**Ingeniería de Software II**

**Documentación Sprint 1: Hospitalización Domiciliaria**

**Grupo 8**

**Integrantes:**

Francisco Álvarez

Francisco Covarrubias

José Martínez

Nicolás Sobarzo

**Profesores:**

Pablo Schwarzenberg

Gustavo Gatica

Luis Rojas

**Carrera:**

Ingeniería Civil Informática

Índice

[**Introducción - Generación de rutas**](#_ekto4g4cox5j) **2**

[**Leaflet (Biblioteca Javascript)**](#_u7shrn847za1) **2**

[**Búsqueda de ruta óptimas**](#_8gn60s6zz21t) **3**

[**Búsqueda de direcciones**](#_e7mdydffc1ae) **4**

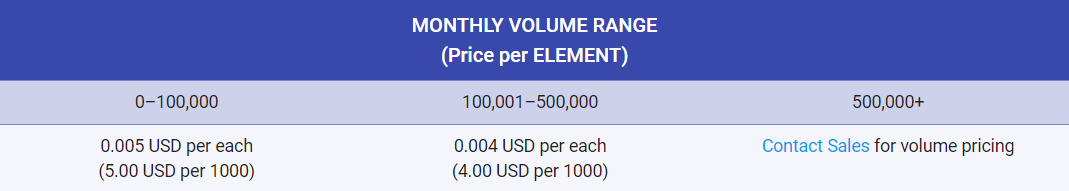
[**Posición actual (geolocalización)**](#_4dk09ndn3qfo) **5**

[**Reportes**](#_td4czuxy6r2x) **6**

[**Visita Pacientes**](#_nmxliehp5v76) **6**

# Introducción - Generación de rutas

Para la generación de rutas entre varias direcciones se pensó inicialmente en utilizar la API de google, lo que, luego de haber conseguido la extracción del cálculo en de las distancias y demoras, fue descartado debido a que se encontró una api gratuita que realiza lo mismo.



**Imagen 1:** Distance Matrix API Usage and Billing.

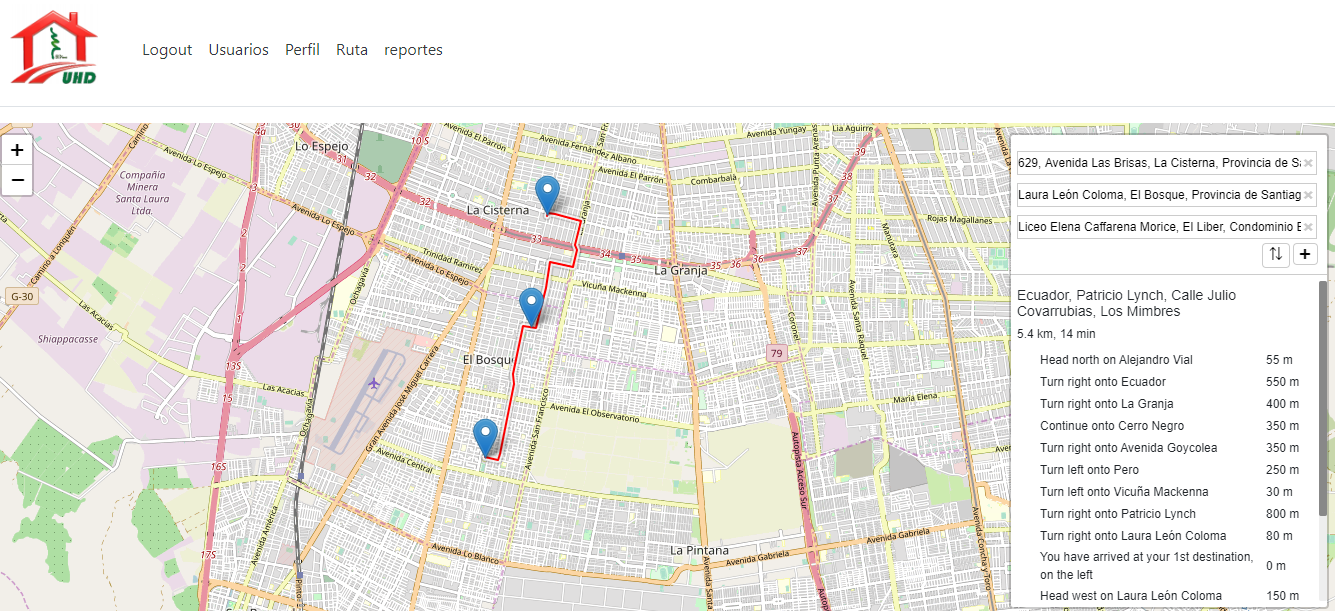
Si bien no es mucho el costo de la API de google (3500 clp por las 1000 consultas), se decidió utilizar una API gratuita debido a que al ampliar la perspectiva de lo que se está realizando, no solo se necesita la API para encontrar la ruta óptima, sino que también hay que utilizar APIs para temas como la ubicación gps en vivo; conseguir la latitud y longitud a partir de una dirección; o la consulta reiterada de la ruta para buscar el orden correcto (explicado más adelante). Tomando esto en consideración, utilizar una APi gratuita que realiza lo mismo sin perder optimización de la ruta es la mejor opción en términos de costos.

# Leaflet (Biblioteca Javascript)

Leaflet es una biblioteca de javascript que se utiliza para generar aplicaciones que requieran el uso del mapa, es de código abierto y gratuita, esta biblioteca tiene una cantidad enorme de plugins para modificar el mapa según sea la necesidad, para efectos de este trabajo se está trabajando con el plugin “Leaflet Routing Machine”, la implementación actual solo genera una ruta óptima entre el punto A y el punto B pero si se le ingresa un punto C calculará la ruta óptima entre A-B y B-C, y no calculará una ruta óptima entre los 3 puntos.

Aquí algunas características del plugin Leaflet Routing Machine:

* Control de folleto estándar, con apariencia y folleto
* Enrutamiento de inicio a destino, con posibilidad de puntos de paso
* Agregue, edite y elimine waypoints a través de la entrada de direcciones y el uso del mapa
* Soporte de múltiples idiomas
* Soporte para varios motores de enrutamiento:
  + OSRM : valor predeterminado incorporado
  + API de instrucciones de Mapbox : integrada en la clase L.Routing.Mapbox; ver el tutorial de enrutadores alternativos
  + GraphHopper - a través del complemento lrm-graphhopper
  + Mapzen Valhalla - a través del complemento [lrm-valhalla](https://github.com/valhalla/lrm-valhalla)
  + TomTom Online Routing API - a través del complemento lrm-tomtom de Mathias Rohnstock
  + Esri - a través del complemento lrm-esri
* Altamente personalizable para uso avanzado
* Aspecto personalizable (tematización / skins)
* Código abierto lanzado bajo licencia ISC (más o menos equivalente con la licencia MIT)
* Definiciones de TypeScript proporcionadas por Chanaka Rathnayaka



**Imagen 2:** Ejemplo de ruta generada con multi-input

# Búsqueda de ruta óptimas

La API de Leaflet como ya fue mencionado, genera la ruta óptima entre las direcciones entregadas, pero presenta el problema de que realiza la ruta en el orden en que son entregadas las direcciones, por lo que se ideó la solución de solicitar varias veces a la API

la ruta óptima mandando como input las direcciones en distinto orden, y así elegir el orden de direcciones que presentan la menor cantidad de tiempo en su ruta.

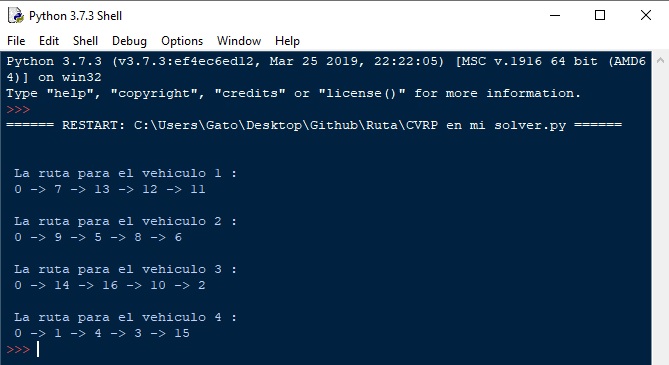
Suponiendo que utilizamos la API de Google, tenemos X cantidad de direcciones sin contar el punto de origen y queremos ordenar estas direcciones en todas las combinaciones posibles para obtener la mejor tenemos la siguiente tabla

|  |  |  |
| --- | --- | --- |
| X! | Cantidad | Precio API Google (CLP) |
| 6! | 720 | 2.552 |
| 7! | 5.040 | 17.867 |
| 8! | 40.320 | 142.934 |
| 9! | 362.880 | 1.029.128 |

**Tabla 1:** Cantidad y precio de las búsquedas de ruta usando APi de Google (con 1 USD = 709 CLP)

Debido a la clara inviabilidad de la utilización de la API de Google y para evitar procesar tanta información con la API de Leaflet es que se decidió que la mejor opción es calcular el orden óptimo con un código desarrollado en python que tiene las siguientes características:

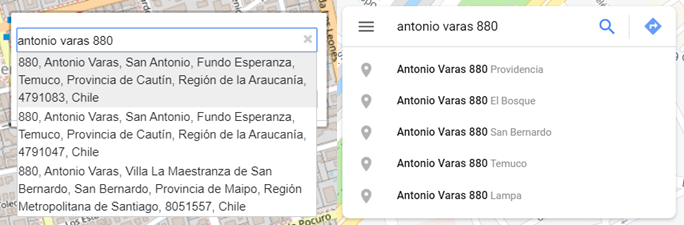
El código escrito en python toma una matriz de distancias entre nodos o de tiempo de viaje entre nodos, recibe parámetros como la cantidad de vehículos en cada ruta y decide a través de la implementación del algoritmo de vecino más cercano una posible solución, entregando una lista ordenada de pacientes a visitar por cada uno de los vehículos asignados.



**Imagen 3**: Vista de rutas posibles entregadas por el programa de python, donde 0 representa el punto de partida y luego el orden de visita a cada paciente enumerado de manera arbitraria.

# Búsqueda de direcciones

Otros de los problemas que encontramos con la API de Leaflet es que, si bien, encuentra las direcciones cuando se buscan por el nombre de la institución (Ejemplo “Hospital el Pino”), esta presenta problemas al momento de recibir una dirección con calle y número como entrada

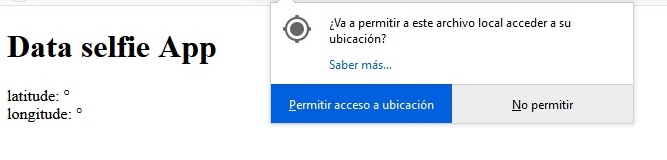


**Imagen 4:** Direcciones en Leaflet vs Google

La solución a esto es usar la API de google, lo que no significa un gran gasto y mejora enormemente la calidad de los resultados obtenidos. Esta dirección solo es necesario buscarla una vez por paciente registrado, esto quiere decir que cada vez que se registra un paciente se registra también su dirección y a ésta se le agregan las coordenadas (latitud y longitud) en la misma base de datos, por lo que se evita tener que hacer la consulta a la API cada vez que se quiera visitar al paciente. Finalmente el precio de estas consulta es de 3.500 clp por cada 1.000 pacientes registrados en la base de datos.

# Posición actual (geolocalización)

Para ver la ubicación en tiempo real en relación a la ruta que se está realizando se planea implementar la API Navigator, la cual informa al usuario del permiso necesario para poder localizar su posición, la cual se actualiza en un pequeño corto periodo de tiempo, con lo cual es posible calcular la ruta según esta nueva posición obtenida. A continuación se muestra un ejemplo de esta API y su funcionalidad.



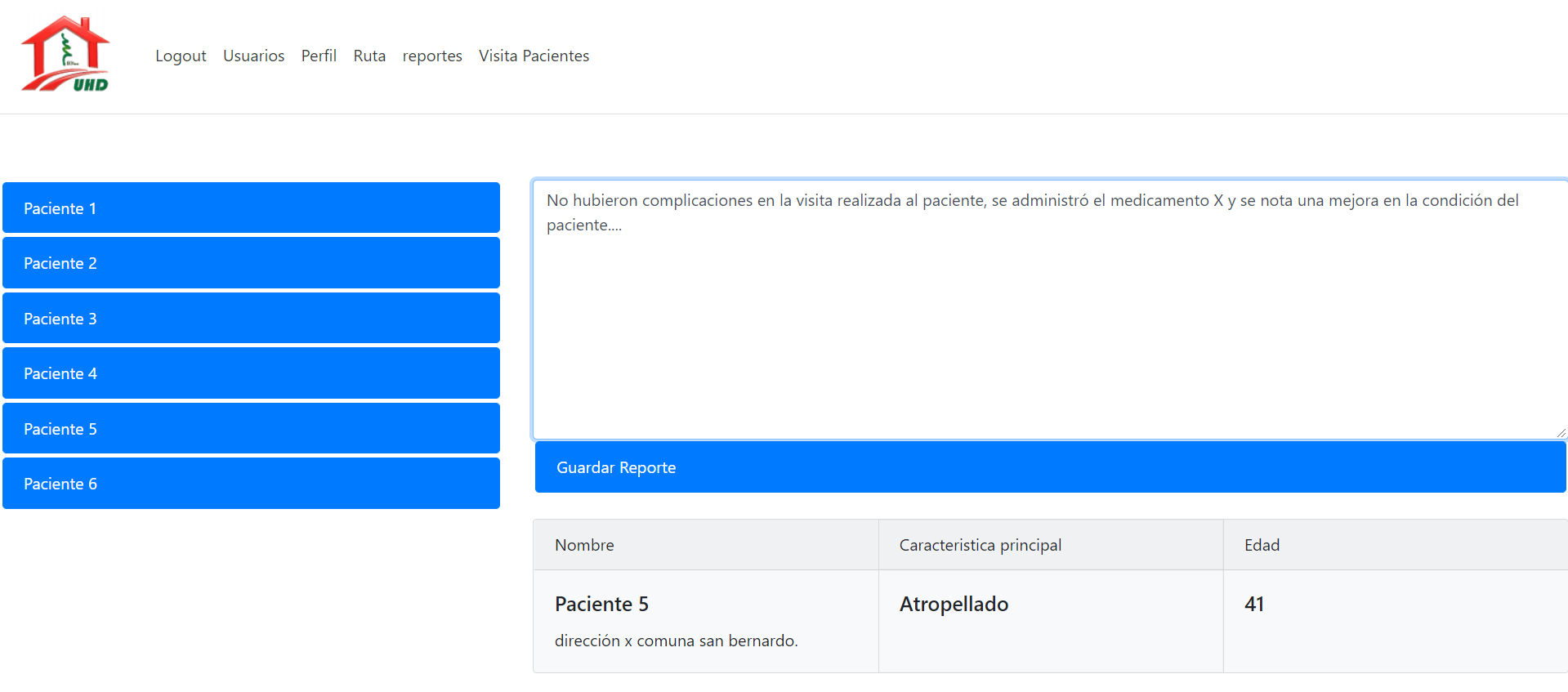
**imagen 5:** permiso de acceso a ubicación de coordenadas.



**imagen 6:** API Navigator entregando las coordenadas del usuario.

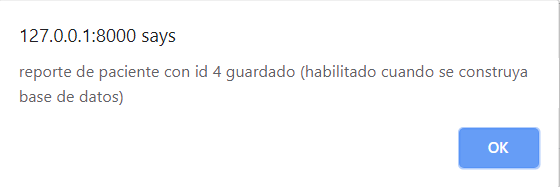
# Reportes

Para que se realicen los reportes se cuenta con una interfaz que permite seleccionar al paciente que se quiere reportar, al seleccionarlo se muestra información relevante para tener una idea más general de quién se trata y al guardar el reporte escrito, se reconoce a qué paciente se le está guardando el reporte



**Imagen 7:** Interfaz de la generación de reportes.

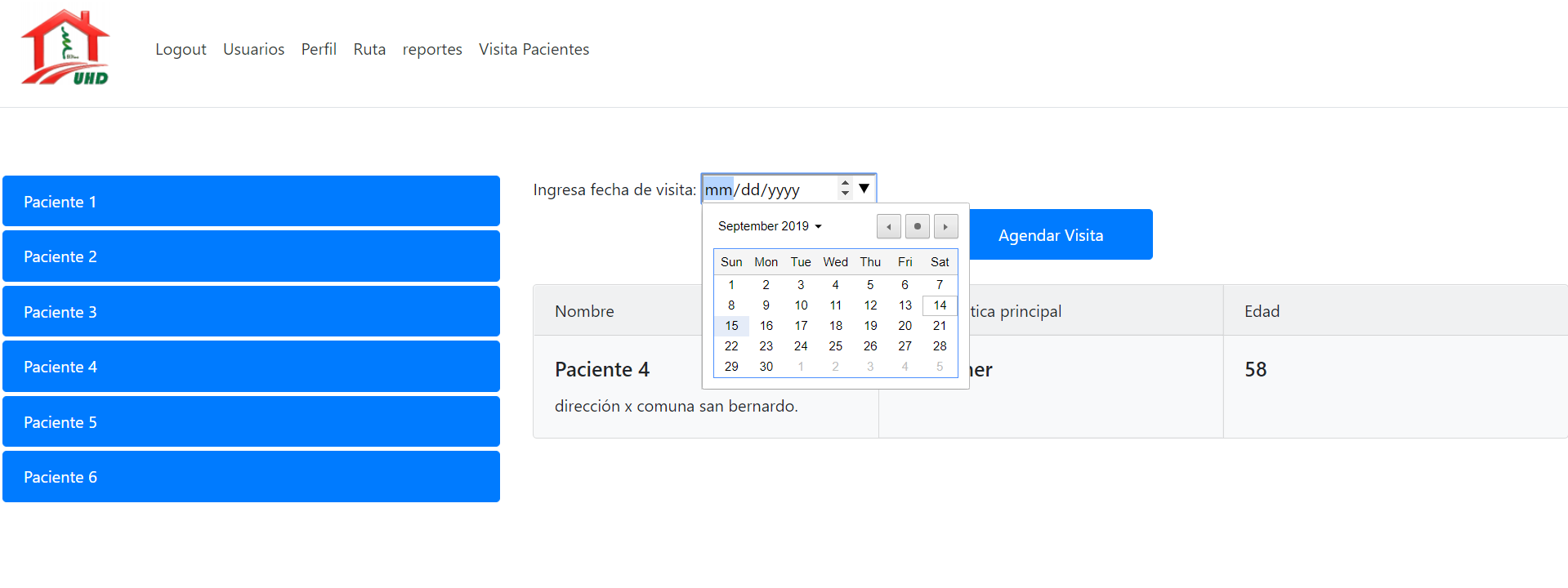
Por ahora varios detalles son temporales (como la información desplegada para cada paciente, el guardado del reporte y el diseño de la interfaz), todo esto debido a que se necesita tener la base de datos implementada para el funcionamiento completo. Por ahora se trabajó en la interacción de los botones y sincronización con dos datos desplegados.



**Imagen 8:** Reconocimiento del paciente seleccionado.

# Visita Pacientes

La agendación se realiza con una interfaz similar a la de reportes, en esta, se selecciona al paciente y se despliega la información general de este y en el cuadro de fecha se despliega la un calendario que permite seleccionar la fecha que se desea. Finalmente, para agendar la visita solo se debe hacer click en el botón “Agendar Visita”



**Imagen 7:** Interfaz de Agendar Visitas.

Al igual que en reportes, para dejar completamente operativa esta página hay que implementar la base de datos. Por ahora se trabajó en la interacción de los botones y sincronización con dos datos desplegados.