6. Imagen I.2.2 Proyección cónica.

https://arquimedes.matem.unam.mx/puemac/PUEMAC_2008/mapas/imagenes/otros/lambconcon.gif

7. Imagen I.2.3 Proyección Cilíndrica.

https://arquimedes.matem.unam.mx/puemac/PUEMAC_2008/mapas/imagenes/otros/millercil.gif

Fragmentos de código, Capturas, mapas, diagramas y diagramas de flujo y figuras no mencionadas

Todos son de creación propia y pertenecientes a este proyecto

La codificación completa relacionada con este proyecto puede ser consultada en:

https://github.com/lestat4415

Anexos

Las aplicaciones pueden ser clonadas y consultadas desde las siguientes ligas

Aplicación Android: https://github.com/lestat4415/myMapAndroid

Aplicación Web: https://github.com/lestat4415/myMapReact

Api de servicios y lambdas de AWS: https://github.com/lestat4415/myMapGo

El sitio web puede ser consultado en: https://mymapt.netlify.app/

La aplicación puede ser descargada en:

https://drive.google.com/drive/folders/1j8ZerIOa_7l6b7fwBe2LT_9XxiTpysYz?usp=sharing

Se agregan las clases más importantes de cada aplicación.

Aplicación Android

MapScreen.kt

```
@Composable
fun MapScreen(
    navigateToFormLocationScreen: (lat: String, lng: String) -> Unit,
    mapViewModel: MapViewModel
) {

    LaunchedEffect(Unit) {
        mapViewModel.initViewModel()
    }

    Scaffold(
        floatingActionButton = { MyFAB(navigateToFormLocationScreen, mapViewModel) },
        floatingActionButtonPosition = FabPosition.Start
    ) {
        MapaGoogle(viewModel = mapViewModel, paddingValues = it)
    }

}

@Composable
fun MapaGoogle(viewModel: MapViewModel, paddingValues: PaddingValues) {

    val ubicacionActual by viewModel.ubicacionActual.collectAsState()
```

```
val locationClicked by viewModel.ubicationClicked.collectAsState()

val currentPositions: List<Positions> by viewModel.positions.observeAsState(
    initial = emptyList()
)

val cameraPositionState = rememberCameraPositionState()

var selectedMarker by remember { mutableStateOf<Positions?>(null) }

// Mover la cámara a la ubicación actual
LaunchedEffect(ubicacionActual) {
    Log.d("MapScreen", "ubicacionActual: $ubicacionActual")
    ubicacionActual.let {
        cameraPositionState.position =
            CameraPosition.fromLatLngZoom(it ?: viewModel.defaultLocation, 15f)
    }
}

GoogleMap(
    modifier = Modifier
        .fillMaxSize()
        .padding(paddingValues),
    cameraPositionState = cameraPositionState,
    onMapClick = {
        viewModel.updateLocationClicked(it)
    }
) {

    UserCurrenPositionMarker(
        fullName = "You",
        location = ubicacionActual ?: LatLng(19.42847, -99.12766)
    ) {

    }

    locationClicked?.let {
        Marker(
```

```
                state = MarkerState(position = it), // Ejemplo de posición
                icon = pointNewPosition(Astral),
                title = "Nueva Posición",
                snippet = ""
            )
        }

        currentPositions.forEach { position ->
            MarkerInfoWindow(
                icon = BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_BLUE),
                state = MarkerState(position = LatLng(position.lat, position.long))
            ) {
                MarkerInfoWindowCustom(position)
            }
        }
    }
}

@Composable
fun UserCurrenPositionMarker(
    fullName: String,
    location: LatLng,
    onClick: () -> Unit
) {
    val markerState = remember { MarkerState(position = location) }
    val shape = RoundedCornerShape(20.dp, 20.dp, 20.dp, 0.dp)

    MarkerComposable(
        keys = arrayOf(fullName),
        state = markerState,
        title = fullName,
        anchor = Offset(0.5f, 1f),
        onClick = {
            onClick()
            true
        }
    ) {
        Box(
```

```
            modifier = Modifier
                .size(48.dp)
                .clip(shape)
                .background(Blue)
                .padding(4.dp),
            contentAlignment = Alignment.Center,
        ) {
            Image(
                modifier = Modifier.fillMaxSize(),
                painter = painterResource(id = R.drawable.ic_profile),
                contentDescription = "Profile Image",
                contentScale = ContentScale.Crop
            )
        }
    }
}

@Composable
fun pointNewPosition(myColor: Color): BitmapDescriptor {
    // Crear un Bitmap desde un Canvas
    val bitmap = Bitmap.createBitmap(100, 100, Bitmap.Config.ARGB_8888)
    val canvas = Canvas(bitmap)

    // Crear el pin
    val paint = Paint().apply {
        color = myColor.toArgb()
        isAntiAlias = true
    }

    // Dibujar un círculo (el pin)
    canvas.drawCircle(50f, 50f, 40f, paint)

    // Crear el descriptor del pin
    return BitmapDescriptorFactory.fromBitmap(bitmap)
}


@Composable
```

```
fun MyFAB(
    navigateToFormLocationScreen: (lat: String, lng: String) -> Unit,
    mapViewModel: MapViewModel
) {

    val mContext = LocalContext.current

    FloatingActionButton(
        onClick = {
            if (mapViewModel.ubicationClicked.value != null) {
                navigateToFormLocationScreen(
                    mapViewModel.ubicationClicked.value?.latitude.toString(),
                    mapViewModel.ubicationClicked.value?.longitude.toString()
                )
            } else {
                mToast(context = mContext, "Selecciona una ubicación en el mapa")
            }
        },
        containerColor = Astral,
        contentColor = White
    ) {
        Icon(imageVector = Icons.Filled.Add, contentDescription = "Add")
    }
}
```

MainActivity.kt

```
@AndroidEntryPoint
class MainActivity : ComponentActivity() {

    private lateinit var navHostController: NavHostController

    private val mapViewModel: MapViewModel by viewModels()

    private val formViewModel: FormLocationViewModel by viewModels()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
```

```
        enableEdgeToEdge()
        setContent {

            navHostController = rememberNavController()

            GeomaticAppTheme {
                MapaApp(
                    navHostController = navHostController,
                    mapViewModel = mapViewModel,
                    formViewModel = formViewModel
                )
            }
        }
        obtenerUbicacionActual()
    }

    @SuppressLint("MissingPermission")
    private fun obtenerUbicacionActual() {
        val fusedLocationProviderClient = LocationServices.getFusedLocationProviderClient(this)
        fusedLocationProviderClient.lastLocation.addOnSuccessListener { location ->
            if (location != null) {
                val currentLatLng = LatLng(location.latitude, location.longitude)
                lifecycleScope.launch {
                    mapViewModel.actualizarUbicacionActual(currentLatLng)
                }
            }
        }
    }
}

@Composable
fun MapaApp(
    navHostController: NavHostController,
    mapViewModel: MapViewModel,
    formViewModel: FormLocationViewModel
) {
    var permisoConcedido by remember { mutableStateOf(false) }
    val solicitarPermisos = rememberLauncherForActivityResult(
```

```
        contract = ActivityResultContracts.RequestPermission()
    ) { isGranted ->
        permisoConcedido = isGranted
    }

    LaunchedEffect(Unit) {
        solicitarPermisos.launch(Manifest.permission.ACCESS_FINE_LOCATION)
    }

    if (permisoConcedido) {
        NavigationWrapper(
            navHostController = navHostController,
            mapViewModel = mapViewModel,
            formViewModel = formViewModel
        )
    }
}
```

FormLocationScreen.kt

```
@Composable
fun FormLocationScreen(
    backPressed: () -> Unit,
    lat: String?,
    lon: String?,
    viewModel: FormLocationViewModel
) {

    val name: String by viewModel.name.observeAsState(initial = "")

    val description: String by viewModel.description.observeAsState(initial = "")

    val type: Int by viewModel.type.observeAsState(initial = -1)

    val isButtonEnabled by viewModel.isAddLocationEnable.observeAsState(initial = false)

    val lifecycle = LocalLifecycleOwner.current.lifecycle
```

```
var showDialogSuccess by rememberSaveable {
    mutableStateOf(false)
}

val uiState by produceState<UiState>(
    initialValue = UiState.Start,
    key1 = lifecycle,
    key2 = viewModel
) {
    lifecycle.repeatOnLifecycle(state = Lifecycle.State.STARTED) {
        viewModel.uiState.collect {
            value = it
        }
    }
}

Scaffold(modifier = Modifier
    .fillMaxSize()
    .background(AllWhite), topBar = {
    MyTopAppBar(backPressed)
}) {
    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(paddingValues = it)
            .background(AllWhite),
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        MyTextField(hint = "Nombre de la posición", value = name, isEnabled = true) {
            viewModel.onRequestDataChanged(name = it, description = description, type = type)
        }
        MyTextField(hint = "Latitud de la posición", value = lat ?: "", isEnabled = false) {

        }
        MyTextField(hint = "Longitud de la posición", value = lon ?: "", isEnabled = false) {

        }
        DescriptionTextField(value = description) {
```

```
    viewModel.onRequestDataChanged(name = name, description = it, type = type)

}
SpinnerDemo {
    viewModel.onRequestDataChanged(name = name, description = description, type = it)
}
MyButton(text = "Crear posición", isEnabled = isButtonEnabled) {
    if (lat != null && lon != null) {
        val request = AddPositionRequest(
            coordinates = CoordinatesRequest(
                lat = lat.toDouble(),
                long = lon.toDouble()
            ),
            name = name,
            description = description,
            type = type
        )
        Log.d("FormLocationScreen", "request: $request")
        viewModel.addPosition(
            request = request
        )
    } else {
        //Do Something
    }
}

when(uiState){
    UiState.Loading -> {

    }
    UiState.Success -> {
        showDialogSuccess = true
        PositionCreatedDialog(show = showDialogSuccess, onDismiss = {
            backPressed()
            showDialogSuccess = false
        })
    }
    UiState.Error -> {
```

```
                //Mostrar toast error
            }
        }

    }
  }
}

@Composable
fun MyButton(text: String, isEnabled: Boolean, onClickedButton: () -> Unit) {
    Button(modifier = Modifier
        .fillMaxWidth()
        .padding(horizontal = 16.dp),
        enabled = isEnabled,
        colors = ButtonDefaults.buttonColors(
            disabledContainerColor = DisableContainer,
            disabledContentColor = DisableContent,
            containerColor = Tradewind,
            contentColor = White
        ),
        onClick = {
            onClickedButton()
        }) {
        Text(text = text)
    }
}

@Composable
fun DescriptionTextField(value: String, onTextChanged: (String) -> Unit) {
    OutlinedTextField(
        modifier = Modifier
            .fillMaxWidth()
            .height(150.dp)
            .padding(vertical = 12.dp, horizontal = 16.dp),
        value = value,
        onValueChange = { onTextChanged(it) },
        placeholder = { Text(text = "Escribe una descripción...") },
        label = { Text(text = "Escribe una descripción...") },
```

```
        maxLines = 5,
        keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Text),
        colors = OutlinedTextFieldDefaults.colors(
            focusedContainerColor = White,
            focusedBorderColor = DarkAstral,
            focusedLabelColor = DarkAstral,
            focusedPlaceholderColor = DarkAstral,
            focusedTextColor = DarkAstral,
            unfocusedContainerColor = DefaultGrayContainer,
            unfocusedBorderColor = DefaultGrayContainer,
            unfocusedLabelColor = DisableContent,
            unfocusedPlaceholderColor = DisableContent,
            unfocusedTextColor = DisableContent,
            disabledContainerColor = DisableContainer,
            disabledBorderColor = DisableContainer,
            disabledLabelColor = DisableContent,
            disabledPlaceholderColor = DisableContent,
            disabledTextColor = DisableContent
        )
    )
}

@Composable
fun MyTextField(hint: String, value: String, isEnabled: Boolean, onTextChanged: (String) -> Unit) {
    OutlinedTextField(
        modifier = Modifier
            .fillMaxWidth()
            .padding(vertical = 12.dp, horizontal = 16.dp),
        value = value,
        enabled = isEnabled,
        onValueChange = { onTextChanged(it) },
        maxLines = 1,
        label = {
            Text(text = hint)
        },
        placeholder = {
            Text(text = hint)
        },
```

```
        singleLine = true,
        keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Text),
        colors = OutlinedTextFieldDefaults.colors(
            focusedContainerColor = White,
            focusedBorderColor = DarkAstral,
            focusedLabelColor = DarkAstral,
            focusedPlaceholderColor = DarkAstral,
            focusedTextColor = DarkAstral,
            unfocusedContainerColor = DefaultGrayContainer,
            unfocusedBorderColor = DefaultGrayContainer,
            unfocusedLabelColor = DisableContent,
            unfocusedPlaceholderColor = DisableContent,
            unfocusedTextColor = DisableContent,
            disabledContainerColor = DisableContainer,
            disabledBorderColor = DisableContainer,
            disabledLabelColor = DisableContent,
            disabledPlaceholderColor = DisableContent,
            disabledTextColor = DisableContent
        )

    )
}


@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun SpinnerDemo(onOptionChanged: (Int) -> Unit) {
    val items = TypesLocation.getListOptions()

    var expanded by remember { mutableStateOf(false) }
    var selectedOption by remember { mutableStateOf(items[0]) }

    ExposedDropdownMenuBox(
        modifier = Modifier.padding(vertical = 12.dp),
        expanded = expanded,
        onExpandedChange = { expanded = !expanded }
    ) {
        TextField(
```

```
    value = selectedOption.nameLocation,
    onValueChange = {},
    readOnly = true,
    modifier = Modifier
        .menuAnchor()
        .fillMaxWidth()
        .padding(horizontal = 16.dp),
    label = { Text("Selecciona el tipo de localización") },
    trailingIcon = { ExposedDropdownMenuDefaults.TrailingIcon(expanded = expanded) },
    colors = ExposedDropdownMenuDefaults.textFieldColors(
        focusedContainerColor = White,
        focusedLabelColor = DarkAstral,
        focusedPlaceholderColor = DarkAstral,
        focusedTextColor = DarkAstral,
        unfocusedContainerColor = DefaultGrayContainer,
        unfocusedLabelColor = DisableContent,
        unfocusedPlaceholderColor = DisableContent,
        unfocusedTextColor = DisableContent,
        cursorColor = Color.Transparent,
        unfocusedIndicatorColor = Color.Transparent,
        focusedIndicatorColor = Color.Transparent
    )
)

ExposedDropdownMenu(
    modifier = Modifier.padding(horizontal = 16.dp),
    containerColor = White,
    expanded = expanded,
    onDismissRequest = { expanded = false }
) {
    items.forEach { option ->
        DropdownMenuItem(
            text = {
                Text(
                    text = option.nameLocation,
                    color = DisableContent
                )
            },
```

```
                onClick = {
                    selectedOption = option
                    onOptionChanged(option.typeLocation)
                    expanded = false // Close the menu
                }
            )
        }
    }
  }
}

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun MyTopAppBar(onClickIcon: () -> Unit) {
  TopAppBar(
    title = {
      Text(
        text = "Agregar Posición"
      )
    }, colors = TopAppBarDefaults.topAppBarColors(
      containerColor = Tradewind,
      titleContentColor = AllWhite,
      navigationIconContentColor = AllWhite
    ),
    navigationIcon = {
      IconButton(onClick = { onClickIcon() }) {
        Icon(imageVector = Icons.Filled.ArrowBack, contentDescription = "back")
      }
    }
  )
}
```

Aplicación Web

map.jsx

```
export const MapT = (()=>{
```

```
  const [markers, setMarkers] = useState([])

  const positions = async() => {
    const points = await getPositions();
    setMarkers(points)
  }

  useEffect(()=>{
    positions()
  },[])

  if(markers.length>0){

  }

  console.log("revisar el punto aqui"+markers[13])
  console.log("revisar el punto aqui"+markers)

  return (
    <div className='map'>
      <Map
        defaultZoom={11}
        defaultCenter={ { lat: 19.45, lng: -99.208138 } }
        mapId={"f9d900b718b6019b"}
      >
        <>
          {
            markers.map(marker =>(
              <MarkerWithInfoWindow point={marker}/>
            ))
          }

        </>
      </Map>
    </div>
  )
})
```

markerWithInfoWindow.jsx

```
export const MarkerWithInfoWindow = ({point}) => {



  const [markerRef, marker] = useAdvancedMarkerRef();


  const [infoWindowShown, setInfoWindowShown] = useState(false);


  const handleMarkerClick = useCallback(
    () => setInfoWindowShown(isShown => !isShown),
    []
  );


  const handleClose = useCallback(() => setInfoWindowShown(false), []);


  return (


    <>
      <AdvancedMarker
        key={point.id}
        position={{lat:point.lat,lng:point.long}}
        ref={markerRef}
        onClick={handleMarkerClick}
        title="Hola marker"


      >
        <Pin
          background={"#3a8e9c"}
          glyphColor={"#3a8e9c"}
          borderColor={"#3a8e9c"}


        ></Pin>
      </AdvancedMarker>
      {infoWindowShown && (
```

```
        <InfoWindow
          anchor={marker}
          onClose={handleClose}
        >
          {GetHTMLFromTypeId(point)}

        </InfoWindow>
      )}
    </>

  );

};
```

utilils.jsx

```
export const GetHTMLFromTypeId = (marker ) => {
   console.log(marker.typeId)
   console.log(marker)
  switch(marker.typeId){

      case 1:
         /* Escuela */
         return (
            <>
            <body bgcolor="#2196f3">

               <center><h2>{marker.name}</h2></center>

               <div className='row'

               >
                  <div  className='column'>
                  <img src="/school.png"/>
                  </div>
                  <div className='column'>
                     <p><e>Coordenadas:</e><br/> {marker.lat}, {marker.long}</p>
                     <p><e>Descripción:</e><br/> {marker.description}</p>
```

```
                <p><e>Tipo:</e><br/> Escuela</p>
            </div>
        </div>

    </body>
    </>
  )
case 2:{
  /* Parque */
  return (
    <>
    <body bgcolor="#4caf50">

      <center><h2>{marker.name}</h2></center>

      <div className='row'

      >
        <div  className='column'>
        <img src="/park.png"/>
        </div>
        <div className='column'>
          <p><e>Coordenadas:</e><br/> {marker.lat}, {marker.long}</p>
          <p><e>Descripción:</e><br/> {marker.description}</p>
          <p><e>Tipo:</e><br/> Parque</p>
        </div>
      </div>

    </body>
    </>
  )
}
case 3:{
  /* Iglesia */
  return (
    <>
    <body bgcolor="#795548">
```

```
            <center><h2>{marker.name}</h2></center>

            <div className='row'

            >
               <div  className='column'>
               <img src="/church.png"/>
               </div>
               <div className='column'>
                  <p><e>Coordenadas:</e><br/> {marker.lat}, {marker.long}</p>
                  <p><e>Descripción:</e><br/> {marker.description}</p>
                  <p><e>Tipo:</e><br/> Iglesia</p>
               </div>
            </div>

         </body>
         </>
      )
}
case 4:{
   /* comercio */
   return (
      <>
      <body bgcolor="#FFC107">

         <center><h2>{marker.name}</h2></center>

         <div className='row'

         >
            <div  className='column'>
            <img src="/shop.png"/>
            </div>
            <div className='column'>
               <p><e>Coordenadas:</e><br/> {marker.lat}, {marker.long}</p>
               <p><e>Descripción:</e><br/> {marker.description}</p>
               <p><e>Tipo:</e><br/> Comercio</p>
            </div>
```

```
            </div>

        </body>
        </>
      )
    }
  }

}
```

index.html

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/vite.svg" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>MyMap</title>
  </head>
  <body>
    <div id="app"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

Aplicación de servicios y lambdas de aws

AddPoint.go

```
func AddPoint(ctx context.Context) models.RespApi {

        var point models.Point
        var response models.RespApi

        responsePoint := responses.RegisterPointResponse{
                StatusCode: -1,
                Message:    "Ocurrió un error al almacenar el punto",
        }

        response.Status = 400
```

```
body := ctx.Value(models.Key("body")).(string)
err := json.Unmarshal([]byte(body), &point)

if err != nil {
        response.Message = err.Error()
        fmt.Println(response.Message)
        return response
}

if point.Coordinates == (models.Coordinates{}) {
        responsePoint.Message = "Las coordenadas son obligatorias"
        response.Status = 200
        response.Data = responsePoint
        return response
}

if len(point.Description) == 0 {
        responsePoint.Message = "La descripción es obligatorias"
        response.Status = 200
        response.Data = responsePoint
        return response
}

if len(point.Name) == 0 {
        responsePoint.Message = "El nombre es obligatorias"
        response.Status = 200
        response.Data = responsePoint
        return response
}

if point.TypeId == nil {
        responsePoint.Message = "El tipo de punto es obligatorias"
        response.Status = 200
        response.Data = responsePoint
        return response
}
```

```
	_, _, err = bd.InsertPoint(point)

	if err != nil {
		responsePoint.Message = "Ocurrió un error en la base de datos"
		response.Status = 200
		response.Data = responsePoint
		return response
	}

	responsePoint.StatusCode = 0
	responsePoint.Message = "Ok"

	response.Status = 200
	response.Data = responsePoint
	return response
}
```

GetAllPositions.go

```
func GetAllPositions() models.RespApi {
	responsePoints := responses.GetAllPositionResponse{
		StatusCode: -1,
		Message:    "Ocurio un error",
	}
	var response models.RespApi

	positions, success := bd.GetAllPositions()

	if !success {
		responsePoints.Message = "No se encontraron datos"
		response.Status = 200
		response.Data = responsePoints
		return response
	}

	if len(positions) <= 0 {
		responsePoints.Message = "No se encontraron datos registrados"
		response.Status = 200
```

```
                response.Data = responsePoints
                return response
        }

        responsePoints.Message = "Ok"
        responsePoints.Points = positions
        responsePoints.StatusCode = 0
        response.Status = 200
        response.Data = responsePoints
        return response

}
```

Handlers.go

```
func Handlers(ctx context.Context, request events.APIGatewayProxyRequest) models.RespApi {
        fmt.Println(ctx.Value(models.Key("path")).(string))
        fmt.Println("Procensing " + ctx.Value(models.Key("path")).(string) + ">>>>" + ctx.Value(models.Key("method")).(string))

        r := models.RespApi{
                Status:     400,
                Message:    "Method Invalid",
                CustomResp: nil,
        }

        switch ctx.Value(models.Key("method")).(string) {
        case "POST":
                switch ctx.Value(models.Key("path")).(string) {
                case "addPosition":
                        return routers.AddPoint(ctx)

                }

        case "GET":
                switch ctx.Value(models.Key("path")).(string) {
                case "getAllPositions":
                        return routers.GetAllPositions()
                }
```

```
	}

	return r
}
```

StartAWS.go

```
func StartAWS() {
	Ctx = context.TODO()

	Confg, err = config.LoadDefaultConfig(Ctx, config.WithDefaultRegion("us-east-1"))
	if err != nil {
		panic("Error al cargar la configuración en aws.go" + err.Error())
	}
}
```