

Universidad Nacional del Litoral

Facultad de Ingeniería y Ciencias Hídricas

Proyecto Final de Carrera



DESARROLLO DE UNA APLICACIÓN EN
ANDROID PARA UBICAR PUNTOS DE INTERÉS
EN LAS DEPENDENCIAS DE LA UNL

Autor: Lautaro Sikh

Director: Emmanuel Rojas Fredini

diciembre de 2018

Agradecimientos

Quiero agradecer a todas aquellas personas que fueron parte de este proceso durante estos 7 años.

A mi familia en primer lugar y a los amigos que hice en la carrera ya que sin ellos hoy sería imposible estar escribiendo estas palabras.

A todos los profesores que he tenido, que en mayor o menor medida han contribuido a que este día llegara, junto con los miembros de la Universidad y nuestra facultad. Principalmente a la cátedra de Programación de Dispositivos Móviles del año 2016 quienes me brindaron la idea para este proyecto y diseñamos la primera propuesta de solución y destaco a Emmanuel Rojas Fredini que me acompañó director en esta oportunidad.

A la Universidad Tecnológica de Pereira (Colombia) y sus autoridades que me brindaron la posibilidad de compartir un semestre en otro país y conocer una nueva cultura que hoy hacen a mi formación personal y profesional.

También quiero agradecer al personal del área de Construcciones de la Universidad, que muy atentamente me brindaron los planos actualizados de las diferentes unidades académicas en las que he trabajado con mi proyecto.

Por último, a Pablo de Jesus y Felipe Lang quienes desarrollaron algunas de las librerías que utilicé para este proyecto, para la empresa TopGroup S.A, donde me desempeñe como desarrollador durante el transcurso del proyecto.

Prefacio

En este trabajo se presentará el camino recorrido desde la concepción hasta el diseño y desarrollo del proyecto en cuestión, que es requisito para aprobar la asignatura Proyecto Final de Carrera, para optar por el título de grado de la carrera de Ingeniería en Informática de la Facultad de Ingeniería y Ciencias Hídricas de la Universidad Nacional del Litoral.

Aquí detallaremos las motivaciones que dieron origen al mismo, las investigaciones realizadas, los aspectos contemplados, tecnologías utilizadas, decisiones de diseño y desarrollo técnico, resultados y conclusiones finales.

También dejaremos un anexo con documentación, informes y recursos que se han utilizado a lo largo de este proyecto.

En el Capítulo 1 presentaremos las motivaciones que llevaron a dar con este proyecto y el impacto que puede tener en la sociedad académica. También enunciamos los objetivos a cumplir y el alcance definido.

En el Capítulo 2 proponemos establecer un marco teórico en función de explicar o introducir conceptos claves de este proyecto y que es importante que el lector conozca para la lectura posterior, debido a que se han aplicado cuestiones de los mismos y que devienen del análisis realizado en cuanto a la problemática y el enfoque propuesto para su solución.

En el Capítulo 3 se hará referencia a todo lo relativo al diseño y desarrollo del Web Service en función de los conceptos presentados en el capítulo anterior, incorporando conceptos más técnicos como ser la base de datos y los frameworks utilizados, entre otras cosas.

En el Capítulo 4 se hará lo propio en el lado cliente especificando el proceso de diseño y desarrollo de la aplicación junto con las tecnologías utilizadas y una noción de los componentes utilizados.

En el Capítulo 5 se mostrarán los resultados obtenidos de forma general acompañando con imágenes de diversos casos de prueba.

Por último, en el Capítulo 6 se mencionarán las conclusiones abordadas y propuestas de mejoras para futuros desarrollos, así como también algunos comentarios finales y consideraciones.

Resumen

La Universidad Nacional del Litoral posee una gran cantidad de dependencias académicas donde circulan una gran cantidad de alumnos, docentes, no docentes y personas relacionadas con la Universidad que concurren a clases, cursos, eventos y otras actividades. Muchas de estas personas no tienen conocimiento específico de hacia dónde deben dirigirse si tenemos en cuenta la gran cantidad de aulas, oficinas o servicios que posee la UNL.

Este proyecto consta de realizar una aplicación para dispositivos móviles bajo la tecnología Android, que a partir de los sensores de ubicación de los mismos y la conexión a Internet, ubique al usuario en alguna de las dependencias de la UNL apoyado en la visualización de Google Maps y le muestre el camino hacia un aula, oficina, biblioteca, baño, edificio u otros puntos de interés que esté buscando.

Una vez ubicada la posición física del usuario, se calculará el camino entre él y el punto de destino para ser mostrado en Google Maps interactuando con su API. Para complementar la visualización se utilizarán los planos de planta de los edificios en sus diferentes plantas e imágenes de los lugares en donde están los nodos por donde se trazó el camino.

Con el objetivo de independizar los datos que representan la lógica del problema de la aplicación, se desarrollará un Web Service donde se podrán consumir las distintas categorías de búsqueda, pero que también será el encargado de calcular el camino óptimo por el que debe pasar el usuario.

Además de esto se propone utilizar otras herramientas como códigos QR que harán de tokens de ubicación para localizar más fácilmente al usuario. Estos pueden ser leídos con la cámara del móvil y contienen la latitud y longitud de dónde estén ubicados y el piso en el caso de estar dentro de un edificio.

También se implementará la posibilidad de que el usuario pueda ver imágenes reales de los lugares por donde circula, relativas a aulas o puntos destacados de la Universidad. Estas imágenes son provistas por el Web Service.

PALABRAS CLAVES: aplicación, web service, ubicación, puntos de interés, geo posicionamiento, planos, QR.

Índice de contenidos

Prefacio	5
Resumen	7
1 Introducción	14
1.1 Motivación	14
1.2 Objetivos del proyecto	17
1.2.1 Objetivo general.....	17
1.2.2 Objetivos específicos	17
1.3 Alcance.....	18
1.4 Recursos utilizados	18
1.4.1 Hardware	18
1.4.2 Software.....	19
1.4.3 Recursos humanos.....	19
2 Análisis y marco teórico	21
2.1 Análisis del problema	21
2.1.1 Requerimientos funcionales.....	21
2.1.2 Requerimientos no funcionales.....	22
2.1.3 Enfoque de la solución diseñada	22
2.2 Elecciones de implementación	24
2.2.1 Lenguaje de programación	25
2.2.2 Visualización de mapas	27
2.3 Conceptos del lado servidor	28
2.3.1 Arquitectura.....	28
2.3.2 Protocolo HTTP	30
2.3.3 Formato de respuesta.....	31
2.3.4 Paradigma orientado a objetos	32
2.3.5 Grafos	33
2.3.6 Servlet.....	36
2.4 Conceptos del lado cliente	37
2.4.1 Sistema operativo Android.....	37
2.4.2 Interfaz de programación de aplicaciones	38

2.4.3	Código QR	38
2.4.4	Sensores de un smarthphone	39
3	Diseño y desarrollo del lado servidor	42
3.1	Implementación de la base de datos	42
3.1.1	Gestor de base de datos.....	42
3.1.2	Modelado del problema	43
3.1.3	Planos de plantas	44
3.1.4	Relevamiento de datos y carga.....	45
3.2	Desarrollo del Web Service	46
3.2.1	Proyecto Maven	47
3.2.2	Mapeo de objetos	49
3.2.3	Modelo por capas	52
3.2.4	Instalación de la aplicación en el contenedor	58
3.2.5	Servicios	59
4	Diseño y desarrollo de la aplicación.....	62
4.1	Nociones del proyecto Android.....	62
4.1.1	Estructura del proyecto	62
4.1.2	Ciclo de vida de una aplicación.....	64
4.1.3	Elementos de interacción con el usuario	65
4.1.4	Comunicación asíncrona	66
4.2	API de Google Maps	66
4.2.1	Markers.....	67
4.2.2	Overlays.....	68
4.2.3	Polilínea	68
4.3	Interfaz de usuario.....	69
4.4	Geo posicionamiento.....	69
4.5	Modelado de la aplicación	70
4.5.1	Flujo principal.....	71
4.5.2	Visualización de imágenes	75
4.5.3	Lectura de tokens.....	77
4.5.4	Base de datos	77
5	Resultados	80

5.1	Casos de prueba	80
6	Conclusiones y trabajos futuros	86
6.1	Conclusiones.....	86
6.2	Trabajos futuros	87
6.3	Comentarios finales.....	88
	Bibliografía.....	90
	Anexo	93

Índice de figuras

Imagen 1.1: Tercer piso del MSQ visto desde Google Maps	15
Imagen 2.1: Diagrama de actividad de la aplicación	25
Imagen 2.2: Máquina virtual de Java.....	27
Imagen 2.3: Arquitectura Cliente - Servidor.....	29
Imagen 2.4: Esquema de una URL HTTP	31
Imagen 2.5: Comparación entre XML y JSON.....	32
Imagen 2.6: Grafo de 4 nodos	33
Imagen 2.7: Recorrido de grafo en profundidad y anchura	35
Imagen 2.8: Ejemplo de código QR.....	39
Imagen 3.1: Diseño de la base de datos	43
Imagen 3.2: Estructura básica de un proyecto Maven.....	48
Imagen 3.3: Diagrama de clases del modelo.....	49
Imagen 3.4: Inversión de Control.....	53
Imagen 3.5: Diagrama de clases del Web Service	55
Imagen 3.6: Diagrama de secuencia de una petición	56
Imagen 3.7: Tendencia de uso de servidores en 2015.....	58
Imagen 4.1: Estructura de proyecto.....	63
Imagen 4.2: Ciclo de vida de una aplicación	65
Imagen 4.3: Marker	68
Imagen 4.4: Diagrama de clases de la aplicación	70
Imagen 4.5: UI de la aplicación	71
Imagen 4.6: Pantalla de búsqueda	72
Imagen 4.7: Diagrama de secuencias de la aplicación.....	73
Imagen 4.8: Representación de una búsqueda	75
Imagen 4.9: Marcador con imagen	76
Imagen 4.10: Camino con marcadores intermedios	76
Imagen 4.11: Pantalla de últimas búsquedas.....	78
Imagen 5.1: Búsqueda de una biblioteca	81
Imagen 5.2: Búsqueda de un laboratorio.....	82
Imagen 5.3: Aula 6 en Nave de Hidráulica	82
Imagen 5.4: Búsqueda en FCM.....	83
Imagen 5.5: Búsqueda de baños.....	84

- Capítulo 1 -

1 Introducción

En esta sección presentaremos las motivaciones que llevaron a la concepción de este proyecto junto con los objetivos y el alcance que se pretendió lograr. Introduciremos el contexto y dejaremos argumentos del porqué de la importancia del mismo.

1.1 Motivación

Actualmente la Universidad Nacional del Litoral cuenta con 10 unidades académicas, 3 centros universitarios, el edificio de Rectorado, un aulario común, un predio, colegios secundarios y escuelas primarias. La mayoría de ellos ubicados en la Ciudad Universitaria, otros dispersos en distintos puntos de la Ciudad de Santa Fe y el resto en otras ciudades.

Sólo en el corriente año, más de 8.000 chicos y chicas se matricularon a la Universidad para comenzar sus estudios y se sumaron a las otras 40.000 personas que ya eran alumnos regulares¹. Muchos de estos son oriundos de otras ciudades y llegan por primera vez a Santa Fe sin mucho conocimiento acerca de las instalaciones de la Universidad. Tampoco hay que dejar de mencionar que estos números crecen año a año, cada vez son más los adolescentes que eligen estudiar en la misma. Además de ello, en las dependencias de la UNL se suelen realizar múltiples eventos de capacitación, divulgación científica, foros, charlas, entre otros; a los cuales acuden muchas personas que ni siquiera están en relación directa con la Universidad. Y tampoco hay que dejar de mencionar que todos los alumnos de la UNL deben completar el cursado de una materia ajena a su carrera, y por lo tanto a su unidad académica en la mayoría de los casos, para finalizar sus estudios.

Lo que se quiere exponer aquí es que por los pasillos de la Universidad y sus instalaciones constantemente circulan muchas personas que no tienen exacto conocimiento de hacia donde se tienen que dirigir para cursar una materia, realizar un curso, hacer trámites en alguna oficina, encontrar un baño o una biblioteca; esto obliga

¹ "UNL en Cifras (2015)", disponible en http://www.unl.edu.ar/categories/view/unl_en_cifras

a tener que buscar por Internet hacia qué unidad edilicia debe acudir, cómo llegar a ella, y una vez llegado preguntar a otras personas por indicaciones hasta encontrar el lugar que estaba buscando. Algunas unidades académicas tienen sus planos en cada piso del edificio, pero su ubicación no es de público conocimiento, sin mencionar que no discriminan qué es lo que se está viendo, simplemente es un plano del edificio: pasillos y paredes. Tampoco se debe dejar afuera del grupo de interesados a la gran cantidad de estudiantes, docentes e investigadores que participan de los programas de movilidad académica de la Universidad y llegan cada semestre a nuestro país a desarrollar diversas actividades.

En esta era digital se esperaría poder contar con una solución más tecnológica, que permita al usuario un poco de interacción con la cuestión, amigable visualmente y que se pueda ejecutar desde una PC o smartphone con toda la información a la mano, de forma centralizada.

Hoy en día Google Maps nos permite cargar a su sistema mapas internos de edificios para poder hacer un recorrido más enriquecido de los lugares, es el caso por ejemplo del *Madison Square Garden* que posee habilitados planos internos de sus 11 plantas, los cuales discriminan todo tipo de servicios que se encuentran en él: bares, restaurantes, baños, tiendas, etc., como se observa en la Imagen 1.1 del texto.



Imagen 1.1: Tercer piso del MSQ visto desde Google Maps

En este proyecto se propone utilizar otras técnicas para implementar una solución similar manipulando la API de Google Maps. Desarrollar una aplicación para dispositivos móviles que utilice Android, ya sea un smartphone o una tablet, que le

permita al usuario localizar algún punto de interés dentro de las dependencias de la UNL, que pueda visualizar el plano de la planta – o plantas si está en un piso superior– y le indique explícitamente el camino que tiene que tomar para llegar, permitiendo al usuario interactuar con otros elementos externos a la aplicación, como por ejemplo códigos QR que le indiquen dónde está parado o imágenes de los lugares que debería estar viendo durante el camino, esto le da al producto una componente de realidad aumentada que la hace más atractiva. Así, se podrá tener de forma fácil una herramienta que de forma centralizada ataque el problema antes mencionado.

Aproximadamente el 85% de los dispositivos móviles en la Argentina poseen activaciones con sistema operativo Android², lo que lo convierte en la plataforma más adecuada para desarrollar este proyecto ya que una gran parte de la comunidad estudiantil tendría acceso a él. Además, hoy en día cobran mucha importancia la implementación de micro servicios a los cuales se pueden acceder por peticiones URL o interacciones con API para consumir datos de forma independiente mediante el protocolo HTTP. No hay que dejar de mencionar tampoco la gran capacidad que tienen los móviles de interactuar con su entorno debido a los diversos sensores con los que están equipados. Por lo tanto, se propone incluir estas tecnologías al proyecto propuesto, para lograr un producto de software con las mayores prestaciones posibles, flexible a cambios implementando lo último del mercado.

Esto aportaría tanto a la comunidad de la UNL como a usuario externos que están indirectamente involucrados, un gran servicio de información que además representaría un verdadero valor agregado a las facilidades que ya posee la Universidad, por un lado porque no se tiene registro de otras instituciones que posean un servicio de este tipo o que tenga antecedentes en desarrollos similares lo cual significa innovar en este ámbito, y por el otro sería una herramienta muy útil y de fácil uso para cualquier interesado.

Pero además, deja la puerta abierta continuar con el trabajo en un futuro integrándola con otra de las aplicaciones para móviles que posee la UNL o complementando el servicio.

Es importante que toda esta información esté alojada en un servicio externo para que la aplicación sea independiente a los datos y no quede atada a constantes actualizaciones. Por lo tanto, al finalizar el proyecto, quedará disponible para futuros trabajos una base de datos que contendrá información acerca de todas las aulas, oficinas, baños, bibliotecas y otras instalaciones que posee la Universidad, que podría facilitar o incentivar a otras personas a realizar un desarrollo a partir de esta información.

² Más información en <https://www.gartner.com/en>

1.2 Objetivos del proyecto

Para este proyecto se definieron los siguientes objetivos generales y específicos, que marcaron el proceso del mismo

1.2.1 Objetivo general

- Diseñar y desarrollar una aplicación que permita al usuario conocer su ubicación actual dentro de las dependencias de la Universidad Nacional del Litoral y encontrar el camino hacia algún punto de interés interactuando con dicha herramienta.

1.2.2 Objetivos específicos

- Definir las funcionalidades y el diseño del producto.
- Investigar sobre la conveniencia de distintos tipos de implementaciones de Web Service para el caso.
- Generar una base de datos de uso abierto con información acerca de todos los puntos de interés de la Universidad y formas de acceder remotamente a ella.
- Lograr una buena representación de las dependencias de la Universidad mediante grafos.
- Probar distintos algoritmos de búsqueda para determinar la mejor relación costo eficiencia del camino obtenido de un punto a otro.
- Desarrollar un Web Service que contenga información acerca todos los puntos de interés relevados y se puedan consumir por peticiones URL.
- Explotar los sensores de ubicación del móvil y la conexión a Internet para darle sensación de realidad a la aplicación.
- Implementar soluciones para evitar posibles problemas de conectividad dentro de los edificios de la Universidad.

- Desarrollar funcionalidades que mejoren la experiencia del usuario.

1.3 Alcance

Entre los alcances del proyecto, se destaca que el mismo deberá poder ser utilizado por público de todas las edades, por lo tanto su interfaz gráfica debe ser simple y no debe prestar a confusiones o permitir un mal ingreso de datos del usuario.

Desde el punto de vista del Web Service el mismo debe tener alta disponibilidad y poder enviar los datos de forma rápida, por lo tanto será importante no cargar al mismo. Este solo contendrá los servicios que requieren la aplicación e imágenes de algunos puntos de interés que se pedirán por separado, no viajan con la información principal que utilizan las funcionalidades del aplicativo.

Quedan excluidos del producto aquellos dispositivos cuya versión de sistema operativo sea menor a Android 4.4, ya que no es posible asegurar la compatibilidad con las tecnologías que se utilizaron, además de que es una versión que está ampliamente soportada hoy en día. Tampoco entran dentro de la consideración aquellos que no puedan conectarse a Internet por alguna razón o no posean sensor de geo posicionamiento.

El límite establecido de dependencias de la UNL que serán relevadas en principio son todas las facultades para las cuales se cuenten con sus planos de plantas, con la posibilidad de agregarse otros en consideración, siendo también estos los censados para formar parte de la base de datos, por esto el ámbito de aplicación de este proyecto será la Universidad Nacional del Litoral.

En cuanto a las funcionalidades provistas, solo estará contemplado el módulo de búsqueda de puntos de interés, acompañado de la lógica de geo localización del usuario. El proceso de mantenimiento de la base de datos para realizar actualizaciones en la ubicación de los puntos de interés queda excluido como funcionalidad del proyecto. Dicha acción puede realizarse directamente desde el servidor de base de datos de forma manual.

1.4 Recursos utilizados

Los recursos utilizados para este proyecto fueron los siguientes:

1.4.1 Hardware

- Computadora Bangho Max i3 de propiedad personal, donde se desarrolló la aplicación, el Web Service, el servidor instalado localmente y la instancia de la base de datos.
- Teléfono LG K10, que es el punto de despliegue de la aplicación ya que cumple con todos los requisitos necesarios para esto.

1.4.2 Software

- Entorno de desarrollo de la aplicación: Android Studio
- Entorno de desarrollo del Web Service: Eclipse Oxygen
- Contenedor de Servlets: Apache Tomcat 8.0
- Sistema Operativo: Ubuntu 16.04 LTS

1.4.3 Recursos humanos

- Alumno ejecutor del proyecto
- Dirección del proyecto

Complementario a esto, se utilizaron otros recursos de diversas índoles. Los mismos son:

- Base de datos de puntos de interés: la misma se generó para este proyecto y quedará disponible para futuros usos.
- Planos de las dependencias de la UNL: otorgados en formato PDF por el departamento de Construcciones de la Universidad.
- Tokens para ubicación dentro de la UNL: contruidos en formato QR para este proyecto en función de la base de datos generada y quedarán disponibles para futuros usos.
- API Key de Google Maps: API Key gratuita para interactuar con la interfaz de Google Maps desde la aplicación.

- Capítulo 2 -

2 Análisis y marco teórico

En este capítulo presentaremos las bases teóricas del proyecto desde el punto de vista del Web Service y la aplicación y su relación con el diseño de solución propuesta. Dicha propuesta será resultado de los requerimientos extraídos en función de la problemática detectada.

También, se mencionarán algunos conceptos básicos mencionados en el inicio de la Introducción del Capítulo 1 y que sirven para comprender las decisiones de implementación que se tomaron, como ser el funcionamiento del protocolo HTTP (*Hypertext Transfer Protocol*), el formato JSON (*JavaScript Object Notation*), teoría de grafos y particularidades de la plataforma Android.

2.1 Análisis del problema

Por lo expuesto en el Capítulo 1 y en función del objetivo general mencionado anteriormente, se pretende que la herramienta desarrollada pueda cumplir con los siguientes puntos y se pueda elaborar una propuesta de acción en consecuencia.

2.1.1 Requerimientos funcionales

- La aplicación debe ubicar al usuario geográficamente.
- La aplicación debe situar el mapa en la posición del usuario y actualizar los cambios de posición detectados.
- El usuario debe poder consultar las categorías de búsqueda disponibles.
- El usuario debe poder seleccionar la unidad académica a la que pertenece la dependencia que quiere buscar.
- El usuario debe poder seleccionar que punto de interés quiere buscar.

- El sistema debe brindarle al usuario el camino que este debe recorrer para llegar a su destino.
- El usuario debe poder escanear códigos QR desde la aplicación y se actualice la posición actual.
- El usuario debe poder visualizar imágenes de los puntos por donde pasa.
- El usuario debe poder visualizar los planos de planta del edificio por donde debe circular.
- El usuario debe poder cambiar el piso de visualización mientras circula.

2.1.2 Requerimientos no funcionales

Además se identificaron algunos requerimientos no funcionales, algunos de ellos ya se anticiparon en el alcance del proyecto. Se dejan a continuación.

- El móvil debe contar con cámara de al menos 3 mega pixeles.
- El móvil debe contar con al menos 10mb libre de espacio de almacenamiento.
- La interfaz de la aplicación debe ser fácil de usar y no permitir ingreso de datos erróneos por parte del usuario.
- El móvil debe tener el GPS activado y conexión a Internet para apoyar la geo localización.
- El móvil debe contar con versión de sistema operativo Android 4.4 KitKat o superior para soportar las tecnologías utilizadas.

2.1.3 Enfoque de la solución diseñada

El flujo principal de actividades que se propuso para brindar una solución al problema planteado se puede observar a continuación. En la Imagen 2.1 se ilustra a

modo de ejemplo parte de la secuencia desde el punto de vista de la aplicación. En los capítulos siguientes se verá de qué manera interactúa con el servidor, luego de presentar los conceptos referidos al mismo.

- El usuario abre la aplicación. La misma le solicita permisos para utilizar los sensores, de ser necesario. La aplicación posiciona al usuario en el mapa que visualiza.
- El usuario, con el sensor de ubicación prendido y conexión a Internet, elige de la aplicación que tipo de dependencia está buscando y selecciona el edificio de la misma.
- La aplicación envía una petición GET al lado servidor que contendrá la información que seleccionó el usuario.
- El servidor atiende esta petición, y a partir de las clases de servicio, recupera una lista de las dependencias a devolver a la aplicación, previamente convirtiendo esta información en un JSON. Por ejemplo, si el usuario selecciono Aulas de FICH, el resultado que recibe del servidor es una lista que contendrá las claves: "Aula 1", "Aula 2", etc.
- El usuario selecciona una de estas opciones y oprime un botón que ejecuta la búsqueda.
- Se envía otra petición del tipo GET al servidor que contendrá: dependencia a ubicar (ej: "Aula 3 – FICH"), latitud y longitud del usuario y el piso donde se encuentre si este no fuera planta baja.
- El servidor recibe esta información y ubica al usuario en el grafo a partir de su posición, para saber en qué parte de la Universidad está.
- Se ejecuta un algoritmo de búsqueda hasta encontrar en el grafo la opción seleccionada por el usuario.
- Se genera un JSON con la información de todos los puntos por donde debe pasar el usuario hasta llegar al objetivo y se envía.
- La aplicación toma esta información, la vuelca en el mapa e identifica en qué edificios se encuentran los puntos recibidos y muestra los planos de planta en función del piso en donde esté el usuario. También mostrará imágenes en tiempo real por los puntos donde debe pasar el usuario que se cargaran bajo demanda.

- Si el punto objetivo estuviera en un piso distinto al que se encuentra el usuario, este podrá ir cambiando de plantas y visualizar cada una de ellas.

Paralelamente a esto, en cualquier punto de la ejecución del flujo principal, el usuario podrá leer un código QR desde la aplicación siempre y cuando cuente con una cámara digital en su dispositivo móvil. Esta acción actualizará la posición del usuario en el mapa en función del token leído.

Además, el sensor de ubicación del móvil estará censando todo el tiempo esperando cambios en la posición del usuario para actualizar la misma y reflejarla en la pantalla.

2.2 Elecciones de implementación

Por lo anterior, ahora se detallaran algunas de las decisiones de implementación que se tomaron en relación a los lenguajes y herramientas utilizadas y que permiten cumplir con los requerimientos mencionados adaptándose a la propuesta de solución.

Como se mencionó anteriormente, se debe generar una base de datos con la información de los puntos de interés de la Universidad, resta ahora establecer cómo consumir esa información. Esto motivó a diseñar e implementar un Web Service que se monte por encima de la base de datos y permita acceder a ella, aplicando la lógica de negocios pertinente.

Las razones para desarrollar nuestro propio Web Service se encuentran en poder disponer de una herramienta dedicada al proyecto que puede lograrse con baja infraestructura de hardware y, a raíz de las tecnologías elegidas que en la siguiente sección detallaremos, podía lograrse a partir de un desarrollo que se enmarcaba dentro del alcance, tiempo y costo de este proyecto. Esto permite definir los puntos de acceso a la base de datos que sean necesarios y cubran las necesidades del proyecto. De la misma forma, al poder desarrollar esta herramienta como parte del proyecto, podemos incorporarle toda la lógica de negocio necesaria.

Uno podría pensar que una solución alternativa sería persistir la base de datos en el dispositivo y almacenar la información de sus tablas en la memoria RAM del mismo, así no habría necesidad de desarrollar un servidor. Pero esto sería un despropósito pues la aplicación quedaría altamente cargada con información y sería la responsable de ejecutar algoritmos de búsqueda complejos. Para algunos dispositivos esta tarea podría ser trivial, pero para otros podría consumir una gran cantidad de recursos. Además, cada vez que se quisiera realizar una actualización, por ejemplo cambiar el nombre de un punto de interés, se debería reconstruir enteramente la APK de la aplicación y distribuirla a todos los interesados.

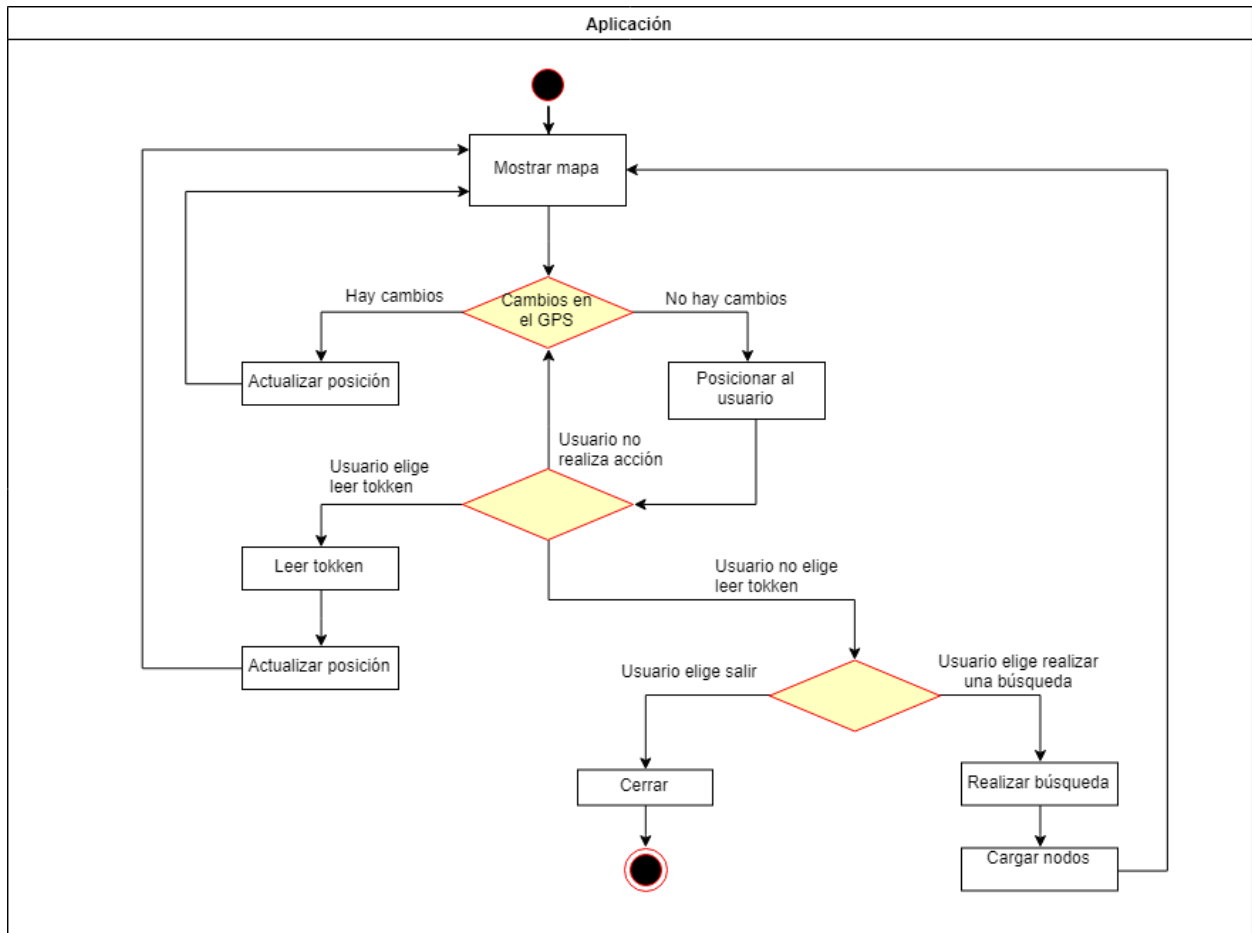


Imagen 2.1: Diagrama de actividad de la aplicación

Con la solución propuesta, estos problemas no ocurrirían. Pero es justo decir que hay otras consideraciones que se introducen a partir de esta perspectiva. Por ejemplo, un móvil que no pudiera conectarse a Internet, no podría acceder a los datos del Web Service.

Los aspectos antes expuestos fueron considerados y evaluados para dar con la propuesta de solución actual por los motivos antes mencionados.

2.2.1 Lenguaje de programación

Para la elección del lenguaje de programación del Web Service, se priorizó aquellos lenguajes que son compilados por sobre los interpretados, debido a que su implementación es más robusta; y aquellos que sean orientados a objetos debido a la naturaleza del problema, ya que del mismo se pueden identificar claramente quienes son las entidades que intervienen y cómo se relacionan entre ellas.

De la propuesta de solución, se puede identificar que la comunicación entre la aplicación y el servidor está claramente establecida ya que se basa en una serie de servicios por los que el móvil debe consumir, muy similar a un modelo MVC, donde los controladores poseen todos los *endpoints* a los cuales se puede acceder y que están relacionados con las entidades del modelo.

Tras una investigación previa, junto con un criterio personal producto de la experiencia obtenida trabajando en proyectos con otras tecnologías, sumado al aval de la dirección del proyecto, se determinó que la utilización de los frameworks Spring y Hibernate, cubrían todo el ciclo de vida del modelo MVC. De los lenguajes compilados orientados a objetos que soportan dichos frameworks, se eligió a Java por sobre .NET (este último posee NHibernate + Spring.NET).

Si bien ambos lenguajes son líderes en el mercado en el desarrollo de aplicaciones del lado servidor, se escogió el primero ya que tiene mayor portabilidad. Java puede ser utilizado desde varios entornos de desarrollo integrado, en cualquier sistema operativo y puede ser levantado por cualquier contenedor de servlets. En cambio .NET solo cuenta con Visual Studio como IDE oficial y se ejecuta en plataformas con sistema operativo Windows de forma nativa, aunque existen versiones *cross-platforms* que son recientes; y solo puede ser servido en *Internet Information Services* (IIS).

La máquina virtual de Java (JVM), por otro lado, se sitúa por encima del hardware del sistema sobre el que se va a ejecutar la aplicación, y actúa como un puente entre el *bytecode* y el sistema sobre el que se pretende ejecutar, como se ve en la Imagen 2.2.

Un factor incidente también es la experiencia del desarrollador del proyecto en estas tecnologías, ya que el aprendizaje de .NET u otro lenguaje conllevaría a una demanda extra de tiempo pudiendo ralentizar los tiempos de ejecución del proyecto que ya estaban fijados de antemano.

Por otra parte, se tuvo en cuenta que para desarrollar la aplicación móvil se podía utilizar tanto Android como Kotlin. Debido a que Android utiliza el mismo SDK que Java, existe una vinculación en cuanto a la sintaxis de estos lenguajes y la incidencia de la curva de aprendizaje de ambas variantes, teniendo en cuenta que tanto en Android como en Kotlin se podían lograr los mismos resultados.

Tampoco hay que dejar de lado la combinación PHP + Symfony para desarrollar servidores *backend* que se amolden al modelo MVC, ya que cuenta con una comunidad activa muy grande. Entre ambas variantes, Spring está mejor catalogado³ que Symfony en la escala de frameworks de desarrollo *backend*, siendo también que

³ Más información en https://www.slant.co/versus/1436/3758/~spring-mvc_vs_symfony

en Java pueden lograrse desarrollos más complejos con un nivel determinado de esfuerzos que con PHP⁴.

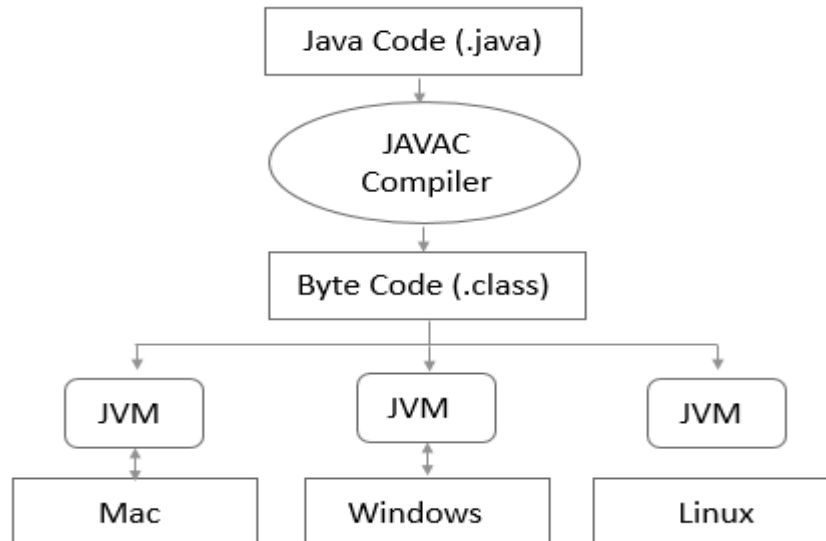


Imagen 2.2: Máquina virtual de Java

Además, y a modo de apreciación personal, la sintaxis de Java es un poco más fácil de interpretar que la de PHP y no requiere de tanta configuración previa para ejecutar un proyecto. Por último, Symfony no tiene soporte para Hibernate, utiliza Doctrine que es un ORM similar, pero el anterior es líder en el mercado.

En el capítulo 3 se hablará con más detalle del porqué de la elección de los frameworks mencionados.

2.2.2 Visualización de mapas

Visualizar la información que se recibe del servidor es otro aspecto importante.

Existen varias alternativas a la API de Google Maps, que finalmente fue la elegida. TomTom, Mapbox, Leaflet son algunas de las variantes⁵.

En función de la cantidad de llamadas a los servicios de la API de Google Maps, que comienza a ser paga a partir de los 100.000 *requests* mensuales, podemos establecer que su costo es nulo para los objetivos de este proyecto. Su acoplamiento es sencillo y directo con la tecnología Android; recordar que Google fue quien desarrolló y mantiene este sistema operativo. Por esa razón, está prácticamente

⁴ Más información en <https://www.quora.com/Which-one-should-I-choose-for-back-end-Java-or-PHP>

⁵ Más información en <http://geoawesomeness.com/google-maps-api-alternatives-best-cheap-affordable>

garantizada la alta disponibilidad de la API y su acceso, además de su constante actualización y precisión.

Si bien se detallarán posteriormente, la API cuenta con ciertos elementos que permiten traducir la información que está persistida en la base de datos en objetos que pueden ser fácilmente agregados al mapa para representar las entidades que se han identificado.

Estas razones permiten bajar el nivel de incertidumbre del proyecto ya que Google Maps es una plataforma ampliamente probada y documentada por sobre las otras alternativas. Esto, si bien no es garantía, es un punto a favor de la planificación del proyecto en cuanto a tiempo y objetivos se refiere.

2.3 Conceptos del lado servidor

Antes de introducir los conceptos mencionados anteriormente es pertinente describir el flujo básico de actividades del sistema, para poder comprender en qué parte del mismo interviene cada uno.

El sistema es un caso típico de una arquitectura cliente-servidor de 3 capas. En ésta, un cliente realiza peticiones a otro programa, el servidor, quien le da respuesta. En secciones posteriores se detallará el comportamiento de cada capa. En el lado servidor, en el caso de este proyecto, se encuentra desplegado un Web Service. A través de Internet, se puede acceder a los servicios que están alojados en dicho servidor por cualquier medio, como se ve en la Imagen 2.3. Para los fines de este proyecto solo nos limitaremos a móviles bajo plataforma Android. El cliente recibe la información y la manipula sabiendo el formato en que ha llegado la misma.

2.3.1 Arquitectura

Como se dijo anteriormente, se determinó que el intercambio de datos entre la aplicación y la base de datos se hará mediante un Web Service.

Un Web Service es un medio utilizado para realizar intercambio de datos entre aplicaciones a través de un canal y formato determinado. Dicho Web Service puede ser consumido por cualquier cliente sin importar el lenguaje en que este esté escrito o en qué plataforma esté montado, además de que brinda una capa de abstracción entre ambos extremos del sistema: la base de datos y el cliente, pudiendo ocurrir cambios en la lógica de negocios sin que ninguno de estos extremos se vea afectado.

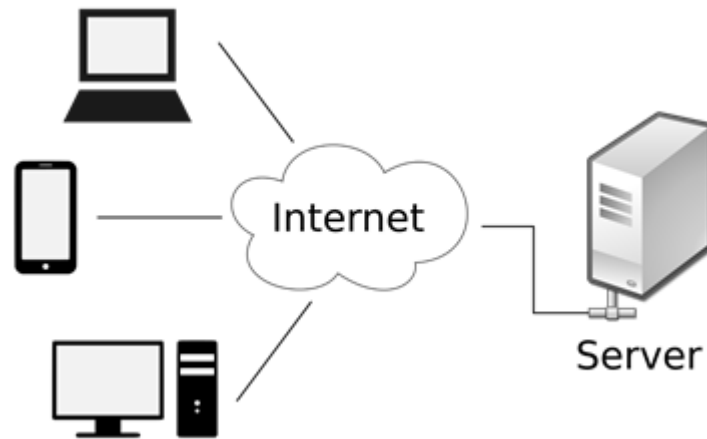


Imagen 2.3: Arquitectura Cliente - Servidor

Amparado en cuestiones de diseño, se decidió que la arquitectura del mismo sea *Representational State Transfer* (REST). Esto se debe a que REST se ajusta a las necesidades del proyecto y es de fácil y rápida implementación.

La disyuntiva planteada en este aspecto era sobre una representación REST o SOAP (*Simple Object Access Protocol*). Por un lado, la arquitectura REST es más nueva y de diseño más sencillo, aunque no contempla cuestiones de seguridad cómo si lo hace SOAP, se determinó que el intercambio de datos entre cliente-servidor no posee contenido sensible y por lo tanto no es necesario realizar esfuerzos extra en preservar la integridad de los datos.

REST está basado en el protocolo de comunicación HTTP que brinda una serie de primitivas básica de petición e intercambio de datos, de las cuales la más importante para este proyecto es la petición GET, que a partir, o no, de ciertos parámetros, realiza consultas en el lado servidor y las envía de nuevo por el mismo canal. Cada mensaje HTTP contiene toda la información necesaria para ejecutar la petición por lo tanto no se requiere guardar ningún tipo de estado de la comunicación sino que todas son independientes entre sí.

La aplicación no realiza sentencias que persistan cambios en la base de datos, sólo consume información de la misma por lo tanto se determinó que no hace falta tener una arquitectura con tantos recursos, sino que es más conveniente contar con una arquitectura simple y robusta.

Otra diferencia entre ambas posibilidades es que la arquitectura SOAP solo admite el formato XML para pasaje de mensajes, en cambio REST admite este último y JSON, por lo tanto no depende de la definición de un WSDL (*Web Services Description Language*).

REST provee una sintaxis universal de modo tal que cada recurso que posee se puede direccionar de forma unívoca a través de la URI, que es la combinación entre la

URL y la URN, acrónimo del inglés *Uniform Resource Name*, que identifica el nombre del recurso en la red en cuestión. De esta forma no pueden producirse ambigüedades ni errores en las consultas.

Además, la tendencia indica que la arquitectura REST será la más utilizada en el futuro⁶. Todas las decisiones de implementación se tomaron teniendo en cuenta el grado de impacto de la herramienta en el mercado y su uso en el ámbito global⁷.

La mayoría de las API de dominio público de los servicios que usamos habitualmente están constituidas en REST (Facebook, Yahoo, Twitter, Mercado Libre, entre otras), es un dato interesante siendo que se busca aplicar tecnologías modernas que sean fácilmente escalables y mantenibles.

2.3.2 Protocolo HTTP

Anteriormente se dijo que la arquitectura REST está basada en el protocolo HTTP; vamos a introducir un poco más esta noción.

El Protocolo de Transferencia de Hipertexto es un protocolo de comunicación que permite las transferencias de información y es el que se utiliza en la *World Wide Web* (vía Internet). Está orientado a transacciones siguiendo un esquema de pedido-respuesta. El cliente hace un pedido mediante un mensaje y el servidor le envía una respuesta.

Particularmente lo que nos interesa conocer de este protocolo son los métodos de peticiones y los códigos de estado de respuesta. HTTP define 8 métodos que indica la acción que desea que se efectúe sobre el recurso identificado. De estos métodos como ya se dijo anteriormente solo nos interesa el GET, que es utilizado para obtener información sobre algún recurso. Es posible también, y es el caso de este proyecto, enviar parámetros mediante la URL en forma de *Query String*. A partir de una URL dada, el servidor devuelve la respuesta al cliente. La misma consiste en un código de estado y el tipo de dato MIME de la información de retorno, seguido de la propia información.

En la imagen 2.4 se puede ver un ejemplo de cómo se constituye una URL para una petición HTTP. Para un entorno de pruebas local, el host será *localhost* aunque también podría ser la dirección IP privada dentro de la red donde se esté trabajando y en el caso de una puesta en producción en Internet del Web Service, será la dirección de IP pública para ese host.

⁶ Más información en <https://rigor.com/blog/2016/06/restvs-soapapis>

⁷ Más información en <https://nordicapis.com/rest-vs-soap-nordic-apis-infographic-comparison/>

El puerto en nuestro caso es 8080, utilizado por el contenedor de Servlets para acceder al Web Service debido a que supone que el puerto bien conocido por defecto para transacciones HTTP está ocupado, que es el puerto 80.

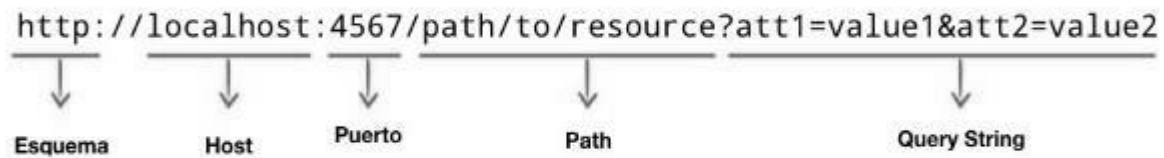


Imagen 2.4: Esquema de una URL HTTP

El path es la dirección dentro del host del recurso que se está consultado, que en nuestro caso será el nombre de los servicios que se desarrollaron en el Web Service y seguido de esto, luego del símbolo '?', el nombre y valor de los parámetros que queremos enviar en la consulta, si los hubiere.

2.3.3 Formato de respuesta

Hasta aquí hemos dicho que un cliente envía una petición al servidor y el mismo, luego de ejecutar los algoritmos pertinentes, retorna una respuesta mediante el protocolo HTTP estableciendo una conexión TCP (*Transmission Control Protocol*). Ahora hay que determinar qué formato tendrá esa respuesta, la misma debe ser homogénea de forma que el cliente sepa de antemano de qué forma debe decodificar la misma.

Para esta oportunidad se evaluaron dos posibilidades, JSON y XML (*Extensible Markup Language*). Se escogió el formato JSON ya que a diferencia de XML, posee mayor soporte y es de más fácil lectura a simple vista. No requiere de etiquetas a diferencia del otro por lo que reduce el ancho de banda de los datos a enviar, como se ilustra en el ejemplo de la Imagen 2.5, lo cual es un punto a favor y al igual que REST la tendencia entre JSON y XML se inclina a su favor. JSON puede representar cuatro tipos primitivos: cadenas, números, booleanos y valores nulos; y dos tipos estructurados: objetos y arreglos.

Una cadena es una secuencia de cero o más caracteres. Un objeto es una colección desordenada de cero o más pares {*nombre:valor*}, donde un nombre es una cadena y un valor es una cadena, número, booleano, nulo, objeto o arreglo. Un arreglo es una secuencia desordenada de cero o más valores.

XML	JSON
<pre><empinfo> <employees> <employee> <name>James Kirk</name> <age>40</age> </employee> <employee> <name>Jean-Luc Picard</name> <age>45</age> </employee> <employee> <name>Wesley Crusher</name> <age>27</age> </employee> </employees> </empinfo></pre>	<pre>{ "empinfo" : { "employees" : [{ "name" : "James Kirk", "age" : 40, }, { "name" : "Jean-Luc Picard", "age" : 45, }, { "name" : "Wesley Crusher", "age" : 27, }] } }</pre>

Imagen 2.5: Comparación entre XML y JSON

Se puede apreciar en la imagen anterior que el formato JSON es de más fácil lectura y en comparación se envían menos caracteres en el paquete HTTP, esto es importante para reducir el ancho de banda del contenido.

2.3.4 Paradigma orientado a objetos

La programación Orientada a Objetos es un paradigma de programación que se basa en tratar a los actores del problema como objetos con entidad propia, denominado Clase, donde cada uno posee atributos y funcionalidades particulares. Una instancia de esta clase es un caso particular de la misma que puede tener atributos distintos al resto y que le llamamos Objeto.

Un objeto contiene toda la información que permite definirlo e identificarlo frente a otros objetos pertenecientes a otras clases e incluso frente a objetos de una misma clase, al poder tener valores bien diferenciados en sus atributos. A su vez, los objetos disponen de mecanismos de interacción llamados métodos, que favorecen la comunicación entre ellos.

La programación orientada a objetos tiene muchas características muy importantes que permiten abstraer el dominio del problema y representarlo muy fielmente, algo que más adelante veremos cómo están aplicadas en este proyecto.

No está de más mencionar que Java soporta este paradigma, y por lo tanto Android también, así como también la vasta mayoría de lenguajes.

2.3.5 Grafos

Introducido el paradigma orientado a objetos, tenemos pie para hablar sobre la representación en forma de grafo que se mencionó anteriormente y como se lleva la misma a código. Pero primero es importante aclarar conceptos sobre los mismos.

2.3.5.1 Definiciones

Un grafo es un conjunto de objetos llamados vértices unidos por enlaces llamados aristas, que permiten representar relaciones entre elementos de un conjunto y se representa como un conjunto de puntos unidos por líneas.

En esta imagen 2.6 vemos un ejemplo de un grafo formado por 4 elementos conectados por 4 aristas. Notar que si estuviéramos parados en el punto 1 y quisiéramos llegar al punto 4, existen dos caminos posibles: $1 - 3 - 2 - 4$ ó $1 - 2 - 4$. Esta noción introduce dos ideas, la primera es el concepto de que este grafo es no dirigido: esto quiere decir que sus aristas no están orientadas por lo tanto se puede ir de un punto A hacia un punto B y deshacer el camino de la misma forma.

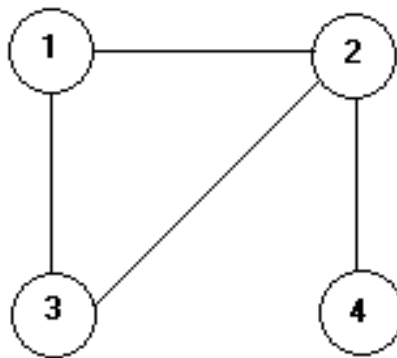


Imagen 2.6: Grafo de 4 nodos

El otro concepto que se desprende es la posibilidad de navegar por el grafo de distintas formas. Como vimos, entre 1 y 4 existen dos caminos sin repetir puntos, aunque el primero mencionado tiene longitud 3 y el segundo tiene longitud 2. En términos de eficiencia sería interesante poder llegar de un punto a otro consumiendo la menor cantidad de recursos posibles.

Ahora, llevando esta idea a nuestro proyecto, sería muy simple poder representar un grafo utilizando el paradigma orientado a objetos. Podría resolverse utilizando una sola clase, la clase Nodo. Si estuviéramos en un marco de 2 dimensiones, podríamos

pensar en un grafo cuyos puntos tengan una coordenada $\{x,y\}$. Si a este grafo lo posicionáramos sobre un mapa, entonces esas coordenadas pasarían a ser la longitud y la latitud de ese punto en el mapa.

Los vecinos de cada Nodo, o sea los puntos que están conectados entre sí, no son más que una lista de Nodo propiamente dicha. Entonces, una clase Nodo tendrá los atributos latitud, longitud y una lista de aquellos objetos de clase Nodo que están conectados con él. Si el objeto A de la clase Nodo tiene al objeto B de la clase Nodo en su lista de vecinos y si el objeto B tiene a A en su lista de vecinos, estamos ante un grafo no dirigido como el que vimos en la imagen.

2.3.5.2 Algoritmos de Búsqueda

Los algoritmos de búsqueda en grafos son algoritmos que nos permiten automatizar el recorrido en un grafo de forma tal que podamos obtener el conjunto de nodos que conectan a un punto A con un punto B , como se ejemplificó más arriba. Existen diversos tipos de algoritmos de búsqueda que son más o menos eficientes en función de la longitud del camino que retornan.

Algunos de estos algoritmos, como lo son el Algoritmo de Primero en Profundidad o Primero en Amplitud, recorren el grafo de manera ordenada expandiendo sus nodos hacia abajo o por niveles respectivamente. Estos algoritmos son unos de los más rápidos, pues retornan el primer camino que encuentran desde la raíz al nodo objetivo, pero este camino puede no ser el mejor, sino el más largo. En la imagen 2.7 se ejemplifica el recorrido en profundidad y en anchura del mismo grafo numerando el orden de recorrido de sus nodos.

Para este caso, por ejemplo, si nos posicionamos en el nodo raíz (1) y quisiéramos llegar al nodo numerado como 6, ambos algoritmos nos darían un camino de igual longitud. Pero nótese que el nodo numerado como 5 en la imagen de la izquierda, se visita como decimo en la imagen de la derecha, un caso inverso pasa con el nodo numerado como 7 en la búsqueda en anchura.

Si bien existen muchas variantes de algoritmos, el que utilizamos en este proyecto para recorrer el grafo, es el algoritmo de Búsqueda de Costo Uniforme, que nos brinda el camino de costo mínimo entre el nodo raíz y el objetivo. En términos espaciales es el camino más corto entre el usuario de la aplicación y el objetivo que está buscando, lo cual sería lo esperable como usuario del sistema.

La forma de medir la distancia entre dos puntos es calculando la longitud del arista que los conecta, que se puede determinar de forma cuadrática como está en la Ecuación 2.1. Siendo A y B los puntos, la distancia S entre ambos está dada por la Ecuación 2.1.

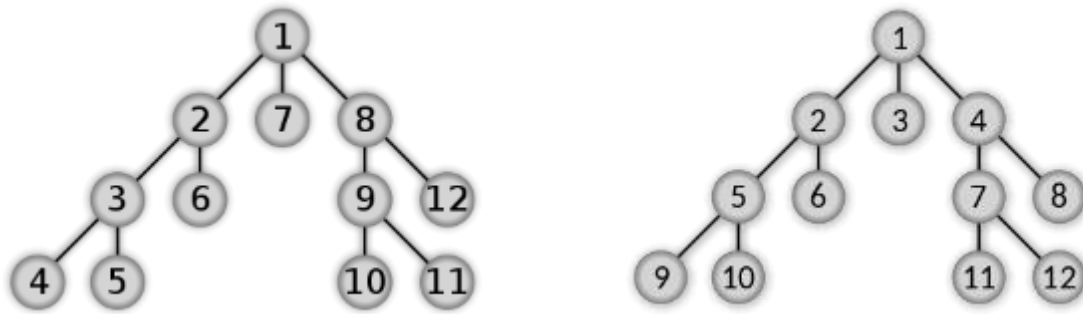


Imagen 2.7: Recorrido de grafo en profundidad y anchura

$$s = \sqrt{(A_{latitud} - B_{latitud})^2 + (A_{longitud} - B_{longitud})^2}$$

Ecuación 2.1: Calculo distancia entre puntos

El algoritmo implica la expansión de nodos mediante la adición a una cola con prioridad, de todos los nodos vecinos no expandidos que están conectados al último nodo analizado. En la cola, cada nodo se asocia con su costo total desde la raíz, donde se les da mayor prioridad a los caminos de costo mínimo. El nodo en la cabeza de la cola es expandido, adicionando sus nodos vecinos con el costo total desde la raíz hasta el nodo respectivo hasta encontrar el objetivo.

2.3.5.3 Inclusión en el proyecto

Visto esto, la razón por la que se decidió definir un grafo que represente a las dependencias de la Universidad se debe a que se necesita poder contar con una estructura de datos que relacione a los puntos de interés con el espacio físico y entre ellos, de forma tal que se pueda navegar por dicha estructura para obtener el camino deseado.

Si no se contara con la información de las conexiones entre nodos, no se podría generar dinámicamente el camino buscado. Para realizar esta tarea, se deberían tener definidas de ante mano todas las posibles combinaciones entre origen y destino de puntos y elegir la más corta; o bien, realizar un trabajo manual para definir el camino más corto entre cada par de puntos, y retornar dicho camino cuando se recibe una petición. Esto es, no podríamos aplicar el algoritmo de búsqueda elegido porque no se podría navegar la estructura.

Además, el poder contar con el grafo en función de la base de datos que se creó para este proyecto, permite definir las aristas en consecuencia del plano del edificio en cuestión, de modo tal que esa arista no pase por un lugar indebido, como podría ser, atravesar una pared.

2.3.6 Servlet

Un Servlet es una clase en lenguaje Java, que permiten extender las aplicaciones alojadas por servidores web, de tal manera que pueden ser vistos como *applets* de Java que se ejecutan en servidores. Esto quiere decir, que nos permite construir aplicaciones web que admiten peticiones a través del protocolo HTTP accediendo a ellos mediante un navegador web. La respuesta generalmente es enviada en formato HTML y representada por dicho navegador.

Es interesante hacer un comentario sobre el protocolo HTTPS. Este está basado en el protocolo HTTP pero está destinado a la transferencia segura de datos ya que utiliza un cifrado sobre el canal de comunicación de forma tal que si el mensaje es interceptado no pueda ser descifrado. Pero como este es un proyecto académico y la información que se transporta de un lado a otro no es de alta sensibilidad, se optó por implementar el protocolo sin cifrado.

De forma extensiva, un contenedor de Servlet es un programa capaz de recibir peticiones de páginas web y redireccionar estas peticiones a un objeto Servlet, esto quiere decir también que podemos tener corriendo varias aplicaciones bajo un mismo contenedor, esto es, un mismo servidor. El Web Service desarrollado es un Servlet que está corriendo en un contenedor de Servlet, dicho contenedor es Apache Tomcat⁸ que puede funcionar como servidor web por sí mismo.

Cómo vimos en la sección 2.3.2, para acceder a cualquier aplicación mediante un browser debemos acceder a ella a través de una URL. Tomcat por defecto direcciona las peticiones de Servlet que llegan al puerto 8080 (por eso se aclaró en aquella sección), aunque esto puede establecerse a cualquier puerto de red libre.

El funcionamiento es muy simple y es el siguiente:

- El navegador pide una página al servidor HTTP.
- El contenedor de Servlet delega la petición a un Servlet en particular elegido de entre los Servlet que contiene.

⁸ Más información en <http://tomcat.apache.org/>

- El Servlet se encarga de generar el texto de la página web que se entrega al contenedor.
- El contenedor devuelve la página web al Browser que la solicitó. En nuestro caso, esta página es un texto en formato JSON puro.

Tomcat está escrito en Java, por lo tanto funciona en cualquier sistema operativo que soporte la máquina virtual Java. La versión utilizada fue Tomcat 8, aunque hoy en día ya está disponible la versión Tomcat 9.

2.4 Conceptos del lado cliente

Ahora retomando la Imagen 2.3, nos posicionamos en el lado cliente. Se introducirán algunos conceptos relacionados a la programación en plataforma Android y las posibilidades que tenemos con ella.

2.4.1 Sistema operativo Android

Android es un sistema operativo diseñado fundamentalmente para ser soportado por dispositivos móviles que posean pantalla táctil, como ser smartphones y tablets y actualmente brinda soporte para relojes inteligentes, televisores y hasta automóviles con el nuevo Android Car.

A lo largo de los años han salido al mercado diversas actualizaciones de este sistema, tal es así que hoy se encuentra disponible la versión Android Oreo (8.0), lanzada en agosto de 2017. La primera de ellas fue Android Apple Pie lanzada en septiembre de 2008.

Particularmente en este proyecto se decidió desarrollar una aplicación que sea compatible con la API nivel 19 o superior; que equivale a aquellos dispositivos que posean Android 4.4 o mayor. Esto se debe a que la gran mayoría de los dispositivos en la actualidad cuentan con activaciones de esta versión o bien superiores⁹.

Hasta mediados del año 2017, todas las aplicaciones Android eran escritas en lenguaje Java, ya que este era el lenguaje oficial para esta plataforma. Actualmente, se ha añadido Kotlin como lenguaje oficial al mismo nivel que Java, y ya es soportado por las versiones más nuevas de los IDE Android.

⁹ Más información en https://www.android.com/intl/es_es/versions/kit-kat-4-4/

2.4.2 Interfaz de programación de aplicaciones

En la Introducción del capítulo anterior se dijo que la API de Google Maps es una herramienta que nos brindaba muchas posibilidades para llevar a cabo este proyecto. Pero es importante que antes de hablar de ella, especifiquemos que es una API y cómo funciona.

Una interfaz de programación de aplicaciones, o en inglés *Application Programming Interface* abreviada como API, es un conjunto de funciones y procedimientos que cumplen diversas funciones con el fin de ser utilizadas por otro software y que se representan como una capa de abstracción debido a que nos permite implementar las funciones y procedimientos que engloba en nuestro proyecto sin la necesidad de programarlas de nuevo.

Uno de los principales propósitos de una API consiste en proporcionar funciones para dibujar ventanas o iconos en la pantalla. Así los programadores se benefician de esto haciendo uso de dichas funcionalidades sin tener que hacer todo desde cero.

En el caso de este proyecto que hemos utilizado la API de Google Maps para Android¹⁰, la misma resuelve muchas cuestiones que serían engorrosas de trabajar manualmente y por tanto nos permite realizar tareas como agregar mapas a la aplicación, planos interiores, imágenes de Street View, marcadores personalizados, entre otras cosas; sin programar mucho código. Más adelante veremos esto con profundidad, pero por ejemplo, para poder visualizar un mapa en nuestra aplicación, simplemente debemos implementar la interface `OnMapReadyCallback` y sobrescribir dos métodos (`onCreate()` y `onMapReady()`). Esto es suficiente para contar con un objeto de tipo `Map` y manipularlo libremente.

Hoy en día el mundo de los sistemas y de la información está tendiendo hacia los microservicios. Una arquitectura de microservicios es un enfoque para desarrollar una aplicación software como una serie de pequeños servicios, cada uno ejecutándose de forma autónoma y comunicándose entre sí a través de peticiones HTTP a sus API. Por lo tanto es importante trabajar con tecnologías que estén en boga en el mercado sabiendo que el futuro tiende a ello.

2.4.3 Código QR

Un código QR es una imagen capaz de almacenar información en forma de matriz de puntos. Podría también decirse que es una versión bidimensional de un código de

¹⁰ Más información en <https://developers.google.com/maps/documentation/android-api/?hl=es-419>

barras. Este formato puede almacenar todo tipo de contenidos: imágenes, texto, URL, mp3 o incluso enviar un correo electrónico. Presenta tres cuadrados en las esquinas que permiten detectar la posición del código al lector, como se ve en la imagen 2.8.

Debido a que en uno de estos códigos se puede almacenar cualquier tipo de información, en este proyecto se utilizan para generar tokens de ubicación. Los mismos están pensados como códigos QR que contienen la latitud, longitud y el piso (en caso de estar dentro de un edificio) del lugar en donde se sitúen. Entonces sería posible distribuir algunos de estos tokens a lo largo de las dependencias de la UNL y un potencial usuario, que no sabe dónde está parado o no puede ser hallado por los sensores de ubicación de su móvil, podrá leer uno de estos con la cámara de su dispositivo de forma integrada con la aplicación y el sistema reaccionará en consecuencia posicionando al usuario correctamente.

Es importante mencionar que a diferencia de otros formatos de códigos de barras bidimensionales, este formato es abierto y sus derechos de patente no se ejercen por lo tanto se puede generar y manipular códigos QR sin restricción alguna.



Imagen 2.8: Ejemplo de código QR

2.4.4 Sensores de un smartphone

Los smartphones hoy en día cuentan con múltiples sensores que interactúan con el entorno y producen respuestas para el usuario, ya sea vibración, lanzamiento de notificaciones, reproducción de sonidos, actualización de posición, etc.

Los sensores más básicos y que vienen hoy en día en la mayoría de los smartphones son los siguientes:

- **Acelerómetro:** mide la aceleración, la inclinación y la gravedad. Se utiliza cuando cambiamos el dispositivo de una posición vertical a horizontal y viceversa.
- **Giroscopio:** Similar al primero, aunque más preciso y menos lineal, pues también mide la dirección y el movimiento angular, siendo capaz de calcular la rotación total.
- **Magnetómetro:** Mide la cantidad de fuerza magnética. Se encarga del funcionamiento de la brújula.
- **GPS:** rastrean el espacio vía satélite para captar en nuestra posición. Se conecta con múltiples satélites y calcula dónde nos encontramos en los ángulos de intersección.
- **Termómetro:** Puede medir tanto la temperatura interna del dispositivo, como la temperatura ambiente.
- **Lector de huellas:** Un sensor capacitivo que es capaz de detectar nuestra huella dactilar.

En este proyecto ocupamos dos de ellos. Por un lado el sensor de GPS nos indica, con una precisión variable, la latitud y longitud de nuestra posición. Esta herramienta es importante para poder ubicar al usuario en el mapa y permitirnos actualizar su posición conforme camina. El otro es el Giroscopio, ya que es interesante que si estamos viendo un mapa, el mismo rote su brújula junto con nosotros, tal cual lo hace Maps, la aplicación de Google Maps que viene instalada por defecto en cualquier móvil con Android. Ambos sensores requieren del permiso del usuario para poder ser utilizados para el caso de este proyecto.

- Capítulo 3 -

3 Diseño y desarrollo del lado servidor

En este capítulo nos centraremos en todo el esfuerzo puesto en diseñar y desarrollar el Web Service, junto con la creación y carga de la base de datos. Introduciremos también los frameworks de Java que fueron de utilidad en el desarrollo y cómo se acoplan al sistema para brindar facilidades en la implementación.

A lo largo del mismo iremos descubriendo el método propuesto de solución aunque a esta altura del informe ya se podría ir imaginando en función de las tecnologías y conceptos presentados anteriormente. Finalmente, presentaremos todos los servicios con los que cuenta nuestra aplicación.

3.1 Implementación de la base de datos

Esta subsección la dedicaremos para tratar todo lo relativo al gestor de base de datos, diseño y carga de la misma.

3.1.1 Gestor de base de datos

Como ya se dijo en la Introducción, el Sistema de gestión de bases de datos elegido fue MySQL¹¹. Es un sistema de gestión de bases de datos relacionales y está considerada como la base datos de código abierto más popular del mundo, junto con otras de licencia propietaria como Oracle y SQL Server, sobre todo para entornos de desarrollo web. La base de datos se distribuye en varias versiones pero particularmente la versión *Community*, que es la que hemos utilizado para el proyecto, está distribuida bajo la licencia pública general de GNU. MySQL es usado por muchos sitios web grandes y populares como Wikipedia, Google, Facebook, Twitter y YouTube. Según las cifras del fabricante, existirían más de seis millones de copias

¹¹ Más información en <https://www.mysql.com/>

de MySQL funcionando en la actualidad, lo que supera la base instalada de cualquier otra herramienta de bases de datos. El *DB-Engines Ranking*¹² sitúa a MySQL como el segundo gestor de base de datos más popular.

En este proyecto hemos trabajado enteramente bajo sistema operativo Ubuntu 16.04 LTS pero MySQL es soportado por casi la totalidad de las plataformas del mercado. Su instalación varía dependiendo el entorno, siendo que para Microsoft basta con descargar el instalador mientras que en entornos Linux se puede hacer desde la consola de comandos. La versión *Community* pesa alrededor de *300mb*, siendo una muy buena opción en caso de tener que instalarla de cero en un servidor por ser liviana y consumir pocos recursos.

3.1.2 Modelado del problema

Una vez instalada la base de datos, la siguiente tarea es definir las tablas y las relaciones entre ellas para representar y persistir el modelo del problema. A continuación, en la imagen 3.1, se verá el diseño de la base de datos y sobre ella desglosaremos la información que contiene.

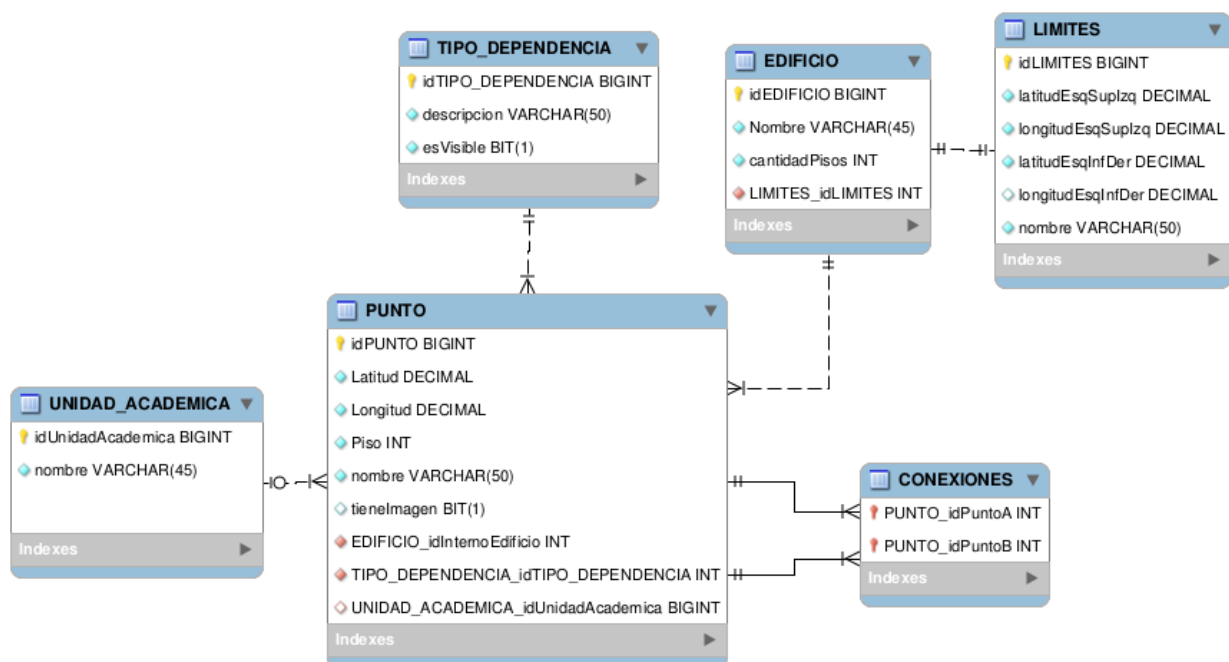


Imagen 3.1: Diseño de la base de datos

¹² Disponible en <https://db-engines.com/en/ranking>

Lo primero que hay que notar aquí es que todas las tablas poseen un id, esto es esencial en una base de datos relacional; a excepción de la tabla Conexiones que será un caso aparte ya que solo posee *Foreign Key (FK)*.

Primeramente tenemos tres tablas que son Unidad_Académica, Edificio y Tipo_Dependencia. La primera tabla hace referencia a las distintas unidades académicas presentes: FICH, FCM, FCBC, etc. La segunda tabla, refleja las plantas edilicias que posee la Universidad y que son los edificios donde están emplazadas las unidades académicas antes mencionadas, ya que por ejemplo FADU y FHUC comparten el mismo edificio. También hay que considerar que existen unidades académicas que tienen puntos de interés en otros edificios, por ejemplo FICH que posee aulas en la nave de hidráulica.

Por último, la tercera tabla sirve para representar las distintas categorías de búsqueda posible, siendo por ejemplo: Aula, Baño, Biblioteca, Oficinas, entre otros.

Notar que se definió la tabla Límites, ésta simplemente tiene los límites en donde están emplazados los edificios. Es totalmente dependiente de la entidad Edificio, para poder determinar su posición en el mapa a partir de la longitud y latitud de sus esquinas superiores e inferiores.

La entidad central de nuestro problema es la entidad Punto, que puede relacionarse con la clase Nodo que se dio como ejemplo en el Capítulo 2, subsección *Grafos*. Esta tabla contendrá los registros que asociamos con los puntos de interés que estamos buscando, por eso podemos ver que tiene una latitud, longitud y piso que nos permiten ubicarlos en el plano. También poseen un nombre; y tres FK hacia las tablas anteriormente mencionadas para saber qué tipo de punto es, a qué unidad académica pertenece y en qué edificio está ubicada. Además, se agregó un campo de tipo BIT (o *Boolean*) para establecer si el Web Service cuenta con una imagen de ese punto en particular, para enviarla a la aplicación si requiere visualizar ese lugar.

Por último, la tabla Conexiones representa la relación de Punto consigo mismo, para poder armar el grafo entre nodos. Un Punto puede tener N vecinos y ese mismo punto puede ser vecino de otros N puntos, esto representa claramente una relación *Many-To-Many*, que en un modelo relacional se traduce en una tabla intermedia. Esta tabla posee el id de ambos puntos a los extremos de los aristas del grafo, denotando una conexión entre un punto A y uno B . Notar que si se duplica en ese registro pero en orden inverso, se puede determinar una conexión entre B y A . Estamos ante la representación de un grafo no dirigido.

3.1.3 Planos de plantas

Antes de explicar cómo se realizó el trabajo de relevamiento y carga de puntos, es importante destacar en esta pequeña subsección que el mismo se hizo en función de los planos de edificios que se pudieron conseguir.

Los mismos son:

- Facultad de Ingeniería y Ciencias Hídricas y Facultad de Bioquímica y Ciencias Biológicas (Planta Baja, Primer, Segundo y Tercer Piso).
- Facultad de Arquitectura, Diseño y Urbanismo y Facultad de Humanidades y Ciencias (Planta Baja, Primer, Segundo, Tercer y Cuarto Piso).
- Aulario Común “Cubo” (Planta Baja, Primer, Segundo, Tercer, Cuarto y Quinto Piso).
- Nave Hidráulica (Planta Baja y Entrepiso).
- Facultad de Ciencias Médicas (Planta Baja).

3.1.4 Relevamiento de datos y carga

Para llenar las tablas previamente definidas, se realizó un trabajo de campo de una sola persona, en los edificios mencionados en la subsección anterior. El trabajo consistió en recorrer los edificios anotando los puntos de interés que pudieran ser parte del sistema.

Los mismos son aquellos que están dentro de estas categorías:

- Aulas
- Baños
- Bibliotecas
- Cantinas
- Escaleras
- Fotocopiadoras
- Laboratorios

- Oficinas
- Talleres
- Puntos de camino

Un *punto de camino* es aquel que no denota ningún punto de interés, pero sirve para armar el grafo en función de conectar pasillos perpendiculares (esquinas) u otras formas debido a que las aristas son líneas rectas.

De estos puntos se toma la latitud, longitud, piso, nombre, unidad académica, tipo y edificio. Estos registros se cargan en la tabla Punto. Se han relevado un total de 215 puntos en la zona comprendida por Ciudad Universitaria. De estos puntos, 152 son puntos que serán de interés para el usuario y que el mismo podrá consultar; los restantes son *puntos de camino*.

A modo de ejemplo, una sentencia SQL para cargar un registro en la tabla Punto tiene la siguiente sintaxis:

```
INSERT INTO Punto (idPunto, latitud, longitud, piso, nombre, idEdificio, idTipoDependencia, idUnidadAcademica) VALUES (2, -31.639953, -60.672090, 0, 'Aula Magna', 2, 1, 2);
```

De la misma forma, se registraron 424 entradas en la tabla Conexiones, que representan las aristas de este grafo. Pero la cantidad real de conexiones es de la mitad, ya que todas están duplicadas e invertidas para representar el camino de ida y de vuelta entre dos vértices adyacentes.

De la misma forma que el caso anterior, un ejemplo podría ser:

```
INSERT INTO Conexiones (idPuntoA, idPuntoB) VALUES (1,2);
```

Un dato interesante para marcar es el hecho de cómo representar la conexión entre pisos dentro de un edificio. Más arriba establecimos que uno de los tipos de punto que se relevaron fueron aquellos correspondientes a Escaleras. Sean los puntos A y B de tipo escalera (por ejemplo A el extremo en planta baja y B el extremo en primer piso) con solo agregar dos registros en la tabla Conexiones para estos dos puntos podremos conectar dos grafos pertenecientes a dos pisos distintos. De forma extensiva se hizo esto con todos los puntos de tipo Escalera, generando una suerte de grafo en tres dimensiones. (*{Latitud, Longitud, Piso}*).

3.2 Desarrollo del Web Service

En esta subsección nos centraremos específicamente en la construcción del Web Service. Esto es, la creación del proyecto, diagramas de clase y de la aplicación, frameworks y conceptos relacionados.

3.2.1 Proyecto Maven

El proyecto Java que se creó para desarrollar el Web Service es un Proyecto Maven. Maven¹³ es una herramienta de gestión y construcción de software que además maneja la inclusión de librerías externas. Al igual que Tomcat, es desarrollado y mantenido actualmente por la fundación Apache.

La característica más importante de Maven es su capacidad de trabajar en red. Cuando definimos las dependencias de Maven, este sistema se encargará de ubicar las librerías que deseamos utilizar en *Maven Central*¹⁴, el cual es un repositorio que contiene cientos de librerías constantemente actualizadas por sus creadores. Maven permite incluso buscar versiones más recientes o más antiguas de un código dado y agregarlas a nuestro proyecto, todo se hace de forma automática.

Maven utiliza un *Project Object Model* (POM, archivo en formato .xml) para describir el proyecto de software a construir, sus dependencias de otros módulos y componentes externos, y el orden de construcción de los elementos. Se puede observar un ejemplo de un proyecto de este tipo en la imagen 3.2. Cada vez que ocurre un cambio en el archivo *pom.xml*, Maven automáticamente detecta los cambios y descarga las dependencias nuevas o quita las que ya no están. Los JAR (*Java Archive*) son descargados en una carpeta local oculta y se incluyen al proyecto como *Maven Dependencias* y quedan listas para usar e importar.

La importancia de esta herramienta radica en la automatización de muchas tareas que se vuelven triviales y que el desarrollador no tiene por qué conocer. Aunque desde el punto de vista de este proyecto, el potencial de Maven se aprovecha al máximo a la hora de descargar e instalar al proyecto las dependencias externas, cómo ser frameworks o librerías de utilidad. Una sentencia básica para incluir una librería externa en el *pom.xml* es la siguiente:

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>${org.springframework-version}</version>
</dependency>
```

¹³ Más información en <https://maven.apache.org/>

¹⁴ Repositorio Maven Central en <https://mvnrepository.com/>

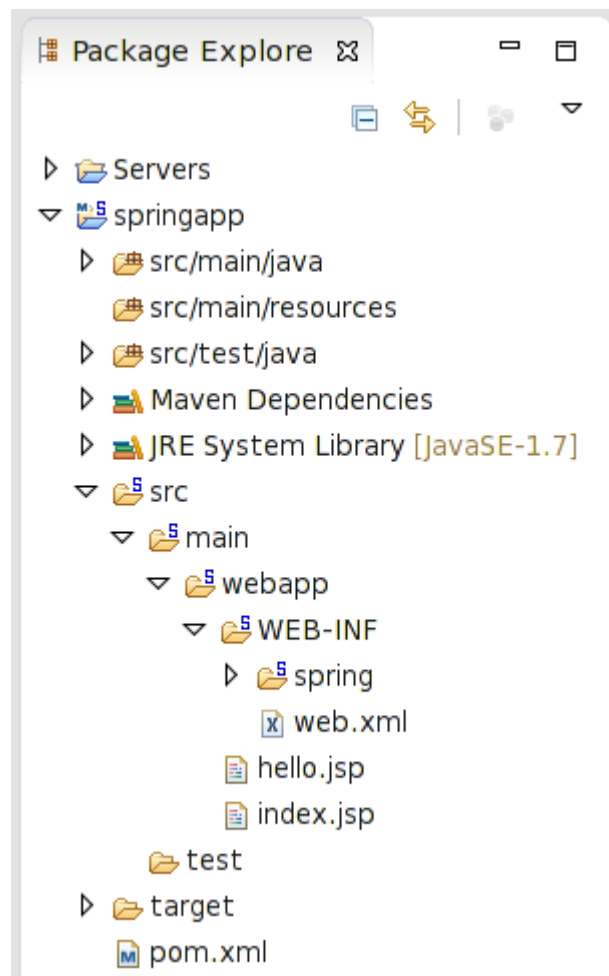


Imagen 3.2: Estructura básica de un proyecto Maven

Entre las dependencias más importantes que se incluyeron al proyecto podemos destacar:

- Spring Framework
- Hibernate Framework
- MySQL Connector¹⁵
- Apache Commons¹⁶
- log4J¹⁷

¹⁵ Más información en <https://www.mysql.com/products/connector/>

¹⁶ Más información en <https://commons.apache.org/>

¹⁷ Más información en <https://logging.apache.org/log4j/2.x/>

Sobre los dos primeros hablaremos más adelante, pero a modo de comentario: MySQL Connector es la librería que permite conectar nuestro código Java con la base de datos MySQL, de forma que podamos hacer consultas. Apache Commons es una librería de utilidades desarrollada por la fundación Apache y log4J es una librería que permite loggear todo lo que suceda en la aplicación, ya sea en un archivo externo o por consola.

3.2.2 Mapeo de objetos

Una vez que se tienen las tablas creadas en la base de datos, se debe mapear las mismas en objetos Java para poder manipularlos. Pero antes de hacer el mapeo, hay que definir las clases objetivo.

De la subsección *Modelado del problema* se presentó el modelo de la base de datos. En función del mismo, ahora presentaremos el modelo equivalente en diagrama de clases de Java, como se ve en la Imagen 3.3.

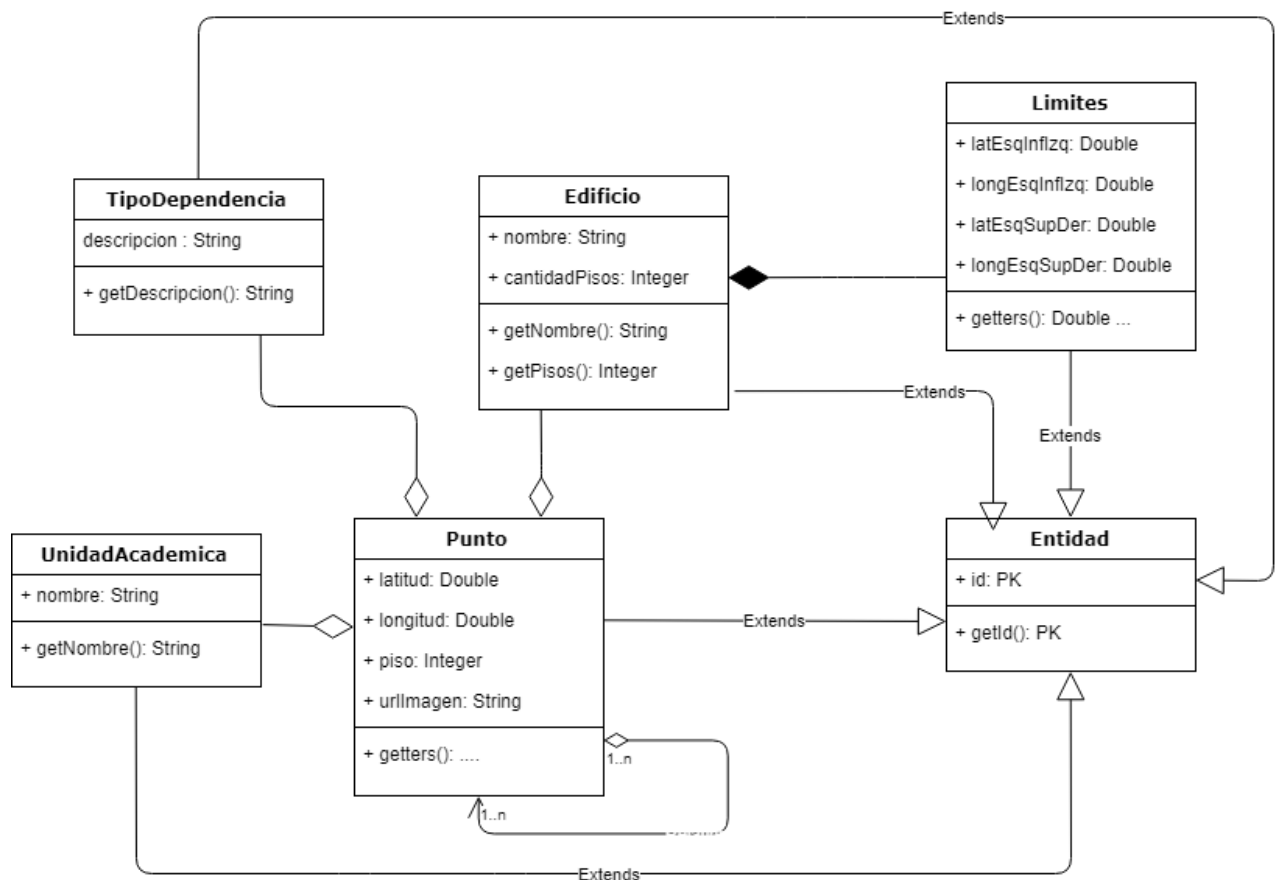


Imagen 3.3: Diagrama de clases del modelo

Notar que la correspondencia es muy similar. La única salvedad es que se definió una clase Entidad, que posee un parámetro *id* templatizable, esto se debe simplemente a una buena práctica de abstracción, ya que todas las clases son entidades, por lo tanto heredan de ella. Notar también que la tabla Conexiones, se traduce como una relación de la clase Punto consigo misma: una lista. Las FK ahora son relaciones de asociación y la dependencia de Límites con Edificio se traduce en una composición.

Ahora, el desafío es poder mapear las columnas de las tablas en la base de datos, con los atributos de los objetos de las clases definidas. Este es un problema que se resuelve con Hibernate¹⁸.

Hibernate es un framework para mapeos objetos-relaciones para Java y .NET, que simplifica la transición de llevar atributos persistidos en una base de datos, a un objeto perteneciente al modelo de objetos una aplicación. Pero también provee muchas facilidades para realizar consultas a dicha base de datos y crear objetos en función de ellos y ejecutar todo tipo de sentencias. El mismo es un ORM, siglas en inglés de *Object-Relational mapping*. Hibernate es el framework de mapeo objetos-relación más utilizado en el mercado por amplia mayoría.

Dicho mapeo se realiza mediante archivos declarativos en formato XML o anotaciones en las entidades correspondientes.

Lo primero que se debe hacer es establecer la conexión entre la aplicación y la base de datos mediante una URL. Para el caso de este proyecto y por tratarse del gestor MySQL, podemos acceder a la base de datos creadas mediante *http://localhost:3306/pfc*. Notar que *pfc* es el nombre de la base de datos. El puerto especificado aquí es el 3306 porque es el puerto por defecto que ocupa MySQL, pero podría definirse otro sin problemas. Los otros parámetros que se deben definir son un usuario y contraseña (si tuviera la base de datos) y el driver de conexión que depende del tipo de base de datos. Para MySQL el mismo es *com.mysql.jdbc.Driver*.

Por cuestiones de buenas prácticas y manejo del framework, otro de los parámetros que hay que establecer es el paquete de Java en donde están las clases a mapear, es por esto que es conveniente seguir una estructura establecida para este caso. Para este proyecto las clases que representan el modelo se encuentran bajo el paquete *com.lsikh.unlmaps.model*. Se pueden configurar muchos parámetros más pero no es el caso entrar en detalle de esto, los mismos se pueden consultar en la documentación oficial. Hasta aquí es lo básico para poder utilizar el framework.

Una vez creada la clase, nos serviremos de un juego de anotaciones de Hibernate para realizar el mapeo. Se explica a continuación con un ejemplo del proyecto, como es la clase UnidadAcademica que se mapea con la tabla Unidad_Academica.

¹⁸ Más información en <http://hibernate.org/>

Primero, se debe especificar que esta clase se relaciona con una entidad, para esto existe la anotación `@Entity` y como parámetro se le pasa el nombre de la entidad. Junto con esta, la anotación `@Table` sirve para indicar a qué tabla de la base de datos debe apuntar esa clase. Esto se define en la cabecera de la clase.

```
@Entity(name = "Unidad_Academica")
```

```
@Table(name = "Unidad_Academica")
```

```
public class UnidadAcademica extends com.lsikh.unlmaps.base.Entity<Integer>{
```

Mapeada la clase con la tabla, ahora hay que mapear los atributos de la clase con las columnas de la misma. Para esto existe la anotación `@Column` y la misma se posiciona por encima del método `get()` del atributo en cuestión dando como parámetro el nombre de la columna objetivo.

```
@Column(name = "nombre")
```

```
public String getNombre() {
```

```
    return nombre;
```

```
}
```

Un caso especial es cuando se trata con la Primary Key de la tabla, ya que se debe agregar la anotación `@Id` para el caso.

```
@Id
```

```
@Column(name = "idUnidadAcademica", nullable = false, unique = true)
```

```
public Integer getId() {
```

```
    return id;
```

```
}
```

Ahora, si se piensa en la entidad Punto, la misma posee un objeto de tipo UnidadAcademica y una relación consigo mismo de tipo *Many-To-Many*. Para resolver el primer caso, se utiliza la anotación `@JoinColumn` junto con el tipo de relación, que en este caso es `@ManyToOne`, debido a que muchos objetos de la entidad Punto, puede estar relacionados con un objeto UnidadAcademica en particular.

```
@ManyToOne(fetch = FetchType.LAZY)
```

```
@JoinColumn(name = "idUnidadAcademica")
```

```
public UnidadAcademica getUnidadAcademica() {
```

```
    return unidadAcademica;
```

```
}
```

Para resolver el segundo caso, utilizamos la anotación `@JoinTable`, que permite unir dos columnas de una tabla con las *FK* correspondientes.

```
@ManyToMany(fetch = FetchType.LAZY)

@JoinTable(name = "Conexiones", joinColumns = {

    @JoinColumn(name = "idPuntoA") }, inverseJoinColumns = {

        @JoinColumn(name = "idPuntoB") })

public Set<Punto> getVecinos() {

    return vecinos;

}
```

Este juego de anotaciones se utiliza para todas las clases del modelo y así quedan mapeadas todas las relaciones.

3.2.3 Modelo por capas

El modelo de desarrollo seguido que marca la arquitectura del sistema *backend* es el de programación por capas. Este modelo define tres capas bien separadas que son la de acceso a datos, la capa de lógica de negocios o servicios y la capa de presentación y es comúnmente aplicado a los sistemas que siguen una arquitectura cliente-servidor.

La principal ventaja de este modelo es que permite trabajar de forma desacoplada en cada capa sin afectar los otros niveles en caso de efectuar cambios internos, pues lo único que se debe mantener es la concordancia de paso de mensajes entre capas.

La capa de modelo es aquella que manipula la persistencia y acceso a los datos de la base de datos, como se explicó en la subsección anterior.

La capa de servicios es la capa intermedia entre el usuario y la base de datos y es la encargada de llevar a cabo las reglas de negocio que sean pertinentes ya que es un nexo entre la petición que efectúa el usuario y la información persistida.

La capa de presentación, como su nombre lo indica, es la encargada de tomar la información que recibe de la capa de servicios y presentarla al usuario. Es común que en esta capa intervengan otras tecnologías apuntadas a hacer más “amigable” la interfaz al usuario, pero en el caso de este proyecto la capa de presentación solo retorna la respuesta en formato JSON de la capa de servicios, no se aplican estilos ni plantillas especiales. En este proyecto le llamaremos Controladores.

Para realizar esta división entre capas se utilizó Spring¹⁹. Spring es un framework para el desarrollo de aplicaciones y contenedor de inversión de control, que es un patrón de diseño en el que el flujo principal de la aplicación está determinado por las respuestas que se deben dar ante un determinado suceso, no se basa en el esquema tradicional de ejecución concurrente de funciones definidas.

La Inversión de Control se basa en la introspección o Reflection, como se conoce en Java. La introspección son técnicas para analizar clases en tiempo de ejecución, y así conocer las propiedades, métodos, clases que extiende e implementa, ejecutar operaciones, setear u obtener valores, entre otras. En la Imagen 3.3 se puede observar que, a diferencia del esquema tradicional, el patrón de Inversión de Control propone una clase intermedia entre el controlador frontal y aquellas que representan la lógica del problema y esta es quien provee cuál de ellas utilizar. Esto favorece la modularización del código, al evitar la dependencia directa entre unos componentes y otros pero también favorece la reutilización del mismo, ya que permite la creación de módulos independientes del resto de la aplicación.

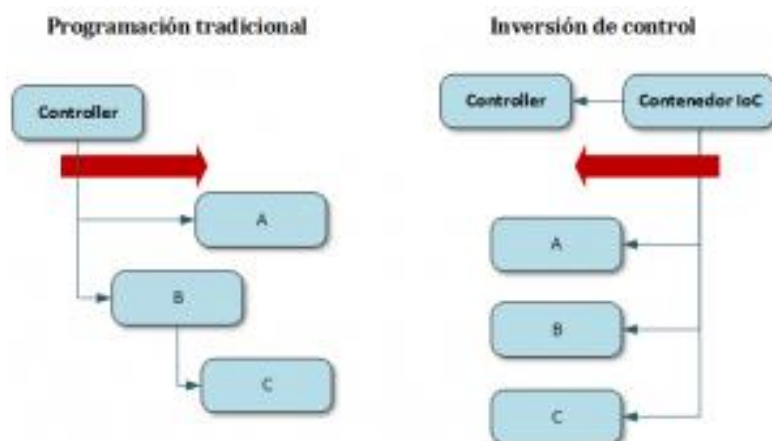


Imagen 3.4: Inversión de Control

Además, aplica otro patrón de diseño muy importante que es la inyección de dependencias, lo que implica que las clases de la aplicación se crean en tiempo de ejecución y son suministradas por una clase contenedora, no se crean objetos y reduce el acoplamiento. El patrón funciona con un objeto que se encarga de construir las dependencias que una clase necesita y se las suministra. Esto implica que la clase ya no crea directamente los objetos que necesita, sino que los recibe de otra clase.

La inyección de dependencias en este proyecto está basada en Interfaces de Java. Cada clase de cada capa del modelo de datos implementa una interfaz y poseen como objetos otras interfaces de la lógica del problema. El framework es el encargado de instanciar las clases correspondientes que implementan esas interfaces utilizando

¹⁹ Más información en <https://spring.io>

Reflection. Así, se pueden modificar comportamientos de objetos reemplazando implementaciones sin afectar a las otras clases.

Dentro de la Ingeniería del Software, la Inyección de Dependencia es uno de los principios de SOLID.

Spring posee un módulo llamado *WebMVC* que está basado en el patrón Modelo-Vista-Controlador que brinda una serie de anotaciones que permiten definir los elementos de nuestra capa de controladores, servicios y *DAO* (del inglés *Data Access Object*). En la imagen 3.5 se presenta el diagrama de clases del Web Service y sobre el mismo se explicará cómo se define cada componente en base el flujo principal de actividades.

El flujo de secuencia se explica a continuación en la imagen 3.6. El usuario quiere consultar todos los tipos de dependencias que se pueden buscar, para esto abre la pantalla correspondiente en la aplicación. La misma, envía una petición GET HTTP asíncrona al Web Service y cuando llega al mismo, el controlador frontal (Dispatcher Servlet) se encarga de enrutarla. Dicha petición pasa por el controlador correspondiente, que es parte de la capa de Controladores mencionada, y el mensaje viaja a través de las capas hasta llegar a la base de datos donde se ejecuta la consulta. El controlador delega la responsabilidad a la capa de servicio quien se encarga de aplicar lógicas de negocio a las entidades. Ésta, a su vez, delega a la capa de Acceso a Datos para que recupere la entidad (o entidades) de la base de datos. El mensaje vuelve por el mismo camino hasta llegar al Dispatcher y este retorna la respuesta en formato JSON.

Los controladores están definidos mediante las anotaciones *@Controller* y *@RequestMapping* en donde definimos el path del recurso, para este caso por ejemplo la URL se constituye como *http://localhost:8080/unlmaps/unidadAcademica*. Notar que *unlmaps* es el nombre que le hemos puesto a esto proyecto.

@Controller

@RequestMapping (value = *"/unidadAcademica"*)

public class *UnidadAcademicaController* {}

Dentro de esta clase podemos definir métodos que funcionen bajo este contexto. Dichas funciones se mapean con la anotación *@RequestMapping* para definir el recurso dentro de este servicio.

@RequestMapping(value = *"/getAll"*, method = *RequestMethod.GET*, produces = *"text/plain; charset=UTF-8"*)

@ResponseBody

public String *getUnidadesAcademicas()* {}

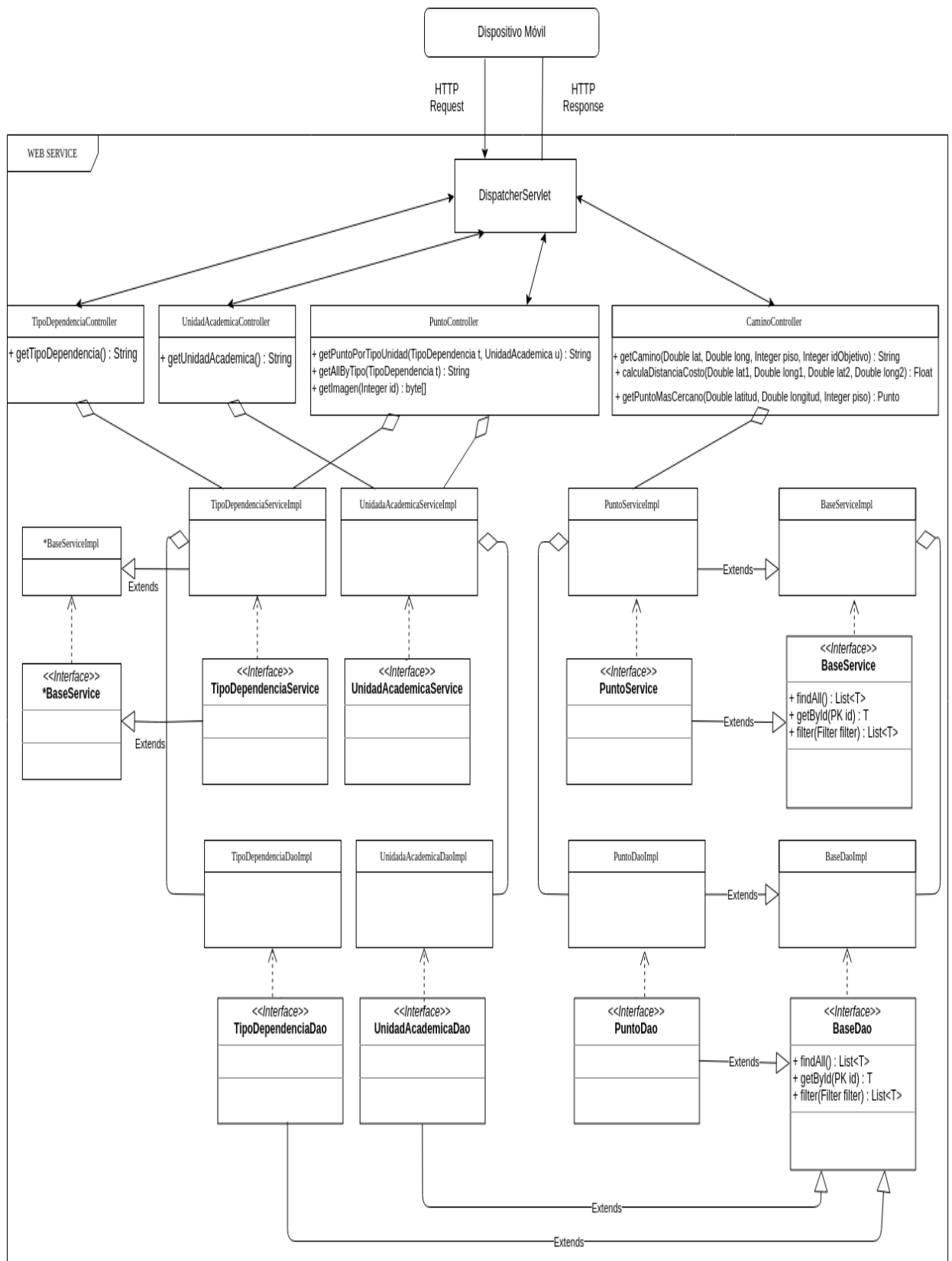


Imagen 3.5: Diagrama de clases del Web Service

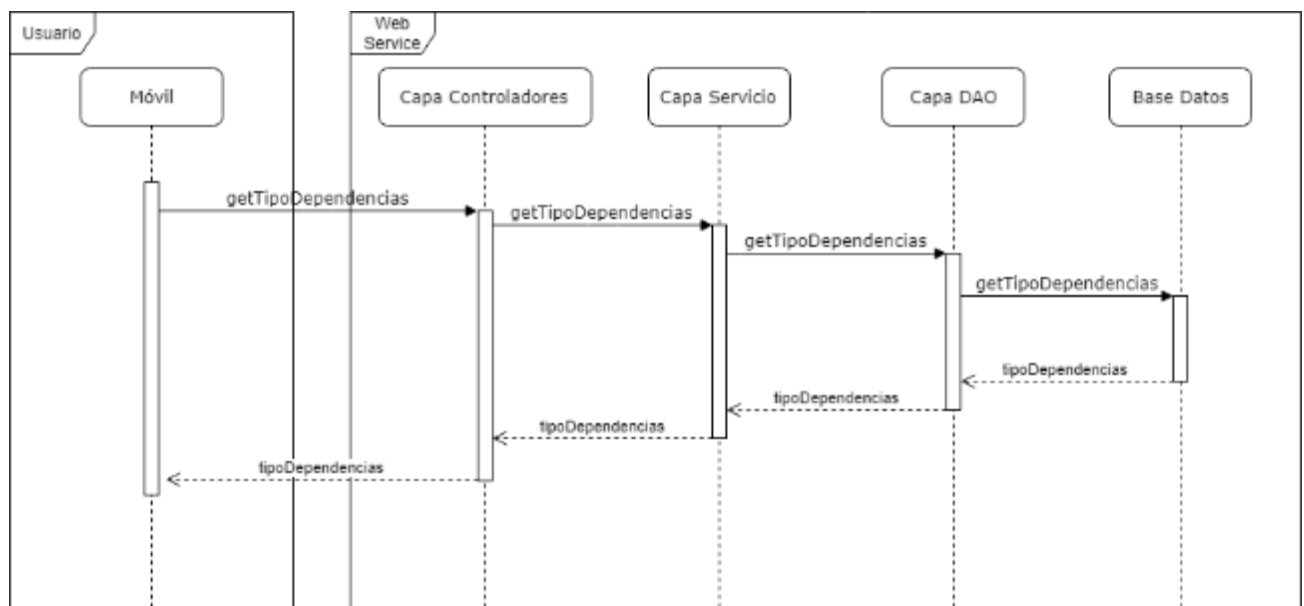


Imagen 3.6: Diagrama de secuencia de una petición

Aquí se ve que */getAll* es el nombre de la *request*, por lo tanto para llegar hasta ella, se debe completar la URL antes mencionada como *http:localhost:8080/unlmaps/unidadAcademica/getAll*. Como definimos en la Introducción, todas las peticiones HTTP son de tipo GET, esto se puede ver en el parámetro *method*.

Este método sirve para obtener todos los registros de la tabla *Unidad_Academica*, para esto el controlador le pasa el mensaje a la capa de servicio.

Para cada entidad, existe un servicio asociado que se encarga de llevar a cabo las consultas, este es el caso de la clase definida como *UnidadAcademicaServiceImpl*. Esta clase implementa la interfaz *UnidadAcademicaService* y para establecer que la misma es un elemento de la capa de servicios, se anota con *@Service*.

@Service

@Scope(ConfigurableBeanFactory.SCOPE_SINGLETON)

```
public class UnidadAcademicaServiceImpl extends BaseServiceImpl<UnidadAcademica, Integer> implements UnidadAcademicaService{}
```

De este ejemplo debemos destacar varias cosas. Por un lado, el *@Scope* está definido como *Singleton*. Esto quiere decir que el framework inyectará siempre la misma dependencia cada vez que se haga uso de la misma. Esto es muy importante para evitar tener múltiples objetos que accedan a la base de datos, ya que implican una conexión más.

Por otro lado, como se puede ver en el código y en la imagen, todas las clases de servicio implementan una interfaz. Esto es una cuestión propia del framework, que inyecta las dependencias en función de las clases que implementen la interfaz correspondiente.

Por último, notar también que todas las clases de servicio y de acceso a datos extienden de una clase base de servicio y acceso a datos. Esto es meramente una cuestión de practicidad y escalabilidad en función de aplicar buenas prácticas de programación, pues todos los servicios realizan las mismas funciones: obtener objetos de la base de datos, filtrarlos, eliminarlos, etc. Por esto se decidió abstraer todas estas funciones en una clase común y que todos extiendan de ella.

Retomando con el flujo principal, la capa de servicios tiene que buscar todos los registros de la tabla en cuestión, para esto le pasa el mensaje al objeto de la capa de acceso a datos que posee, *UnidadAcademicaDao*.

Los Objetos de Acceso a Datos devienen de un patrón de diseño para crear una capa de persistencia y por lo tanto son considerados una buena práctica de programación. La ventaja de usar objetos de acceso a datos es que cualquier objeto de negocio no requiere conocimiento directo del destino final de la información que manipula. Este patrón también define los llamados DTO (*Data Transfer Object*) que son objetos utilizados para mapear las clases persistentes para manipularlas en la capa de negocio, pero en este proyecto se decidió trabajar directamente con la clase en sí.

Los DAO, al igual que los servicios, tiene su propia anotación que es *@Repository* y se definen como *Singleton* por la razón explicada más arriba.

@Repository

```
public class UnidadAcademicaDaoImpl extends BaseDaoImpl<UnidadAcademica, Integer>
implements UnidadAcademicaDao{}
```

El objeto DAO posee un objeto de tipo *Session Factory*. El mismo está definido por Hibernate y es el encargado de ir a la base de datos y ejecutar consultas. Dichas consultas pueden ser ejecutadas mediante HQL, que es el lenguaje SQL propio de Hibernate muy similar al estándar, o mediante *Criteria* que es la API de consultas de *Hibernate*. Este último es nuestro caso. La misma es una interfaz que mediante objetos permite definir restricciones, cláusulas, alias y todo tipo de elementos de SQL de una forma más visible y manipulable para el desarrollador en comparación con HQL. Dicho *Session Factory* es inyectado por *Spring*.

El DAO realiza la consulta y le entrega el mensaje al servicio nuevamente, este aplica las reglas de negocio que hagan falta y le retorna el mensaje al controlador. El controlador parsea la respuesta en formato JSON y entrega el mensaje al controlador frontal para que este le envíe la respuesta al móvil.

Si bien este ejemplo no es el caso, contemplemos la posibilidad de querer enviar parámetros al Web Service. Se utiliza la anotación `@RequestParam` seguido del nombre del parámetro. Por ejemplo, si quisiéramos retornar una Unidad Académica en particular, la función de la llamada se definiría de la siguiente manera desde el controlador.

```
@RequestMapping(value = "/getById", method = RequestMethod.GET, produces =  
"text/plain;charset=UTF-8")
```

```
@ResponseBody
```

```
public String getUnidadById(@RequestParam("id") String id) {}
```

Para este caso la URL de acceso se constituiría de esta forma: `http://localhost:8080/unImaps/unidadAcademica/getById?id=p` donde `p` es el id de dicho registro en la base de datos.

3.2.4 Instalación de la aplicación en el contenedor

Como adelantamos en el capítulo 2, la aplicación desarrollada trabaja dentro de un contenedor de *Servlet*, Apache Tomcat. Existen diversas opciones a la hora de escoger un contenedor, además de Tomcat también se puede optar por *GlassFish*, que incluso es de propiedad de Oracle²⁰. Lo cierto es qué, y como se dijo anteriormente, todas las decisiones de tecnología en este proyecto se tomaron en base a la utilidad de las mismas en función de los objetivos a cumplir y en la comunidad que posean detrás.

La imagen 3.7 presenta una comparación a nivel mundial en el año 2015²¹ acerca de las tendencias de servidores en el mercado. Tomcat domina casi el 60% del mismo.

Application server market share 2015

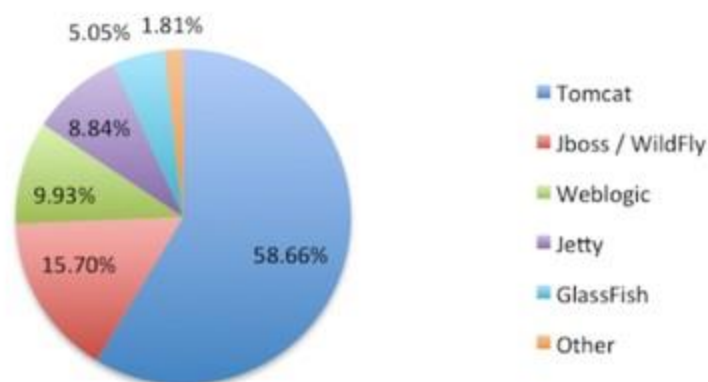


Imagen 3.7: Tendencia de uso de servidores en 2015

²⁰ Más información en <http://www.oracle.com/technetwork/middleware/glassfish/overview/index.html>

²¹ Más información en <https://plumbr.io/blog/java/most-popular-java-ee-servers-2016-edition>

La instalación se puede hacer directamente desde *Eclipse* para entornos de prueba, ya que el mismo soporta la inclusión de múltiples contenedores de *Servlet*, aunque para un entorno productivo se puede realizar la instalación a partir del *Tomcat Monitor*, una herramienta provista por Tomcat para esto, sin necesidad de utilizar un *IDE*.

3.2.5 Servicios

Para finalizar este Capítulo se especificarán todos los servicios desarrollados, su funcionalidad, punto de acceso y el tipo de respuesta.

- localhost:8080/unidadAcademica/getAll = retorna una lista de todas las unidades académicas que están en la tabla Unidad_Academica en formato JSON con la estructura

```
    {"response":  
      [{"id","nombre"}]}
```

- localhost:8080/tipoDependencia/getAll = retorna una lista de todos los tipos de dependencias que están en la tabla Tipo_Dependencia en formato JSON con la estructura

```
    {"response":  
      [{"id","descripcion"}]}
```

- localhost:8080/dependencias/getPuntos?unidad=p1&tipo=p2 retorna una lista de todas los puntos que están en la tabla Punto que cumplen con que unidadAcademica=p1 y tipoDependencia=p2 en formato JSON con la estructura

```
    {"response":  
      [{"id","nombre"}]}
```

- localhost:8080/dependencias/getDependenciasPorTipo?tipo=t retorna todas las dependencias que sean de tipo t, por ejemplo: todos los baños o cantinas; en formato JSON con la estructura

```
    {"response":
```

{{"id","descripcion"}}}

- localhost:8080/camino/getCamino?latActual=lat&lonActual=lon&piso=p&id=id retorna una lista de los puntos en donde la misma comienza en el punto del grafo más cercano al usuario que está ubicado geográficamente en (lat, lon) y en el piso p de algún edificio. El último punto de esta lista es aquel que tiene como id la que se envía como parámetro que es aquella dependencia que selecciono el usuario, en formato JSON. El resto de los puntos son los determinados por el algoritmo de Costo Uniforme que en función de encontrar el camino de menor recorrido hacia el objetivo. La estructura del JSON es la siguiente

{"response":

[{"id","latitud","longitud","piso","nombre","idEdificio","idUnidadAcademica","idTipoDependencia","tieneImagen"}]}

- localhost:8080/dependencias/getImagen?id=id retorna una imagen JPEG del punto por donde se traza un camino con id = id en formato byte[] (byte array).

- Capítulo 4 -

4 Diseño y desarrollo de la aplicación

En este capítulo abordaremos todo lo relativo al diseño y desarrollo de la aplicación en cuestión. Primero, mencionaremos algunas nociones básicas relacionadas a Android en cuanto a características y estructura del proyecto y luego profundizaremos en el desarrollo que hemos hecho. Para esto presentaremos formalmente la API de Google Maps para conocer las posibilidades que nos brinda y las herramientas que hemos usado para interactuar y consumir del Web Service desarrollado.

4.1 Nociones del proyecto Android

En esta subsección vamos a desarrollar o profundizar algunos conceptos importantes en lo que a un proyecto Android respecta. Esto implica conocer la estructura de carpetas, paquetes y funcionamiento.

4.1.1 Estructura del proyecto

En la imagen 4.1 se puede ver la estructura típica de un proyecto Android. El primer elemento que aparece allí es el *AndroidManifest.xml*. El *manifest* es un archivo de configuración que se sitúa en la raíz de todos los proyectos donde se establecen las configuraciones básicas de la aplicación, como ser el nombre de la estructura de paquetes, los permisos que hay que solicitarle al usuario, el nombre de la aplicación, el icono, entre otras cosas.

Por ejemplo, esta sentencia es la requerida para que la aplicación tenga permisos para manipular la cámara y se coloca dentro del *AndroidManifest.xml*

```
<uses-permission android:name="android.permission.CAMERA" />
```

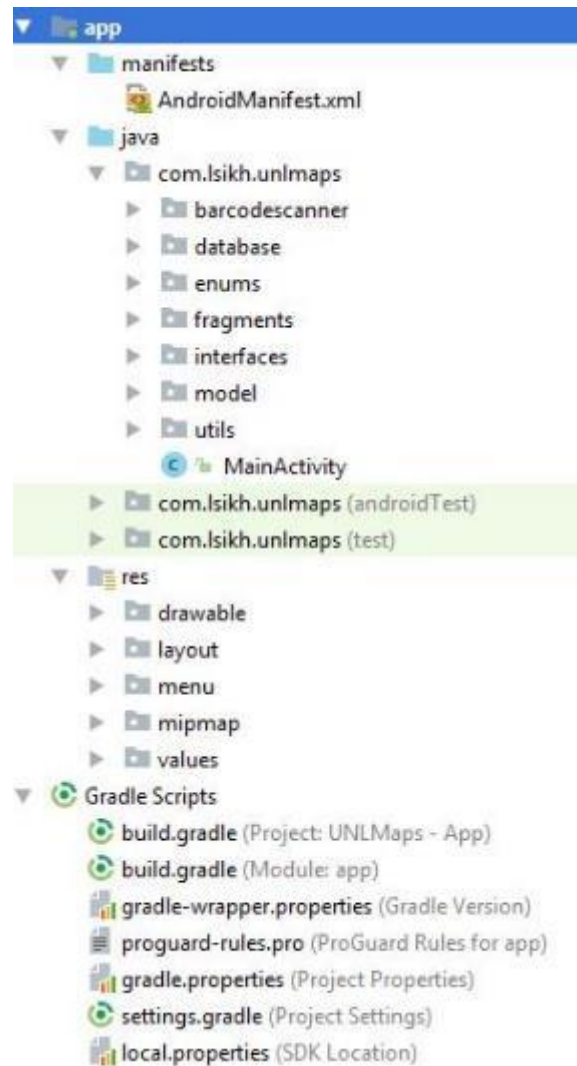


Imagen 4.1: Estructura de proyecto

Seguido de esto se encuentra la estructura de paquetes de java donde están las clases programadas. Dentro de este directorio se puede definir la estructura de paquetes que se quiera junto con las clases, interfaces, enumerables y cualquier elemento que podemos identificar en Java.

Además de toda esta estructura, para todo proyecto se define un *MainActivity*. Un *Activity* es una clase que establece una interfaz en dónde se pueden definir elementos para que el usuario interactúe. El *MainActivity* es la *Activity* principal que se ejecuta cuando se inicia la aplicación.

Luego vienen los *resources*. Aquí podemos definir todos aquellos recursos que podamos necesitar en nuestro proyecto, como ser imágenes, layout, archivos XML y valores predefinidos para utilizar en toda la aplicación. Por ejemplo, en este proyecto todos los planos de los edificios están contenidos en un directorio de la carpeta *drawable*. En el directorio *layout* se encuentran todos los *layout* que definen las pantallas de la aplicación.

También es importante el directorio *values*, donde podemos definir todo tipo de valores, como ser cadenas de texto o colores, que luego podemos llamar desde cualquier punto de la aplicación de forma tal que siempre estemos usando el mismo recurso y, si quisiéramos realizar un cambio, solo deberíamos hacer ese cambio en el recurso definido en la carpeta *values*, y ese cambio se aplicará para toda la aplicación. Por ejemplo, dentro de *res/values/colors.xml* podemos definir

```
<color name="colorPrimary">#3F51B5</color>
```

Entonces cada vez que quisiéramos utilizar ese color, podemos llamarlo por su nombre y aplicarlo. Del mismo modo podemos hacerlo con cadenas de texto en *res/values/strings.xml*

```
<string name="nav_header_main_bottom_text">Universidad Nacional del  
Litoral</string>
```

Por último están los parámetros de configuración de Gradle. Como dijimos en un capítulo anterior, Gradle es la herramienta que se encarga de construir el proyecto. Aquí podemos definir el SDK mínimo (versión de Android) que se debe tener para ejecutar nuestra APK, las dependencias que debe descargar, el número de versión de la aplicación, etc.

```
compileSdkVersion 26  
buildToolsVersion '26.0.2'  
  
defaultConfig {  
    applicationId "com.lsikh.unlmaps"  
    minSdkVersion 19  
    targetSdkVersion 26  
    versionCode 1  
    versionName "1.0"  
    multiDexEnabled = true  
    testInstrumentationRunner  
    "android.support.test.runner.AndroidJUnitRunner"  
}
```

Aquí por ejemplo, estamos definiendo que el SDK a compilar será la versión 26, correspondiente con Android 7.0 *Nougat* del smartphone LG K10 utilizado para desarrollar. La versión mínima de SDK será 19, que como dijimos anteriormente corresponde con Android 4.4 *KitKat* y, por ser un proyecto académico, la versión es la 1. Pero si este proyecto estuviera disponible en Google Play Store, por cada actualización que se haga de la aplicación se debe aumentar la versión.

4.1.2 Ciclo de vida de una aplicación

Una aplicación Android corre dentro de su propio proceso, que se crea con la aplicación y continua vivo hasta que ya no sea requerido y el sistema reclame su

memoria. Las actividades se gestionan mediante métodos implementados en los *Activity*²². Cuando se inicia un *Activity* se coloca en el tope de una pila que contiene todas las actividades de la aplicación. En la siguiente imagen se esquematizan los posibles estados y transiciones de una aplicación.

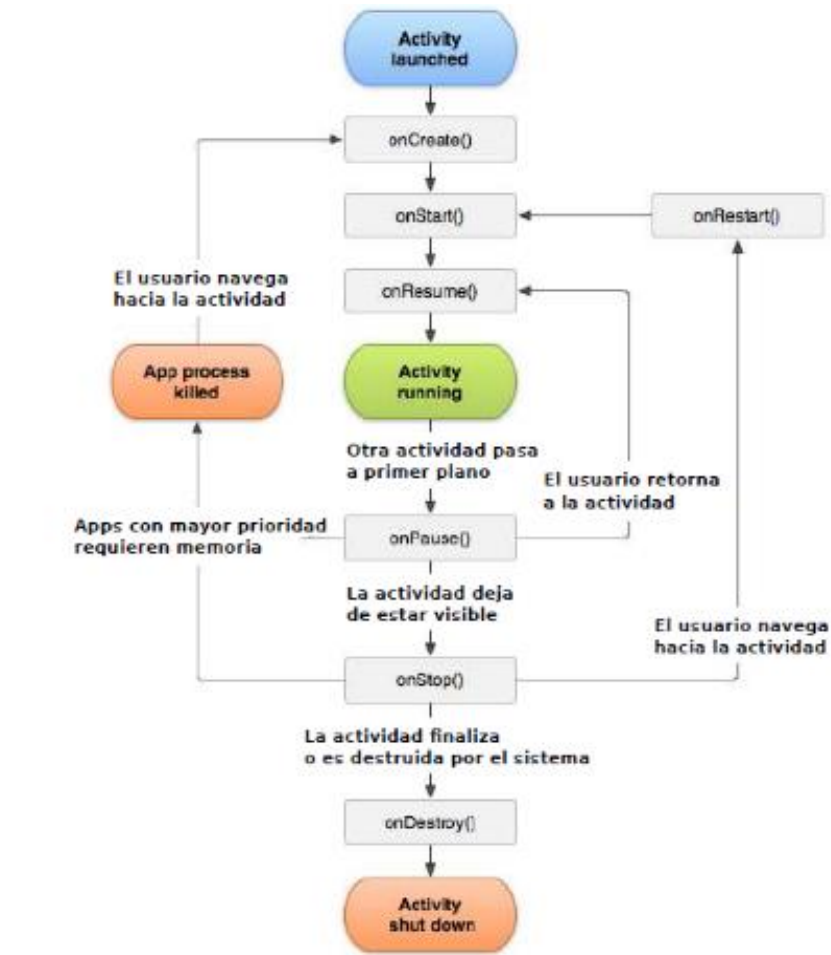


Imagen 4.2: Ciclo de vida de una aplicación

4.1.3 Elementos de interacción con el usuario

Android organiza los componentes visuales en una pantalla mediante *layout*. Existen distintos tipos de *layout* y los mismos son definidos en un archivo XML ubicándose en el directorio *res/layout*.

En dichos contenedores se puede definir la estructura básica de una pantalla y agregar componentes de interacción como ser botones, menú desplegable, campos de texto, entre otros. Igualmente, solo es necesario definir el contenedor *layout*, puesto que dichos componentes también pueden ser agregados dinámicamente (programáticamente en Java).

²² Más información en <https://developer.android.com/guide/components/activities?hl=es-419>

Cada *Activity* se relaciona con un layout y mediante el método *onCreate()* se levanta dicho archivo y se genera la pantalla, pudiendo acceder a los componentes de la misma para setear funcionalidades extra. Por ejemplo, así como dijimos que existe un *MainActivity*, existe un *activity_main.xml* que es el layout asociado a esa *Activity*. Así, si quisiéramos definir un botón en una pantalla, el mismo se puede hacer desde el xml correspondiente de esta forma

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="15dp"
    android:textColor="@color/fragment_search_search_button_color"
    android:text="@string/fragment_search_search_button_label"
    android:id="@+id/searchButton" />
```

Una vez creada nuestra *Activity*, podemos referenciar este botón a partir de su id y manipularlo. Lo mismo sucede con todos los componentes.

4.1.4 Comunicación asíncrona

Para comunicar la aplicación con el Web Service y así poder consumir sus recursos se utilizó la librería *HttpComponents* de la Fundación Apache²³.

La misma es un *toolset* de componentes en bajo nivel para Java enfocada en el protocolo HTTP para implementar servicios cliente – servidor (solo el primero es nuestro caso) con un mínimo nivel de abstracción ya que muchas cuestiones básicas están resueltas. Al igual que todas las dependencias externas que se han utilizado para este proyecto, esta librería está disponible en el repositorio central de Maven y se puede agregar al proyecto con la herramienta Gradle.

A partir de una URL válida, la librería establece una conexión asíncrona con el servidor, esto quiere decir que las solicitudes HTTP suceden fuera del hilo de la interfaz de usuario y su respuesta se procesa una vez que llega al dispositivo en segundo plano. Esto es importante ya que no bloquea la pantalla ni la interacción con la misma.

4.2 API de Google Maps

La API de Google Maps para desarrollo bajo tecnología Android permite al desarrollador colocar un mapa de Google Maps en la pantalla de la aplicación e interactuar con él tal como si se estuviera utilizando en una computadora. Para poder

²³ Más información en <https://hc.apache.org/>

acceder a esta herramienta se debe contar con una *key* provista por Google. La misma es una cadena de caracteres alfanuméricos, mayúscula y minúscula, que se proveen a un determinado desarrollador para incluirla en sus productos y pueden obtener de la web de la API de Google Maps para desarrolladores²⁴.

Las API de Google Maps están disponibles para diversas plataformas de forma tal que se pueden incluir sus prestaciones de forma Web, Android e iOS.

Para poder utilizar la API en Android debemos definir en nuestro *AndroidManifest.xml* la *API Key* obtenida y la versión del *Google Play Services* que estamos utilizando. Además, debemos definir un permiso para recibir mapas a través de Internet.

La API provee de muchas facilidades que son útiles para el proyecto. Permite interactuar con un mapa, dejando que el usuario se desplace hacia cualquier dirección e incluso hacer *zoom in* y *zoom out*. Luego permite posicionar diversos elementos dentro del mapa que enriquecen la interacción. A continuación se detallan los utilizados en el proyecto.

Para poder agregar un mapa a la pantalla, se debe implementar la interfaz *OnMapReadyCallback*. De la misma se deben implementar dos métodos y posicionar el mapa en el *layout* de la *Activity* en cuestión. El método *onMapReady(GoogleMap map)* es aquel que crea el objeto mapa.

La Google Maps Android API ofrece cuatro clases de mapas y una opción para que no se muestre ninguno:

- Normal: Mapa de carreteras típico.
- Híbrido: Datos de fotos satelitales con mapas de carreteras agregados.
- Satélite: Datos de fotos satelitales.
- Tierra: Datos topográficos. En el mapa se incluyen colores, líneas de contornos y etiquetas, y sombreados de perspectiva.

4.2.1 Markers

Los marcadores indican ubicaciones únicas en el mapa y se posicionan a partir de la latitud y longitud. Los mismos pueden contener un título e imágenes, también se puede cambiar el icono que lo representa.

²⁴ Disponible en <https://developers.google.com/maps/documentation/android-api/>



Imagen 4.3: Marker

Para agregar un marcador a un mapa, la sintaxis es muy sencilla.

```
map.addMarker(new MarkerOptions()  
.position(new LatLng(-31.952854, 115.857342)).title("Titulo");
```

4.2.2 Overlays

Un *ground overlay* o superposición de suelo es una imagen que se fija a un mapa. Las mismas se orientan respecto de la superficie terrestre en lugar de la pantalla. Por lo tanto, las acciones de rotación, inclinación o zoom al mapa harán que cambie la orientación de la imagen. Para agregar dicho elemento se puede especificar un punto del mapa junto el ancho y alto correspondiente de la imagen o también se pueden definir los puntos de la esquina inferior izquierda y superior derecha. En nuestro caso serán los planos de los edificios por donde pase el camino que el usuario deba recorrer.

4.2.3 Polilínea

Las mismas son líneas que van de marcador en marcador por encima del mapa, como se suele ver en Google Maps cuando se pide información de cómo llegar a tal punto desde otro. Se definen a partir de la sucesión de puntos por donde pasa la misma y se pueden setear en color, transparencia, ancho del trazo, etc.

Para agregar una polilínea a un mapa, la sintaxis es muy similar al caso de los marcadores.

```
Polyline line = map.addPolyline(new PolylineOptions()  
.add(new LatLng(-37.81319, 144.96298), new LatLng(-31.95285, 115.85734))  
.width(25)  
.color(Color.BLUE));
```

4.3 Interfaz de usuario

La interfaz de usuario es todo aquello que el usuario puede ver y todo aquello con lo que este puede interactuar. Al igual que el cualquier otro componente, la UI puede ser definida en XML y levantada por el *MainActivity*.

Como se comentó en secciones anteriores, la ventaja de utilizar Android Studio como entorno de trabajo brinda entre otras cosas la posibilidad de desarrollar aplicaciones en base a plantillas predefinidas.

A la hora de crear un proyecto, el IDE brinda plantillas de todo tipo y que soportan diversos componentes propios del lenguaje. Por ejemplo, pantallas de *login*, pantallas *fullscreen*, *scrolleables*, entre otras.

Particularmente para este proyecto se optó por el esquema de *Navigation Drawer*. Esta es una pantalla que cuenta con una barra de títulos en la parte superior, y un panel desplegable en el borde izquierdo donde se pueden agregar funcionalidades, mensajes, botones, etc. Muy similar es el caso de la interfaz gráfica de la aplicación de *Twitter* o *Play Store*. Incluso es el mismo formato que utiliza otra de las aplicaciones desarrolladas por la Universidad, *Litus TV*.

Como se ve en la imagen 4.4, es posible agregar botones a la pantalla y cambiar el juego de iconos. Esta es la primera vez que se presenta la interfaz de la aplicación. En las siguientes secciones detallaremos la misma y sus funcionalidades. En la misma se observa el posicionamiento del mapa ocupando la mayoría de la pantalla y dentro de él se observa un marcador. Dicho mapa puede ser desplazado hacia cualquier dirección y se puede acercar o alejar.

Cuando se instala por primera vez, la aplicación solicita los permisos requeridos al usuario antes de presentar la UI de forma completa.

4.4 Geo posicionamiento

En la imagen anterior se puede observar que en el mapa hay un marcador de posición, correspondiente con la posición del usuario en ese momento. Para poder detectar la ubicación del usuario, Android provee un *LocationListener*²⁵. Un *listener* es una interfaz que reacciona a cualquier tipo de cambio o evento que ocurra.

El *LocationListener* monitorea la posición del móvil a partir del sensor de GPS y la conexión a redes, cuando detecta un cambio en la latitud o longitud (depende de la calidad de la señal en ese lugar) ejecuta alguna función. En este caso, actualiza la

²⁵ Más información en <https://developer.android.com/reference/android/location/LocationListener>

posición del marcador de posición del mapa. De esta forma se mantiene la posición del usuario en todo momento conforme se mueve.

4.5 Modelado de la aplicación

Antes de comentar la funcionalidad de la aplicación y ya habiendo revisado los conceptos más importantes, veremos cómo se modela la misma. A continuación presentamos el diagrama de clases de la misma y el detalle de sus componentes en la imagen 4.5.

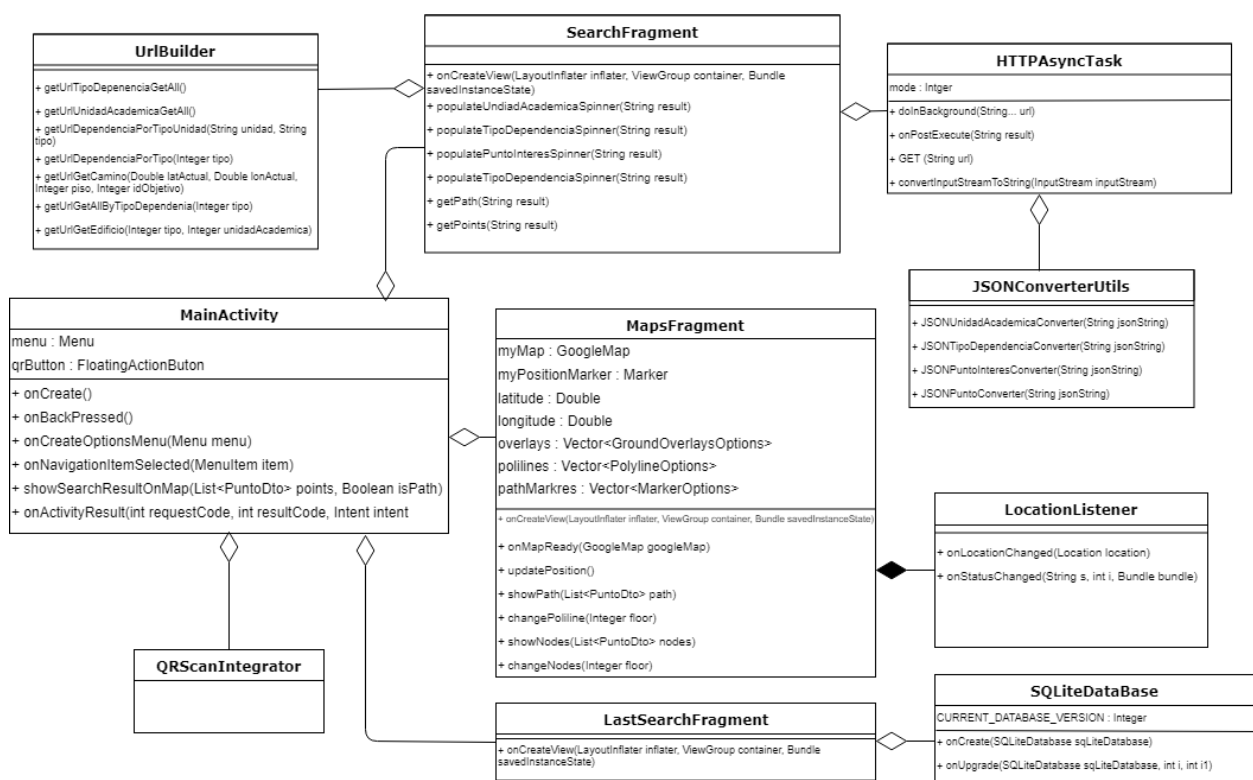


Imagen 4.4: Diagrama de clases de la aplicación

Como se dijo en subsecciones anteriores, el MainActivity es quien maneja el flujo de la aplicación y es la encargada de cambiar las pantallas de acuerdo a las funcionalidades.

Para este proyecto se han utilizado *Fragment Activities*. Un Fragment es una porción de la interfaz de usuario que puede ser añadida, removida o cambiada de la UI como si fuera un módulo independiente²⁶. Así, es posible mantener el marco de la pantalla estático cambiando el contenido de este fragmento.

²⁶ Más información en <https://developer.android.com/guide/components/fragments?hl=ES>

4.5.1 Flujo principal

Cuando se inicia la aplicación, lo que se observa es el *MapsFragment*. El mismo es un fragmento que contiene un objeto de tipo Map como se ve en la imagen 4.5. En dicho fragmento están contenidos todos los objetos y variables necesarias para representar la ubicación del usuario.

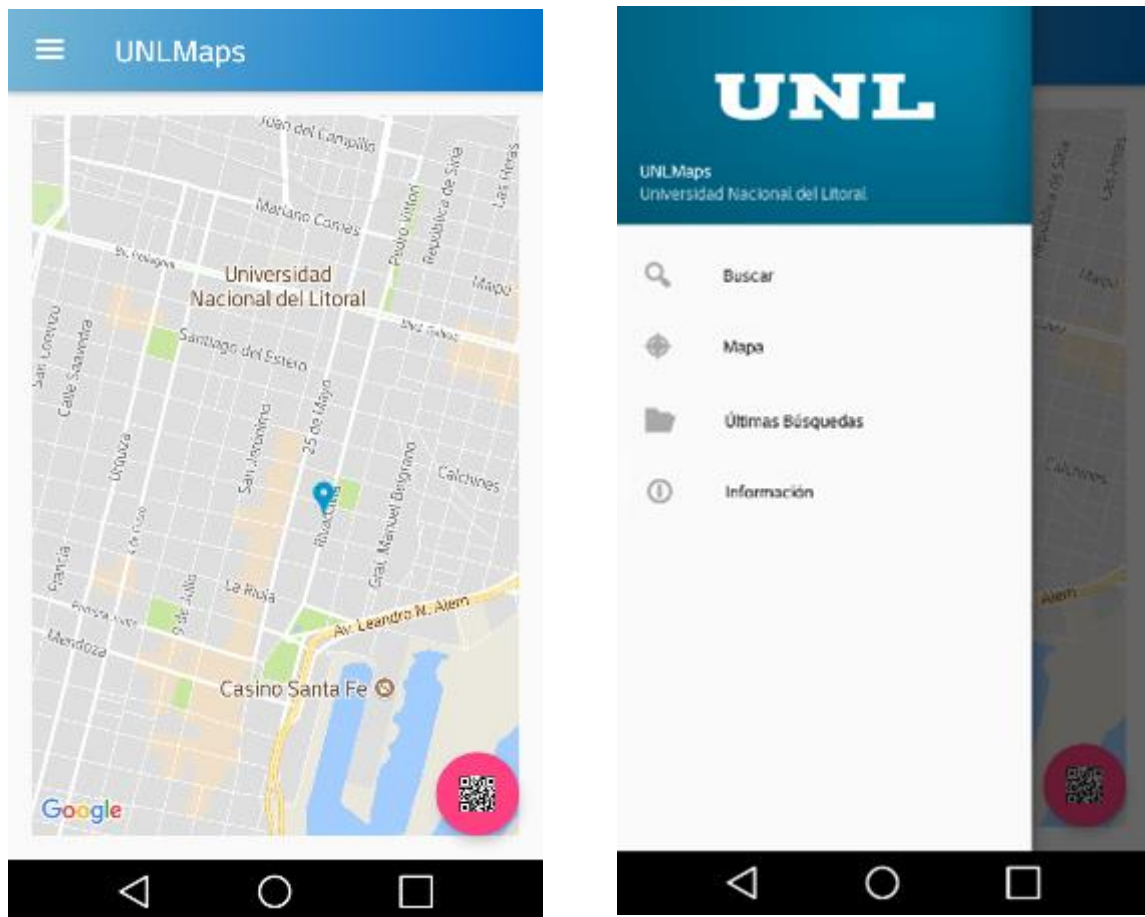


Imagen 4.5: UI de la aplicación

A partir del menú desplegable de la izquierda, se pueden invocar los otros fragmentos que contienen las funcionalidades. También se puede ver en la esquina inferior derecha el botón para activar la lectura de tokens. Más adelante dedicaremos una sección para esta funcionalidad.

A esta altura ya sabemos que la funcionalidad más importante de la aplicación es poder seleccionar algún punto de interés cargado en la base datos y encontrar el camino hacia este. Esta funcionalidad es controlada por el *SearchFragment*.

Aquí como ejemplo, en la imagen 4.6 vemos la acción para buscar el Taller 5 correspondiente a la Facultad de Urbanismo y Diseño.

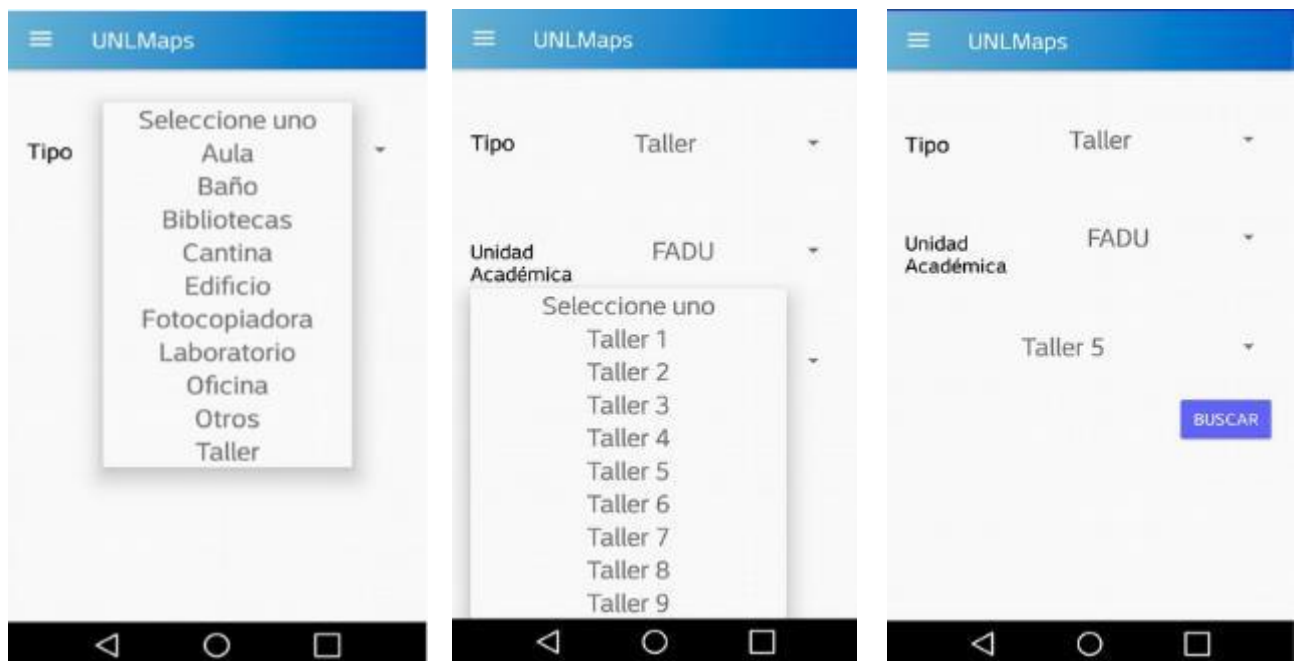


Imagen 4.6: Pantalla de búsqueda

La imagen 4.7 presenta el diagrama de secuencia para el armado de esta pantalla y es análogo al diagrama de secuencias mostrado en el capítulo anterior, sección **Modelo por capas**. Para completar el menú de tipos de dependencias y unidades académicas, se envían dos peticiones simultáneas al Web Service cuyas respuestas en JSON son parseadas por la aplicación. Para el caso del menú de puntos de interés, en la request se envían por parámetros los id de los campos de los menús anteriores, para poder filtrar en la base de datos aquellos puntos que correspondan a la unidad académica seleccionada y el tipo correspondiente.

Una vez seleccionado algún punto, se puede buscar el mismo. La petición que se envía ahora contiene el id en la base de datos del punto en cuestión más la información relativa a la ubicación del usuario.

La respuesta llega en formato JSON y contiene todos los puntos por donde debe pasar la polilínea detallada en la subsección 4.2.3, junto con los marcadores y los *overlays* correspondientes. Este punto se puede realizar ya que de ante mano se conoce el formato en el que vendrá la respuesta, producto del desarrollo realizado en el capítulo anterior.

Es importante mencionar que la construcción de la URL a donde hacer la petición se realiza en la clase `UrlBuilder`. Esta es una clase con métodos estáticos y públicos que posee toda la información necesaria para construir una dirección: URL base, puerto, nombre de controladores y servicios. De forma tal que si se realizara un cambio

de cualquier parámetro en el lado servidor, solo hay que modificar esta clase que abstrae a la aplicación la tarea de determinar la dirección adecuada.

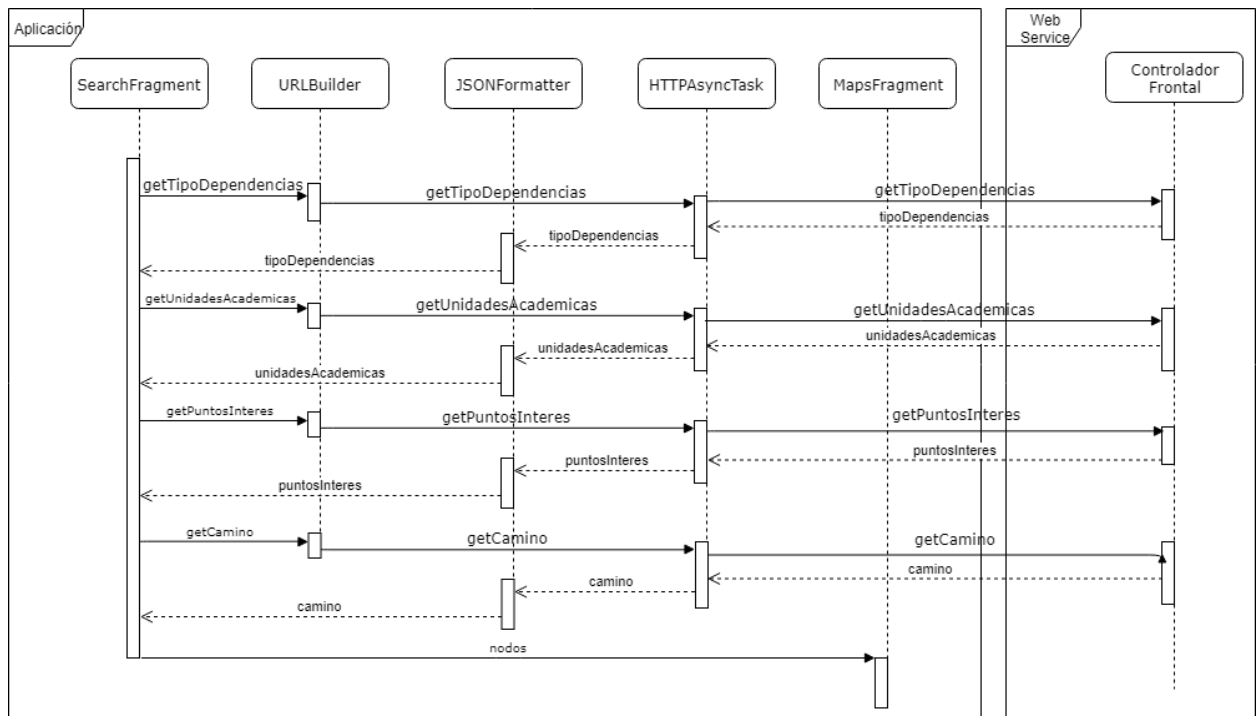


Imagen 4.7: Diagrama de secuencias de la aplicación

La respuesta que retorna el Web Service es recibida y formateada por la clase JSONConverter. Al igual que el caso anterior, esta clase estática de métodos públicos se encarga de transformar el JSON de respuesta en una lista de objetos Java para poder manipularlos. Las razones son las mismas, cualquier cambio en el lado servidor del formato de respuesta solo debe ser modificado en esta clase. Este diseño de implementación responde a aplicar el patrón de diseño *Builder*, que reduce el acoplamiento e independiza el código de construcción de la representación.

El control vuelve a MapFragment quien se encarga de colocar los elementos correspondientes en el mapa como se ve a continuación.

Como se ve en la imagen 4.8, el objetivo búsqueda está en el cuarto piso de la Facultad de Arquitectura y Urbanismo. Los planos se posicionan en su lugar y mediante el menú que está en la esquina superior derecha se puede cambiar el piso de visualización. La aplicación discrimina los distintos tipos de puntos de interés con un icono distinto relativo al mismo.



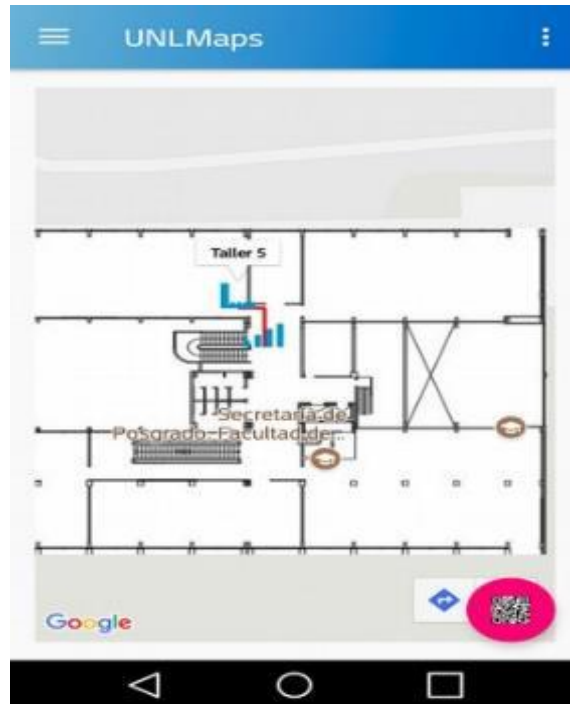


Imagen 4.8: Representación de una búsqueda

4.5.2 Visualización de imágenes

Además de esto, también es posible visualizar para aquellos puntos que lo posean, una foto del lugar que representan. En la imagen 4.9 se puede ver que el marcador del portón principal de Ciudad Universitaria cuenta con este recurso.

Esta funcionalidad está habilitada para aquellos puntos para los cuales se cuenta con una foto que deberá estar alojada en el Web Service, ya que el mismo posee un servicio para enviar estos recursos, que toma la aplicación y los coloca en el marcador correspondiente.

En el caso de que se quisiera trazar un camino para encontrar cierto punto de interés, y el camino entre el usuario y el objetivo pase por un punto que cuenta con una imagen, se visualizará un icono con una pequeña cámara, caso que se puede observar en la imagen 4.10, que al clickearlo desplegará la foto correspondiente, como es el caso de la imagen siguiente, que muestra el camino para llegar al Aulario Cubo.

Estas funcionalidades ayudan al usuario a ubicarse mejor dentro de la Universidad pero también explotan las capacidades del móvil de interactuar con recursos y servicios en la nube.

Esta funcionalidad se pudo lograr gracias a la inclusión de la librería Picasso²⁷ para Android, que permite cargar imágenes en objetos de forma asíncrona.

²⁷ Más información en <http://square.github.io/picasso/>



Imagen 4.9: Marcador con imagen



Imagen 4.10: Camino con marcadores intermedios

4.5.3 Lectura de tokens

Retomando un tópico anterior, es posible ejecutar la lectura de tokens con la cámara del móvil. Para agregar esta funcionalidad se utilizó la librería de Zxing “*Zebra Crossing*”²⁸. La misma es una librería *open source* que permite la lectura de códigos de barra en 1D y 2D implementada en Java.

Requiere la descarga de una aplicación tercera llamada *Barcode Scanner* desarrollada por dicho grupo. Esta aplicación provee una interfaz de visión de la cámara para realizar la lectura y cuando se captura un código válido, la librería dispara la acción que el desarrollador implemente en el código de su aplicación. En este caso, un método que actualiza la posición del usuario.

4.5.4 Base de datos

Android permite realizar persistencias en una base de datos relacional local. En lugar de tener un sistema cliente – servidor de base de datos, la biblioteca *SQLite*²⁹ se enlaza con el programa pasando a ser parte del mismo. Esto reduce la latencia en el acceso debido a que las llamadas a funciones son más eficientes que la comunicación entre procesos. Muchas de las aplicaciones que usamos incluyen instancias de *SQLite* para persistencia de información debido a su tamaño, como por ejemplo Windows Phone, Google Chrome, iOS y por supuesto Android.

Se incluyó una funcionalidad para guardar las últimas búsquedas que hace el usuario, como se ve en la imagen 4.11. Las mismas se persisten en la base de datos y pueden ser vueltas a lanzar desde otra pantalla. Esto ahorra ancho de banda ya que no se deben realizar las 3 consultas previas necesarias, que son consultar la tabla *Tipo_Dependencia*, *Unidad_Academica* y *Punto* para popular los spinners correspondientes. Se almacenan los id necesarios y se realiza solo la request que ejecuta el Web Service para retornar los elementos a mostrar en el mapa.

²⁸ Más información en <https://github.com/zxing/zxing>

²⁹ Más información en <https://www.sqlite.org/index.html>

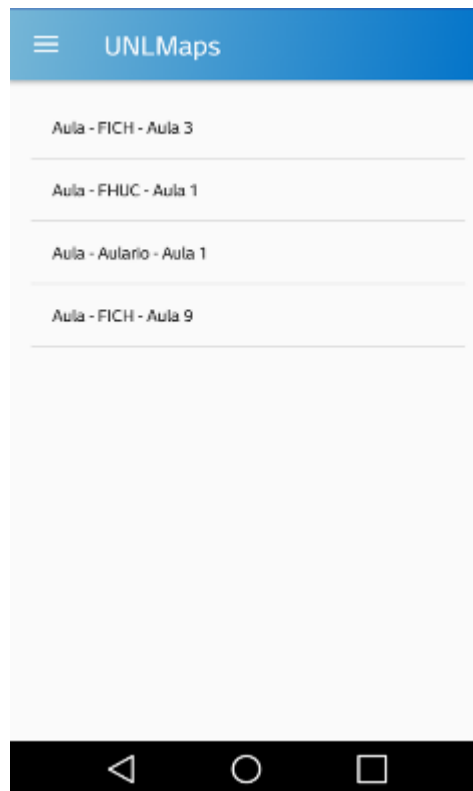


Imagen 4.11: Pantalla de últimas búsquedas

- Capítulo 5 -

5 Resultados

En este capítulo presentaremos más casos de prueba del sistema a modo de resultados para tener un panorama más claro de la representación que se ha hecho en la base de datos de los puntos identificados y cómo interactúa el lado servidor con el lado cliente.

5.1 Casos de prueba

Para efectuar los casos de pruebas de la aplicación, se simuló un servidor en la red local de trabajo y en smartphome de propiedad personal. Para esto se desplegó el Web Service en un equipo cuya dirección IP privada se fijó en 192.168.1.123 a partir de su dirección MAC. Las consultas HTTP desde la aplicación hacia el servidor se hicieron a esa IP, estando el smartphome conectado a la misma red.

A continuación dejaremos algunas búsquedas en las diversas unidades académicas censadas para ejemplificar aún más el grafo de conexiones que se realizó.

En la imagen siguiente se puede ver el caso para buscar las bibliotecas cargadas en el sistema. En este caso son dos, la que está ubicada en el tercer piso de la Facultad de Ingeniería y Ciencias Hídricas y la que está ubicada en el segundo piso de la Facultad de Arquitectura, Diseño y Urbanismo. El proceso es similar al caso mostrado en el capítulo anterior: se selecciona el tipo de dependencia que se quiere buscar y en menú desplegable de abajo aparecerán las opciones.

En primer caso, se muestra en la imagen 5.1 cómo llegar a la biblioteca mencionada, con la particularidad de que el usuario que está consultado la información, se encuentra fuera de Ciudad Universitaria, por lo tanto el camino se muestra desde el portón de ingreso de la misma.

Ahora presentaremos el caso de una búsqueda de un usuario que está posicionado en el edificio de la Facultad de Ingeniería y Ciencias Hídricas y tiene como objetivo un laboratorio de la Facultad de Humanidades y Ciencias, detallado en la imagen 5.2.

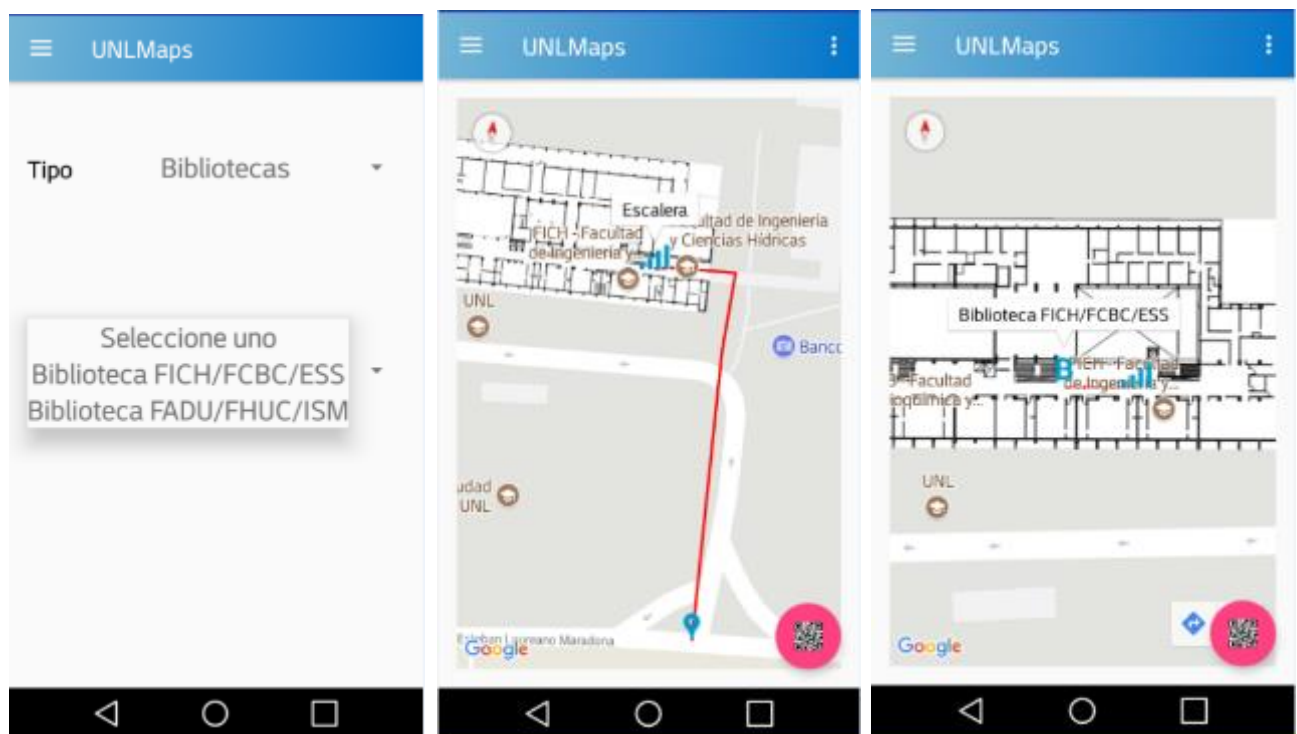


Imagen 5.1: Búsqueda de una biblioteca

Para este caso, el camino se enruta desde el punto en el grafo más cercano al usuario, en este caso la escalera que está en planta baja de este edificio. Al igual que en el primer caso mostrado en el capítulo anterior, el objetivo está en el tercer piso. El usuario manualmente cambia el piso de visualización conforme sube las escaleras o toma el ascensor al piso objetivo.

Aquí se podrá ver que las distintas categorías de búsquedas cuentan con un icono referente a ese punto. Para el caso de los laboratorios, es un matraz de Erlenmeyer.

En la imagen 5.3, se muestra el caso de ubicar el Aula 6 de FICH, que se encuentra en la nave de hidráulica, desde la misma posición que el caso anterior.

En el ejemplo de la imagen 5.4 se puede observar el algoritmo de búsqueda de costo uniforme que ejecuta el servidor. La misma es para encontrar el camino entre el punto de referencia que hemos tomado hasta ahora, y el Laboratorio de Microscopia de la Facultad de Ciencias Médicas. Notar que el camino trazado lleva al usuario por detrás de la Facultad de Ingeniería y Ciencias Hídricas y no por delante como se esperaría.

Por último, para algunas categorías es conveniente no mostrar el camino hacia una de ellas puntualmente, como ser baños, cantinas o fotocopadoras. Esto se pensó así

ya que para estos casos uno generalmente se dirige al primero que encuentra en su camino.

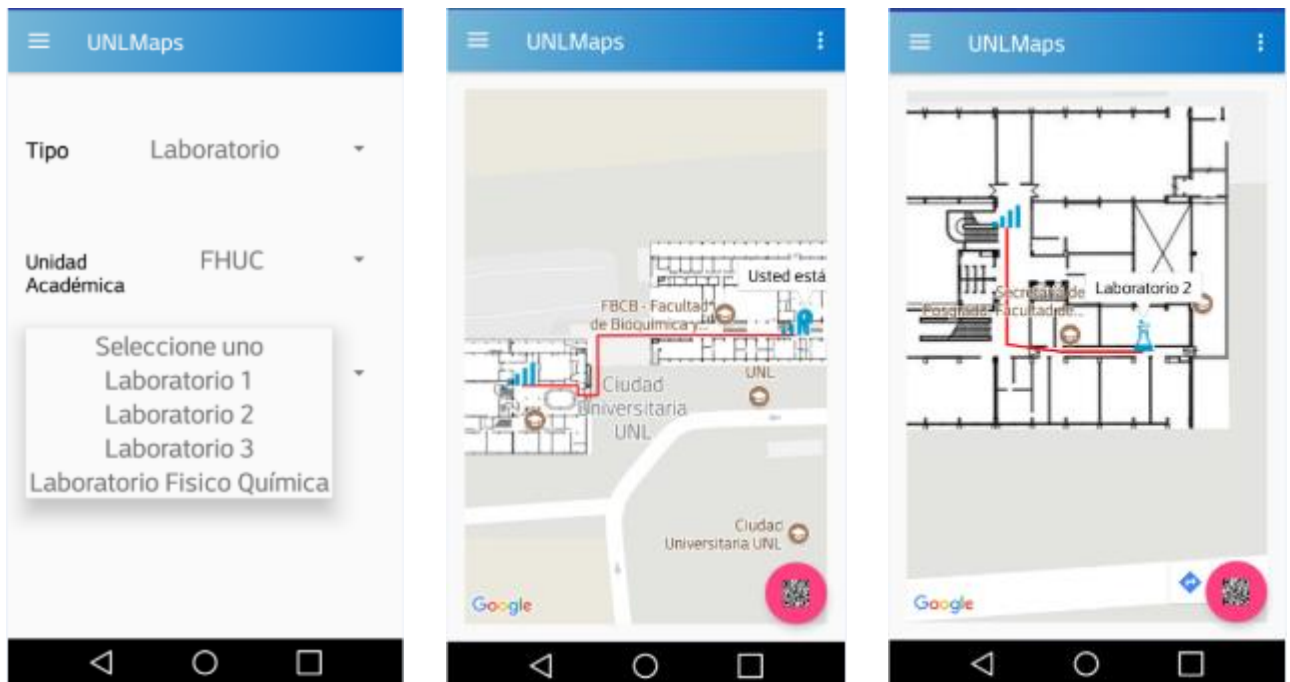


Imagen 5.2: Búsqueda de un laboratorio

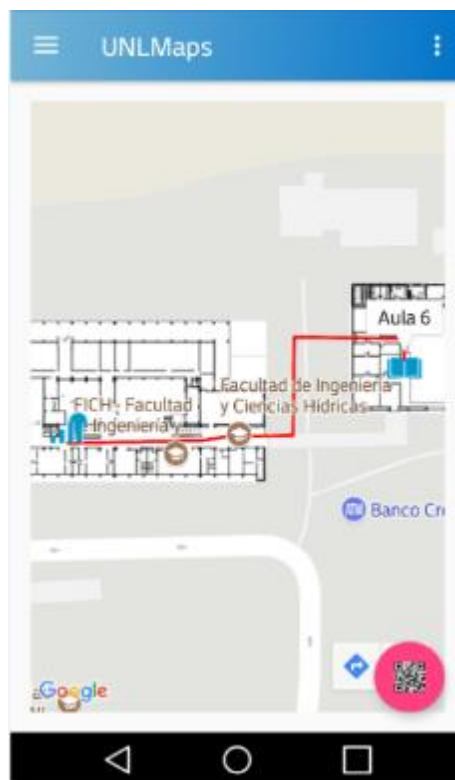


Imagen 5.3: Aula 6 en Nave de Hidráulica

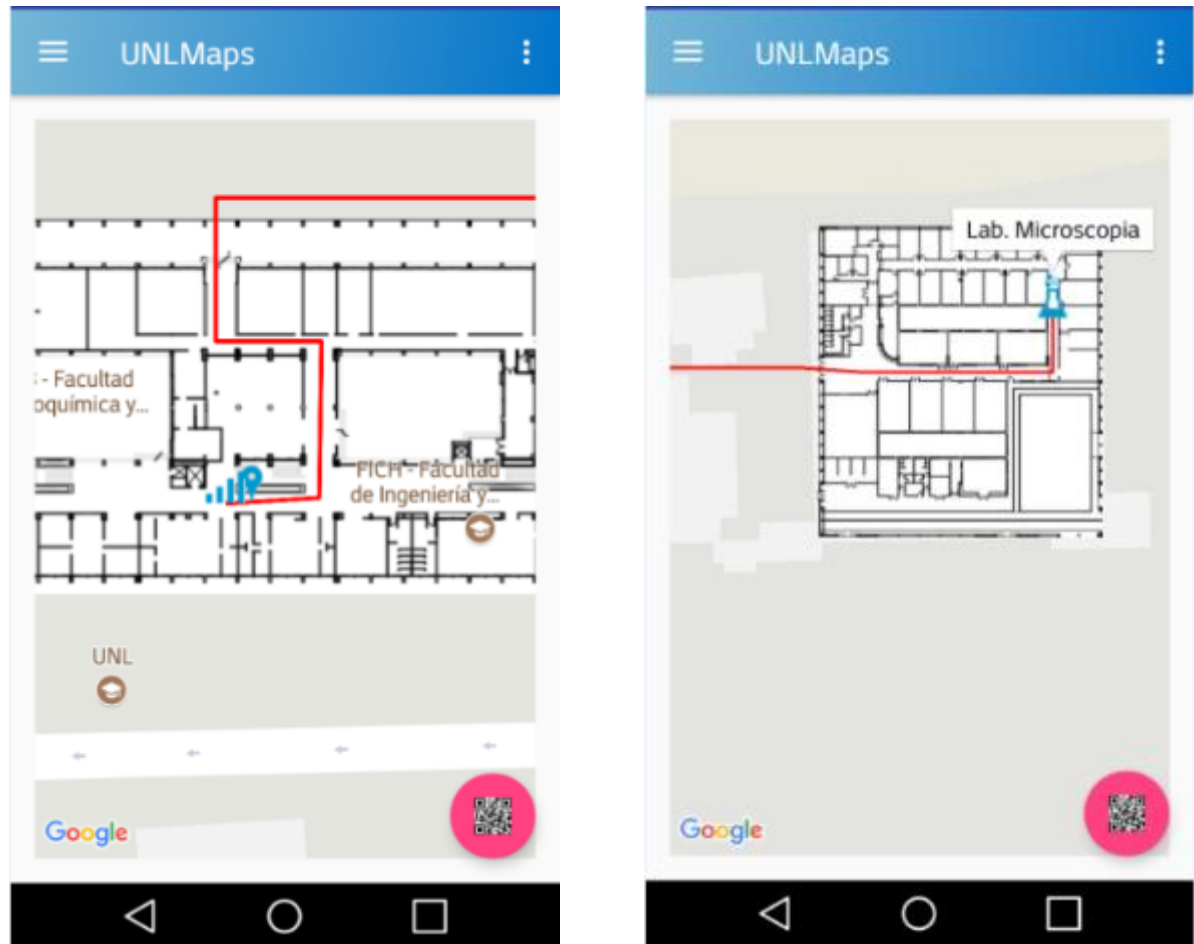


Imagen 5.4: Búsqueda en FCM

Entonces, cuando el usuario busca por alguna de estas tres categorías, el sistema muestra todos los puntos cercanos al mismo, con su ubicación en el mapa y discriminado por pisos. El usuario puede decidir a cual ir.

En este caso vemos que el sistema muestra los baños de planta baja de las unidades académicas cercanas al usuario.

Mediante el menú desplegable que está en la esquina superior derecha se puede cambiar el piso de visualización y así ver, en este caso los baños, de otros pisos.



Imagen 5.5: Búsqueda de baños

- Capítulo 6 -

6 Conclusiones y trabajos futuros

6.1 Conclusiones

En este proyecto se ha realizado un sistema completo, tanto el lado servidor como el lado cliente utilizando las tecnologías que están en boga hoy en día en el mercado. La aplicación desarrollada consume los datos necesarios para representar las búsquedas del usuario del Web Service, que tiene implementada la lógica necesaria para atender las peticiones que a él lleguen.

La importancia de lo anterior radica en desarrollar un sistema que aplique las tecnologías que se utilizan en el mercado y en general en la comunidad de la ingeniería de software. Esto respalda el trabajo realizado con documentación y comunidades activas.

También se ha realizado una base de datos de puntos de interés de las dependencias trabajadas que puede ser exportada y utilizada en futuros proyectos. La misma cuenta con más de 150 puntos.

En función de lo anterior, la representación en grafos de las dependencias de la Universidad permite navegar con facilidad por los edificios trazando las conexiones entre puntos en las zonas correspondientes a los pasillos generando líneas. Aun así, esta metodología dificulta la representación de arcos o figuras curvas, pues para lograrlas se deberían definir una sucesión de puntos que sigan dicha forma.

Se han evaluado diferentes alternativas diseño y modelos en función de las tecnologías que se iban a usar y su adecuación a las funcionalidades básicas que se pretendía alcanzar a partir de los objetivos que se plantearon. Para este caso se buscaron implementar la mayor cantidad de patrones de diseños y buenas prácticas de programación, que hagan que el sistema sea fácilmente mantenibles y escalable en el futuro.

El algoritmo de búsqueda aplicado asegura que el camino obtenido para llegar a cierto destino será el más corto posible, dentro de la representación del grafo. Siendo que la distancia entre puntos es la función de costo del algoritmo, la búsqueda es óptima si se salvan los bucles. Otro algoritmo, digamos de búsqueda en profundidad, sería temporalmente más rápido pero el camino obtenido no sería óptimo. Incluso

podría generar caminos indeseados, por ejemplo, tener que subir y bajar un piso para ubicar un objetivo que estaba en la misma planta que el usuario; debido a que en la disposición de la base de datos este camino está primero en las tablas.

La interfaz de usuario que se diseñó para la aplicación se hizo con componentes básicos de Android nativo, aunque en lo que a cuestiones de diseño gráfico refiere puede resultar un tanto simple; esto se debe a que esta no era una de las finalidades de proyecto, sino la construcción del sistema en sí.

También es importante destacar que se han podido explotar algunas bondades de los *smartphones*, como ser la utilización de sus sensores, conexión a internet, implementación de API y lectura de tokens. Esto hace más interesante al producto en sí al aplicar componentes que tienen cierto contenido de innovación.

El desarrollo de este proyecto se llevó a cabo desde la creación de la base de datos hasta el diseño de la interfaz de usuario, incluyendo el despliegue en un servidor de aplicaciones. Esto involucra un paso por todas las capas necesarias para poner en producción un producto que brinden un entendimiento global del sistema y el proceso de *deployment*.

Es preciso comentar qué, más allá de las soluciones que se buscaron para atacar el problema de la conectividad, igualmente se observaron problemas de geo localización del móvil sobre todo cuando uno se sitúa en la planta de baja de algún edificio que posea varios pisos, debido a esta condición sumado al espesor de las paredes de dichos edificios. Podrían investigarse la viabilidad de otros métodos de posicionamiento, como ser la triangulación por *beacons* o geo localización satelital; aunque estas variantes dependen fuertemente de la infraestructura con la que se cuente. Salvando esta cuestión en particular, la geo localización en Ciudad Universitaria puede logarse sin mayores esfuerzos.

6.2 Trabajos futuros

En función del trabajo realizado hasta aquí, se proponen algunos trabajos futuros para mejorar el funcionamiento o prestaciones de la aplicación y el Web Service. Los mismos devienen de cuestiones que se han analizado en el proceso de diseño y desarrollo del proyecto pero que no se pudieron llevar a cabo por cuestiones ya sea de tiempo o porque quedaron fuera del alcance inicial del mismo, pero sería interesante poder encarar estos puntos.

- Realizar el despliegue del Web Service en un servidor que esté disponible las 24hs y pueda ser accedido mediante una IP pública. Se deberá realizar un *port forwarding* y una asignación de IP privada fija al equipo en cuestión.

- Internacionalizar las claves de texto, para que pueda ser utilizado por usuarios extranjeros.
- Censar dependencias faltantes para cargarlas a la base de datos y agregar a la aplicación los planos correspondientes
- Cumplido el primer punto, se podrá dejar disponible la aplicación en Google Play Store para ser descargada por cualquier usuario y utilizada en cualquier momento. Para esto, también sería recomendable disponer de un pequeño equipo de desarrollo que pueda encargarse de la puesta a punto a nivel productivo de la aplicación, teniendo en cuenta que podrá ser usada por una gran diversidad de usuarios.
- Integrar otros servicios de la Universidad a la aplicación. Por ejemplo, consumir información del Web Service de Bedelía para consultar horarios de cursado

6.3 Comentarios finales

A modo de comentario final, me es justo decir que en mi actual empleo me encuentro desarrollando un sistema de gestión integral para un ente provincial en un equipo de 10 personas. Dicho sistema se basa en un servidor *frontend* que consume de un API REST. El desarrollo del servidor *backend* se está realizando en lenguaje Java junto con Spring y Hibernate, utilizando el módulo Spring MVC entre otros, sirviendo dicho servidor en un contenedor Tomcat.

Las tecnologías mencionadas son las que he utilizado en mi proyecto, que me ha servido de experiencia y me ha dado mejores herramientas para desempeñarme en mi vida profesional. Del mismo modo la manipulación de bases de datos, control de versionado, aplicación de buenas prácticas de programación, arquitecturas REST, el protocolo HTTP, entre otros; son otras cuestiones que me ha dejado el proyecto y que también puedo aplicar día a día en mi labor.

Considero esto importante ya que se alcanzó un objetivo que no estaba planificado como tal, que es el desarrollo personal del alumno.

Bibliografía

- [1] Ander-Egg, Aguilar. *Como elaborar un proyecto*. Edit. Lumen/HVMANITAS, 13va Edición, 1996
- [2] C. Bauer, G. King. *Java Persistence with Hibernate*. Manning, 2nd Edition, 2006.
- [3] D. Coward. *Java EE 7: The Big Picture*. McGraw-Hill, 2nd Edition, 2014.
- [4] I. Darwin, J. Brittain. *Tomcat: The Definitive Guide*. O'Reilly Media, 2nd Edition, 2008.
- [5] F. Gutierrez. *Introducing Spring Framework: A Primer*. Apress, 1st Edition, 2014.
- [6] D. Gourley, B. Totty, S. Reddy. *HTTP: The Definitive Guide*. O'Reilly Media, 1st Edition, 2002.
- [7] C. Haseman. *Android essentials*. Berkeley, CA; New York, NY, 2008.
- [8] C. Hoisington. *Android Boot Camp for Developers using Java, Comprehensive: A Beginners guide to Creating Your First Android Apps*. Cengage Learning, 1st Edition.
- [9] T. Khare. *Apache Tomcat 7 Essentials*. Packt Publishing, 1st Edition, 2012.
- [10] S. Komatinen, D. MacLean. *Pro Android 3*. Apress, 1st Edition, 2009.
- [11] M. Masse. *REST API Design Rulebook. Designing Consistent RESTful Web Service Interfaces*. O'Reilly Media, 1st Edition, 2011.
- [12] M. Murphy. *Android Programming Tutorials*. CommonsWare, LLC, 3rd Edition, 2010.
- [13] S. Pachev. *Understanding MySQL Internals*. O'Reilly Media, 1st Edition, 2009.
- [14] S. Patni. *Introducing Pro RESTful APIs: Design, Build and Integrate with REST, JSON, XML and JAX-RS*. Apress, 1st Edition, 2017.
- [15] Y. Prajapati, V. Ranapariya. *Java Hibernate Cookbook*. Packt Publishing, 1st Edition, 2015.
- [16] Project Managment Institue. *Guía de los fundamentos para la dirección de Proyectos*. PMI Publications, 4ta Edición, 2008.
- [17] D. Poo, D. Kiong, S. Ashok. *Object-Oriented Programming and Java*. Springer, 2nd Edition, 2007.
- [18] W. Raj Amal. *Learning Android Google Maps*. Paperback, 3rd Edition, 2015.

- [19] L. Richardson, S. Ruby. *RESTful Web Services*. O'Reilly, 2008.
- [20] A. Sarin. *Getting started with Spring Framework: a hands-on guide to begin developing applications using Spring Framework*. O'Reilly, 3rd Edition, 2016.
- [21] I. Somerville. *Ingeniería de Software*. México: Editorial Pearson, 9na Edición, 2011.
- [22] G. Svennerberg. *Beginning Google Maps API 3*. Apress, 1st Edition, 2010.
- [23] A. Vukotic, J. Goodwill. *Apache Tomcat 7*. RA-MA Editorial, 1st Edition, 2011.
- [24] C. Walls. *Spring*. Manning/Anaya, 4ta Edición, 2015.
- [25] H. Williams, S. Tahaghoghi. *Learning MySQL*. O'Reilly Media, 1st Edition, 2009.

Anexo