UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

# GRAPHCOLLAB: SISTEMA COLABORATIVO OPEN SOURCE PARA EDICIÓN Y MANTENCIÓN DE GRAFOS DE GRAN ESCALA

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN

TOMÁS WILLY WOLF VON BREITENBACH

PROFESOR GUÍA:
JAVIER BUSTOS JIMENEZ

MIEMBROS DE LA COMISIÓN:
JOSÉ PIQUER GARDNER
NANCY HITSCHFELD KAHLER

Este trabajo ha sido financiado por NIC Labs, CORFO

SANTIAGO DE CHILE
2018

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN
POR: TOMÁS WILLY WOLF VON BREITENBACH
FECHA: 2018
PROF. GUÍA: SR. JAVIER BUSTOS JIMENEZ

# GRAPHCOLLAB: SISTEMA COLABORATIVO OPEN SOURCE PARA EDICIÓN Y MANTENCIÓN DE GRAFOS DE GRAN ESCALA

Chile es el país mas largo del mundo con sobre 4300 km de longitud y un ancho promedio de 180 km. Armar una red que abastezca de internet al país naturalmente tiende a unir del extremo norte al extremo sur. Esta tendencia suele comprometer la robustez de la red, haciendo que fallas en esta se propaguen dejando incomunicado al país.

En el marco de los proyectos InnovaChile CORFO se propone llevar a cabo el proyecto "Estudio y Recomendaciones sobre la Resiliencia de la Infraestructura del Internet Chileno".En este proyecto se realizará un mapa que represente como un grafo la infraestructura de lared a nivel país, para luego hacer estudios acerca de la robustez de ésta.

El objetivo es entonces resolver el problema de la creación y mantención de un grafo que represente la red física del internet chileno, mediante una herramienta colaborativa.

Para llevar a cabo el objetivo de la memoria se realiza un análisis del estado previo de la labor de crear y mantener la red, para luego describir casos de uso y definir condiciones para completar el desarrollo. Se hace un estudio de tecnologías para apoyar la creación de la solución y se procede con ella. Se agregan features para creación y modificación del grafo, incluyendo un sistema de historial de acciones para hacer time travel sobre este. Se hace pruebas con datos ficticios para luego usar los datos reales previamente recolectados manualmente.

Se valida la solución en términos de cumplirse los casos de uso definidos, usabilidad de la solución y cumplimiento de objetivos específicos.

El resultado final es una aplicación web que permite la creación y mantención colaborativa de la red de internet chileno.

# Tabla de Contenido

# Índice de Ilustraciones

# Agradecimientos

A mi familia, por todo el apoyo durante mi vida y durante este último tiempo. Los llevo siempre conmigo.

A mis compañeros de laboratorio por siempre proveer una grata compañía. A Felipe por la ayuda durante el inicio del proyecto. A Patricia, por estos años de ayuda y buena onda. A Javier por el apoyo, la buena onda y por tu empatía.

A mis amigos de toda la vida. Los de scout, los del colegio, los de sección, los de carrera, los de carretes, los de los juegos y los de convivir en el dia a dia. Quiero agradecer sin un orden particular a una serie de amigos han sido parte de mi vida durante mis años universitarios: Matias García, Daniel Lopez, David Armijo, Miguel Cisneros, Leonardo Apsé, Yoshiro Mukae, Amanda Ibsen, Elisa Anguita, Mario Fernandez, Benjamín Holloway, Bruno Stefoni, José Bustamante, Bernardo Küpfer, Gonzalo Carrillo, Victor Caro, Daniel Aviv, Jaime Sanz, Salomón Torres, Boris Romero, Sebastián Blasco, Nelson Higuera, Caterina Muñoz, Pedro Franz, Fernando Riveros, Matias Yañez, Bruno Aguiló, Joaquín Galindo, Rodrigo González, Cristian Valdés, Sofía Valenzuela, Pablo Reszczynski, Gabriel de la Parra, Ignacio Pérez, Matia Parra, Ian Fell, Luciano Radrigán, Alvaro Sepúlveda, Byron Cornejo, Fernando Morales, Camilo Morales, Patricio Isbej, Beltran Larrain, Eduardo Illanes, Sebastián Ramos, Daniel Mendez, Sebastián Rossello, Erick Sierra, Karim Pichara, Leonardo Ferraro, Felipe Hantscheruk, Javiera Montero, Diego Bustamante, Gabriel Valenzuela, Camila León, Sebastián Hanson, Sebastián Pössel, Julio Avilés, Joaquín Aguilar, Nicolás Suazo, Alfredo Villalobos, Claudio Casafont, Fabián Astudillo, Ignacio Valdebenito, Bastian Garrido, Diego Valenzuela, Jose Said, Camilo Leon, Fernanda Muñoz, Felipe Garrido, Max Castro, Jose Aguilera y muchos otros que se me quedan en el tintero.

# Capítulo 1

# Introducción

## 1.1. Motivación

Con la alta adopción que han tenido las tecnologías basadas en internet, es crítico que la red no deje de funcionar. En 2015, el uso de servicios de mensajería instantánea via internet como WhatsApp fue superior al 80 % [1]. Hospitales, carabineros y servicios bancarios son también ejemplos de sistemas que no deben dejar de funcionar. Luego del terremoto de 2010 se vio que la red chilena no pudo funcionar correctamente. Se estima que cerca del 80 % de los sitios chilenos estuvieron inaccesibles por varias horas[2]. Podemos ver entonces que tener una red robusta es crítico y fundamental para el correcto funcionamiento del país, y con esto se ve la importancia de crear modelos que puedan sugerir mejoras para ella.

En el marco de los proyectos InnovaChile CORFO se propone llevar a cabo el proyecto "Estudio y Recomendaciones sobre la Resiliencia de la Infraestructura del Internet Chileno". En este proyecto se realizará un mapa que represente como un grafo la infraestructura de la red a nivel país, para luego hacer estudios acerca de la robustez de ésta.

Los grafos suelen ser creados a partir de información que se obtiene de forma automática. Por ejemplo en twitter podemos recorrer a los usuarios y armar un grafo con los usuarios seguidos. Obviando los problemas de cantidad de datos, no hay grandes problemas para crearlo.

Lamentablemente este no es siempre el caso. En específico para el proyecto de estudio de resiliencia, no se puede automatizar este proceso, y se hace necesario el tener una herramienta para modificar y mantener el grafo.

El problema de automatización se debe principalmente a que no hay información geográfica precisa de los nodos. Una primera aproximación para mapear la red es recorrer las distintas redes autónomas que abarcan Chile, revisando las distintas IP conectadas aprovechando los Interior Gateway Protocol para esto. Lamentablemente esto no nos entrega información

[1] http://www.latercera.com/noticia/chilenos-duplican-promedio-mundial-de-uso-de-whatsapp/
[2] https://www.dcc.uchile.cl/node/280

geográfica precisa, y aunque lo hiciera no nos entrega información del camino físico que toman los cables de internet.

En grafos particularmente grandes, el trabajo de mantención manual de este no puede ser hecho por una sola persona (esto tomaría demasiado tiempo), y se hace necesario poder dar soporte para que multiples personas puedan accesar y editar el grafo simultáneamente sin perder la consistencia de este.

El ingreso de los datos para el proyecto de resiliencia proviene de distintas fuentes: SUBTEL, municipalidades, compañías de telecomunicaciones y distintas organizaciones a través del país.

Esta problemática específica es la que se abordará durante el desarrollo de esta memoria: ingreso y curado de datos de fuentes diversas y distribuidas a fin de crear un grafo que detalle del estado actual de la red física de internet en Chile.

Para esto, se crea una herramienta donde se puede visualizar y mantener el estado del grafo, teniendo en cuenta que este deberá ser usado por distintos usuarios y que también deberá ser posible hacer revisión sobre este. Para hacer esta revisión, se registrarán los cambios y los autores de estos, podrá visualizar el grafo en instantes específicos en el tiempo y en caso de necesidad revertir el mapa a ese instante sin peligro de pérdidas de información.

La solución implementada deberá también ser accesible desde un sitio web, de forma que tanto los usuarios que ingresen los datos como los que los editen posteriormente puedan acceder al sitio sin tener que instalar software nuevo, y puedan usarlo desde cualquier dispositivo con un navegador. Es necesario que la localización geográfica del usuario no sea un problema al momento de usarlo.

Para el correcto desarrollo de esta memoria será necesarios conocimientos de aplicaciones web, front-end y back-end, junto con uso de tecnologías de sistemas de información geográfica, como son PostGIS [7] y Leaflet [4].

## 1.2. Objetivos

### 1.2.1. Objetivo General

Resolver el problema de crear, editar y mantener grafos de gran escala de forma colaborativa.

Para esto, se creará una herramienta de creación, visualización, edición y mantención de un grafo de la red física chilena sobre un mapa.

También se creará un sistema que de soporte para validar, ingresar, editar y procesar los datos de la red física de internet en Chile, provistos por compañías de telecomunicaciones, organizaciones, municipalidades y la SUBTEL tal que correctamente represente la infraes-

tructura de internet en Chile.

### 1.2.2. Objetivos Específicos

- Creación de un sistema de recepción de datos.
- Implementación de un sistema de historial de la red, para poder ver qué cambios han sido hechos en este, por quién y los estados intermedios de este.
- Desarrollo de una herramienta para visualizar el mapa de la red.
- Selección e implementación de un método para sugerir conexiones entre las componentes que se dan como input para reflejar el estado actual de los enlaces.
- Hacer pruebas de usuario para asegurar que la solución a la que se llegue sea útil y valorada por los usuarios finales.

## 1.3. Plan de Trabajo

Tomando como base el estado actual del proyecto se definen los casos de uso y condiciones objetivo del proyecto.

Se adapta la base de datos ficticia usada en una versión previa del proyecto para que se soporte el historial de acciones del grafo.

Para el desarrollo de la solución de usa Test Driven Development, en particular en la API que se encarga de la modificación de los datos. De esta forma se asegura la estabilidad de de las herramientas usadas en el back-end.

Se agregan las funcionalidades de mantención del grafo y luego la funciones de time travel en el historial del grafo. Luego de pruebas de usuario se concluye con la creación del menú de acciones para mejorar la usabilidad y experiencia de uso.

### 1.3.1. Metodología

Se usó metodologías ágiles para el desarrollo de esta memoria. Además del uso de Test Driven Development, se trabaja haciendo productos mínimos viables, para luego pulir e integrar al sistema. Se usa este modelo para poder adaptarse rápido al cambio sin hacer trabajo innecesario.

## 1.4. Estructura de la memoria

En el capítulo 2 se analizan las bases de lo que será necesario hacer para llegar a la solución. En el capítulo 3 se proponen las distintas partes de la solución y se muestran detalles

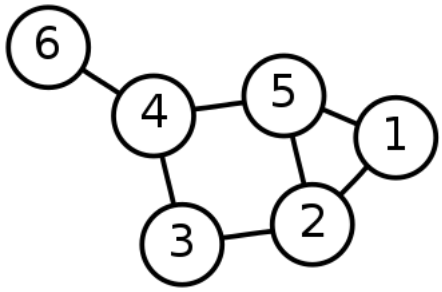


Figura 1.1: Ejemplo de grafo con 6 nodos y 7 arcos

de implementación del trabajo. En el capítulo 4 se hace una evaluación de la solución. Se concluye en el capítulo 5, y se propone trabajo a futuro.

En el Anexo A. se describe el uso del software.

## 1.5. Terminología

### 1.5.1. Grafos

Los grafos se usan para modelar relaciones binarias entre elementos. Los elementos a relacionar son los nodos, mientras que las relaciones son los arcos.

Durante esta memoria se usará cable y arista como sinónimos de arco del grafo. Esto además viene con la implicancia de que el cable tiene una posición geográfica, mientras que tradicionalmente un arco sólo indica el hecho de que nodos distintos están conectados.

Definimos la conexidad entre dos sub grafos (dos conjuntos distintos de nodos de un mismo grafo) si es que existe al menos una trayectoria entre nodos de un subgrafo hacia el otro. En términos de internet, representa si podemos enviar o recibir información de cierto nodo. Si un subgrafo no está conectado al grafo total de la red, se puede considerar como que no está conectado a internet.

La robustez de una red se puede interpretar como la abilidad de soportar fallas y perturbaciones sobre ella. Refiere a que tan resiliente es una red frente a estas fallas y qué tan conexo es un grafo luego de perder conexiones.

### 1.5.2. Áreas

En el trabajo de memoria se menciona el uso de niveles administrativos y áreas. Chile está subdividido geográficamente en 15 regiones. Estas 15 regiones también se pueden subdividir de dos formas distintas. La primera es el nivel administrativo Comuna, y la segunda es el nivel administrativo Provincia, que está compuesto de distintas comunas de una misma región.

Las áreas están definidas con polígonos en formato TopoJSON, para representar sus contornos.

### 1.5.3. Historial

Se define el historial de acciones como la secuencia de modificaciones que se ha hecho sobre el grafo. Este historial de acciones se usa para definir instantes sobre los que hacer time-travel (viaje en el tiempo). El time-travel refiere a ver cual era el estado de la red en un instante de tiempo determinado.

Un rollback refiere a no solo ver el estado del grafo en un tiempo, si no también devolver el estado de la red a dicho estado previo, anulando modificaciones hechas después de este instante de tiempo. Cabe notar que un rollback no elimina permanentemente los datos, si no que los mueve a una tabla de historial. De esta forma los rollbacks son reversibles.

# Capítulo 2

# Bases para la Solución

Dada la necesidad definida de de crear un mapa que represente el internet chileno para distintos finos (como son estudios sobre su robustez), y que tenga capacidad de ser editado colaborativamente, se hace una toma de requisitos.

## 2.1. Requerimientos

- Las fuentes originales de los datos son empresas de telecomunicaciones, organizaciones y municipalidades, entre otros. Dado que el sistema será usado en distintos lugares geográficos del país, no se podrá dar soporte físicamente a los usuarios de este. Es entonces necesario contar con una solución que requiera un mínimo esfuerzo para ser instalada y usada.
- Dado el tamaño de la red se requiere que múltiples usuarios puedan usar la plataforma al mismo tiempo.
- Dado que se busca mejorar la robustez del internet en Chile, es importante saber donde están físicamente los cables. Tener esta referencia geográfica de donde pasan estas líneas ayuda a agilizar la mantención de estas como también reparaciones en caso de corte. Se deriva entonces la necesidad de visualizar la red gráficamente.
- Además de ser necesaria la visualización para estos fines, es también necesaria para asistir en la colaboración y mantención del grafo. Para esto se define la necesidad de crear y eliminar cables, como también la necesidad de unir o conectar cables. Esto es lo que hará que el grafo sea conexo.
- Dado que habrá mantención colaborativa y los cambios pueden requerir validación, se define la necesidad de crear un historial de datos, y capacidad de visualizar estados pasados de la red, y en caso necesario de revertir la red a dichos estados.

Las herramientas que existen hoy en dia para hacer y mantención de grafos se consideran insuficientes. Lo más cercano en torno a visualización de grafos es Gephi[2]. Esta herramienta nos provee una interfaz para visualizar y explorar grafos, pero no tiene soporte para creación colaborativa de estos. Además, Gephi no posee soporte para editar gráficamente los datos, por lo que su uso para mantención de estos es deficiente.

Dados los requerimientos recién mencionados y las herramientas insuficientes que existen actualmente es que se motiva el diseño de la solución implementada durante el desarrollo de la memoria.

## 2.2. Tipos de Usuario

Se definen primero los 3 tipos de usuario para el sistema:

- Visita: Este usuario puede visualizar el estado del grafo, como también visualizar sus estados previos.
- Editor: Este usuario puede visualizar el grafo como también crear, borrar, unir y conectar cables. También puede ver los estados previos.
- Administrador: Puede visualizar y mantener la red. También puede ver los estados previos, y además puede hacer rollbacks[1] en el tiempo.

## 2.3. Casos de uso

### 2.3.1. Visualización

Partiremos definiendo los casos de uso compartidos por todos los usuarios:

- Usuarios seleccionan nivel administrativo de áreas.
- Usuarios seleccionan área en el minimapa y se despliega la visualización de la red en dicha área.
- Usuarios seleccionan área buscando por nombre y se despliega la visualización en dicha área.
- Usuarios haciendo click revisan información del cable, incluyendo nodos inicial y final, id y tipo de cable.
- Usuarios pueden visualizar estados previos de la red eligiendo un momento en específico.

[1]Se retorna la base de datos a estados previos.

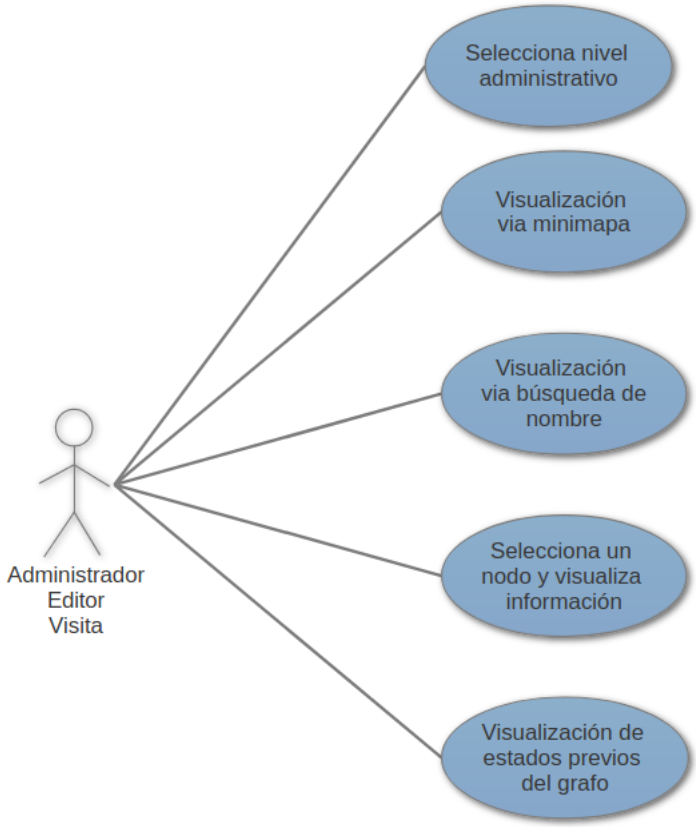


Figura 2.1: Casos de uso de visualizacion

### 2.3.2. Edición

Luego están los casos de uso de nivel edición, compartidos por Administrador y Editor:

- Usuarios crean cables.
- Usuarios eliminan cables.
- Usuarios unen[2] cables.
- Usuarios conectan[3] cables.

[2]Unir cables toma dos cables distintos y los transforma en un único cable sin perder la integridad de los nodos iniciales y finales del cable final

[3]Conectar cables toma dos cables distintos y conecta un endpoint de cada uno a través de un nodo. Esto nos ayuda a mantener un grafo conexo.

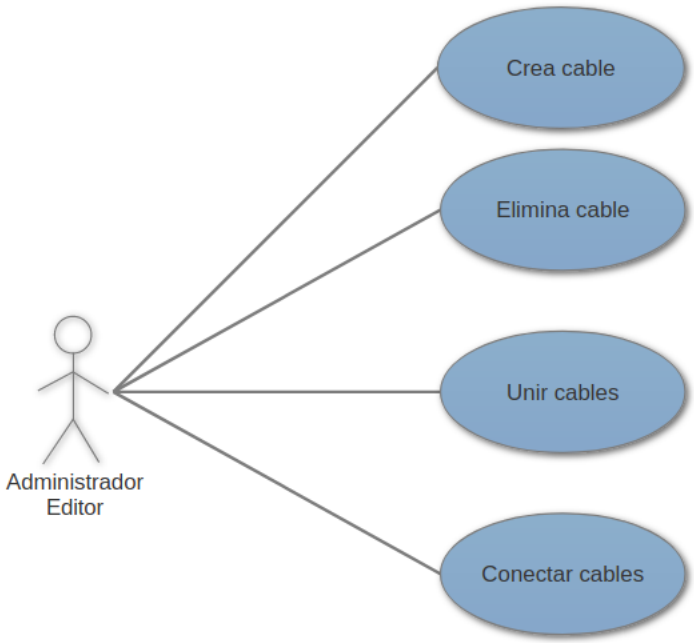


Figura 2.2: Casos de uso de edicion

### 2.3.3. Administración

Al final tenemos el caso de uso exclusivo del administrador:

- Usuarios hacen rollback sobre el grafo eligiendo un momento en específico.

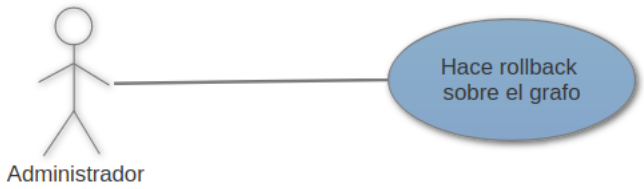


Figura 2.3: Casos de uso de administración

## 2.4. Decisiones de implementación

Dado que se requiere una solución que requiera mínimo esfuerzo para ser usada, una aplicación web es una buena forma de lograrlo. Funcionaría nativamente en distintos tipos de plataformas y no requeriría ninguna instalación por parte del usuario.

Todo el software utilizado tiene licencias Open Source, para poder proveer una solución puramente Open Source.

El uso en particular de Flask como framework se debe a estar basado en python, por tener experiencia previa en este lenguaje de alto nivel. Otra razón es que dados los casos de uso definidos, el estilo minimalista de flask acelera el desarrollo, sin sobrecargarse usando frameworks mas grándes, como Django.

El uso de d3 para hacer el minimapa se debe a la naturaleza de esta herramienta de poder crear visualizaciones basadas en datos (en este caso en particular los datos TopoJSON de las áreas).

El uso de leaflet para visualizar el mapa y el grafo se deben a que también existe un plugin que ayuda a dibujar sobre este. Leaflet también posee un rendimiento deseado. Pruebas muestran que soporta grafos con 60 veces más elementos que el caso real.

Uno de los puntos mas importantes para el desarrollo de la solución es la implementación del sistema de historial de datos. Este historial deberá proveer información acerca de las acciones realizadas sobre el grafo, quien las hizo y cuando fueron hechas. A partir de esta información de las acciones, se deberá ser posible dibujar el grafo en estados anteriores en el tiempo, para luego analizar la evolución de este.

Esto viene también motivado por la necesidad de validar la consistencia e integridad del grafo con que se trabaja. Un ejemplo claro en que la revisión de los datos es importante es el caso de Wikipedia, una enciclopedia colaborativa en que cualquier persona puede hacer contribuciones a un artículo aunque estos posteriormente serán editados y validados por la comunidad y editores contratados.

Analizando el caso de Wikipedia y su modelo de datos podemos guiar la solución a implementar. En Wikipedia, las páginas son el núcleo del modelo de datos. Campos relevantes en esta tabla son la page_id de la página (no cambia a través de ediciones) y page_latest, que referencia la última revisión hecha a la página.

Las revisiones en si están asociadas a tablas donde se guarda el contenido que muestra cada página. En la tabla de revisiones se encuentra toda la información acerca de la edición, como por ejemplo el id de revisión, de la página que referencia, del texto en esta revisión, resumen de la edición hecha por un editor, id del usuario que hizo la edición y un timestamp para la revisión. De esta forma, se puede buscar todas las revisiones por página y se pueden ordenar por fecha para obtener un historial de cambios en ellas.

El modelo de datos preliminar a utilizar está basado en Slowly Changing Dimension de tipo 4 [8]. Éste se basa en usar tablas de historial, en que una tabla contiene la información actual del dato y se usa una tabla adicional para mantener el registro de las ediciones hechas, sus fechas y quien la editó.

# Capítulo 3

# Diseño e Implementación de la solución

En el presente capítulo se describe y detalla el diseño e implementación del sistema desarrollado durante la memoria y las justificaciones detrás de las decisiones tomadas. Inicialmente se explicará el cómo se decide completar las condiciones de los casos de uso, para luego continuar con una explicación las partes del sistema para mostrar de forma incremental los usos de diferentes tecnologías.

Para cumplir con los aspectos de visualización de los casos de uso definidos por los requerimientos se usa una interfaz de 2 partes principales. La primera es un menú lateral de navegación, donde se podrá elegir el área a visualizar manualmente desde un mapa, como también a través de búsqueda de texto de área.

La idea es que una persona pueda llegar al área que busca de forma interactiva, dado que sabe la posición relativa del área respecto al mapa de Chile, como también poder buscar diréctamente un área en particular, sea para ahorrar tiempo o si no se sabe la localización del área.

Para visualizar el estado de la red, se usa un mapa del terreno interactivo en base a Leaflet, donde se despliegan los cables de la red en un área en particular.

Para poder visualizar la red en un tiempo pasado, se selecciona una acción del historial (que viene con un timestamp en particular), y se muestra el estado del grafo luego de dicha acción.

Para cumplir los aspectos de edición de los requerimientos, se hace uso del plugin Leaflet.Draw. Gracias a él se puede dibujar cables nuevos y borrar cables existentes.

Además, se puede unir o conectar los cables manualmente, seleccionando los nodos en particular sobre los que trabajar.

Para cumplir con el caso de uso de rollback de la base de datos, se decide un punto en el tiempo y por debajo el software se preocupa de guardar toda la información relevante, elegir qué cables y que nodos deberán estar vivos en el instante elegido, y luego se modifican y guardan los cables para mostrar que están nuevamente activos.

Se procede a explicar la arquitectura del sistema, mostrándose el funcionamiento básico del backend usando el framework Flask y se justifica el modelo de datos utilizado. Posteriormente se muestra las tecnologías que usa el front-end y como fueron usadas en la implementación, como el controlador maneja los datos y al final una breve explicación del testing que se hizo.

## 3.1. Arquitectura del sistema

Se usa el patrón Modelo-Vista-Controlador, justificándolo con que ya se había usado previamente, y que el framework utilizado lo implementa de forma muy sencilla.

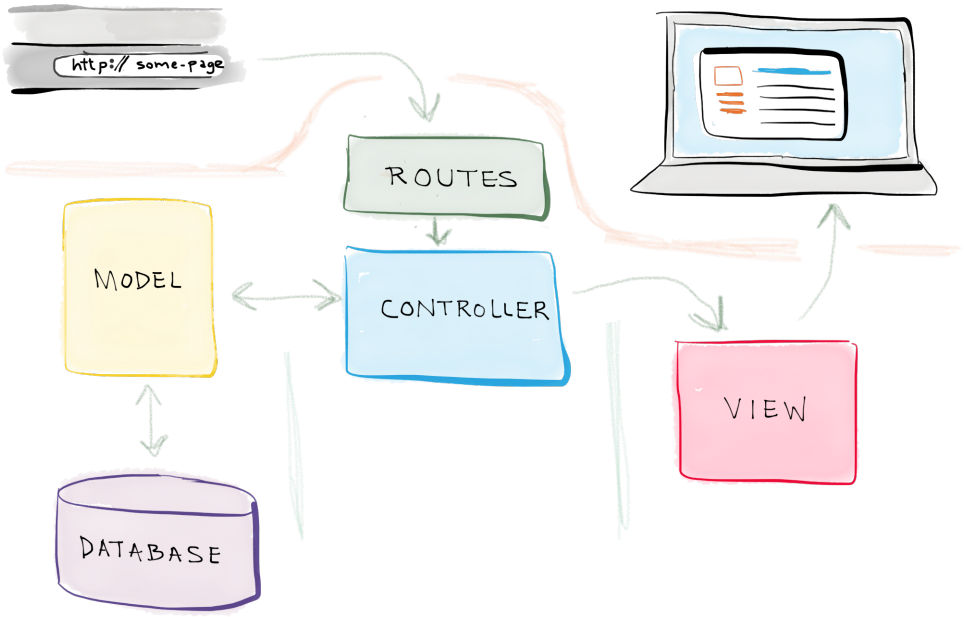


Figura 3.1: Flask MVC

Observando la figura 3.1[1] podemos ver el funcionamiento de MVC. La interacción parte con un *request* HTTP hacia una URL para obtener una cierta página. La URL es capturada por el controlador, quien decide que información deberá ser mostrada al usuario. Si esta información requiere acceder a la base de datos, el controlador se comunica con el modelo para comunicarse con la base de datos. El controlador luego retorna una vista usando los datos obtenidos y se renderea el contenido HTML que el usuario ve en su browser.

### 3.1.1. Base de datos

La base de datos a utilizar es PostgreSQL. PostgreSQL es un sistema para manejar bases de datos objeto relacionales, con énfasis en extensibilidad. La elección de la base de datos se hizo principalmente pensando en poder guardar y manipular datos geoespaciales. Existe la extensión PostGIS que agrega este soporte para datos GIS[2]. Dado el caracter open-source del

[1] https://realpython.com/blog/python/the-model-view-controller-mvc-paradigm-summarized-with-legos/
[2] Geographic Information System

proyecto, cabe notar que ambos PostgreSQL y la extensión PostGIS poseen licencias libres, PostGIS usando GPL[3].

### 3.1.2. Justificacion del historial

Dada la necesidades de visualizar estados previos del grafo y de matener un historial de acciones y edición, se guía la solución con el estudio del Slowly Changing Dimensions (SCD). SCD comprende y reconoce distintos tipos de metodologías establecidas para mantener la información sobre datos que cambian en una base de datos.

Las distintas metodologías pueden ser descritas de la siguiente forma:

- La metodología SCD tipo 1 simplemente sobreescribe los datos, y por lo tanto no mantiene información acerca de cambios, solo mantiene una tabla activa.
- El tipo 2 funciona agregando columnas extra a la tabla, por ejemplo con una columna de versión que se incremente, y en caso de ser necesario usar el dato, se busca el dato con versión mas alta.
- El tipo 3 funciona agregando un nuevo atributo que mantiene información acerca de versiones anteriores. Por ejemplo en un cable se puede guardar cual era la versión original del este. Otra versión de este es en que un atributo se usa para saber cual era la versión anterior del cable mismo. Lamentablemente, este sistema preserva una historia limitada.
- El tipo 4 es el normalmente conocido como tabla de historial. En esta tabla se guardan todas las versiones anteriores de un objeto, indicando las fechas de las ediciones e información relacionada.

En el caso particular de este proyecto, la tabla de historial es quien mejor guarda la información para propósitos de viajes en el tiempo. Además, se agrega una tabla de acciones, donde se guardan los timestamp de las acciones hechas en el sistema, el usuario que las hizo y los cables involucrados en dichas acciones. La tabla de acciones facilita mantener las fechas en que ha habido cambios sobre los datos.

Inicialmente se optó por usar triggers SQL para mantener la tabla de historial y de acciones. Un trigger sería gatillado cada vez que se hace una operación sobre la tabla de datos activos, guardando la información del usuario que hizo el cambio además de los datos antes de operar con la información de la tabla de datos activos. Se deja de lado esta opción por varias razones. Los triggers funcionan a nivel de base de datos, haciendo que la lógica del sistema esté en la capa de base de datos. Esto opone resistencia a la arquitectura MVC previamente seleccionada. Otra razón extra es que es difícil analizar el comportamiento de los datos a menos de que se esté activamente conciente de que hay cambios que se hacen a nivel de bases de datos, haciendo complicado el proceso de debug. La última razón para no usar triggers es que no se puede mantener información del usuario que hizo los cambios de forma natural. Las conexiones del modelo a la base de datos están todas hechas bajo un mismo usuario que se comunica con la base de datos. Para poder proveer la funcionalidad de triggers, cada usuario

[3]GNU General Public License

del sistema deberá tener un usuario en la base de datos, lo que es un proceso engorroso y poco intuitivo para un sistema MVC.

Se decide entonces que la lógica detrás de la mantención del historial de los datos deberá estar en el controlador de flask que funciona con python.

## 3.2. Backend y framework

Se usó flask como el micro framework a elección para el desarrollo del sistema. Flask se caracteriza por ser minimalista, presentando un uso fácil de aprender teniendo todo el poder de python a su disposición. En su minimalismo, flask no provee nativamente una capa de abstracción de base de datos ni componentes que usen librerías externas (third-party).

Sin embargo, Flask es capaz de soportar extensiones que pueden implementar dichas funcionalidades como si estuvieran nativamente provistas por el framework.

Ejemplos de extensiones usadas en el proyecto son Flask-Bcrypt (provee hashing seguro para creación de usuarios), Flask-Login (para manejar permisos y login), Flask-SQLAlchemy (para comunicar el modelo de datos con la base de datos) y Flask-Testing (para hacer tests unitarios).

Flask nos permite hacer templates html para las vistas usando el motor Jinja2.

### 3.2.1. Login

En el siguiente código se podrán ver varias características de Flask, ejemplificadas en la validación de usuario de login del sistema:

```
1  @app.route('/login', methods=['GET', 'POST'])
2  def login():
3      if request.method == 'GET':
4          return render_template('login.html')
5
6      username = request.form['username']
7      password = request.form['password']
8      user = load_user(username)
9      # validate user credentials
10     if user is not None and user.validate_password(password):
11         login_user(user)
12         next_page = request.args.get('next', "mapa")
13         return redirect(next_page)
14     return render_template('login.html', error=True)
```

En la primera línea se puede ver el cómo se define una request. Se usa la anotación `@app.route`, que define las rutas a aceptar y los tipos de request HTTP aceptados. Si el

request es de tipo *GET*, se asume que el usuario quiere ir a la página de login, por lo que se retorna el render del template HTML que corresponde al login. En caso de ser de tipo *POST* se valida al usuario: en caso positivo se redirige al usuario al mapa, y en caso negativo el servidor retornará el render del login especificando que hubo un error de login.

### 3.2.2. Uso de templates

Un caso básico de uso de templates es el de permitir que una tabla que extienda a otra inserte código en un área específica. Un ejemplo es en el head de un documento para que se pueda agregar distintos metadatos y recursos extra.

Un ejemplo de uso en el proyecto es para implementar el menu de navegación.

```
<head>
    <script src="https://code.jquery.com/ui/1.12.1/jquery-ui.js">
    </script>
     ...
    {% block head %}{% endblock %}
</head>
<body>
<div id="mySidenav" class="sidenav">
     ...
</div>
{% block content %}{% endblock %}
</body>
```

Los bloques *content* y head serán reemplazados cuando una template que la extienda la llame y defina el contenido del bloque, como podemos ver a continuación:

```
{% extends "base.html" %}
{% block head %}
    <script type="text/javascript" src="static/js/d3.v3.js"></script>
     ...
{% endblock %}
{% block content %}
    <div id="panel" class="mypanel" hidden>
     ...
{% endblock %}
```

De esta forma, al extender un template se da flexibilidad de desarrollo y orden para el código.

### 3.2.3. Uso de modelos ORM SQLAlchemy

Para acceder a la base de datos, se crean objetos que mapean a PostgreSQL, una base de datos relacional.

```
class Cables(Base):
    __tablename__ = 'cables'
    id = Column(Integer)
    route = Column(Geometry(geometry_type='LINESTRING'))
    type = Column(String(30))
    endpoint_1 = relationship(Nodes, primaryjoin=
        "(Cables.endpoint1_id == Nodes.id)")
    endpoint_2 = relationship(Nodes, primaryjoin=
        "(Cables.endpoint2_id == Nodes.id)")
```

Estos modelos luego pueden ser llamados en python, sin necesidad de usar diréctamente código SQL:

```
def cables_connected_to_node(node_id, given_session=session):
    return given_session.query(Cables).filter(
        or_(Cables.endpoint1_id == node_id,
            Cables.endpoint2_id == node_id)).all()
```

Para obtener el conjunto de los cables conectados a un nodo en particular, la sesión de la base de datos hace un query de la tabla Cables previamente definida, y luego filtra solamente los cables en que su punto inicial o final son el nodo que se busca.

### 3.2.4. Conexión directa a PostgreSQL

Otra forma de acceder a la base de datos es saltándose el modelo y ejecutando queries diréctamente sobre PostgreSQL.

```
    def setUp(self):
        ...
        self.conn.execute("CREATE DATABASE resilienciatest
            WITH TEMPLATE resilienciatemplate OWNER tw")
```

En este ejemplo, se crea una base de datos sobre la que hacer tests, basándose en un template previamente definido.

## 3.3. Modelo de datos

### 3.3.1. Usuarios

Una función fundamental del sistema es el permitir ciertas acciones a solo ciertos usuarios.

Los usuarios se registran con un mail y password. Se guarda el hash de la password en vez de esta misma para agregar seguridad ante ataques. Esto funciona usando various rounds con-

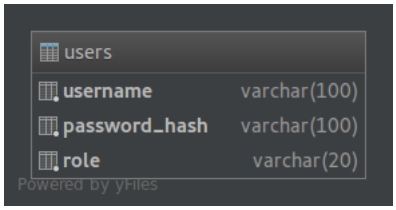


Figura 3.2: Tabla users

catenados de funciones de hash. Este método es intencionalmente caro computacionalmente, para hacer que sea infactible un ataque destinado a vulnerar las contraseñas[6].

Dado que hay interés en mantener la base de datos a través de ediciones, es necesario proveer roles para los usuarios. Los tipos de usuarios serán el usuario básico y el revisor. El revisor podrá hacer rollbacks en el tiempo.

### 3.3.2. Grafo

Para representar un grafo hay que preocuparse de dos componentes. Los nodos y los arcos, que representarán los nodos y los cables en el mapa de internet.

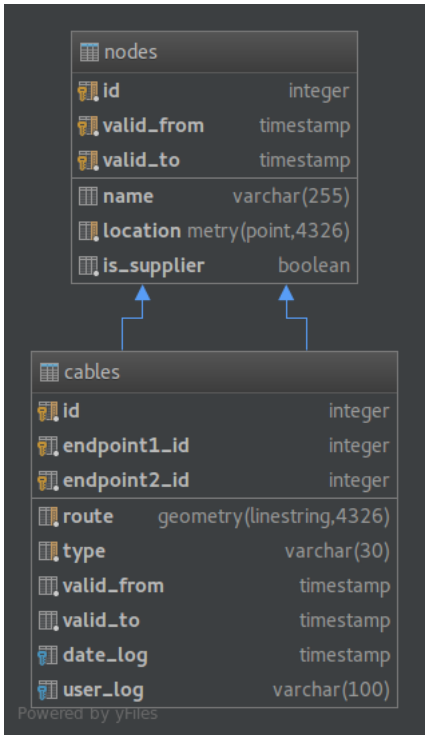


Figura 3.3: Modelo cable y nodo

El trio id, valid_from y valid_to son la llave primaria del nodo. Además, se posee un dato geométrico para representar su localización. Esta está representada con un punto usando EPSG:4326 como sistema de referencia. Cabe destacar que solo puede haber un único nodo en un punto especificado.

La id de los nodos creados sigue una secuencia de la base de datos, lo que asegura que ningun nodo compartirá id con otro.

Podemos notar que los nodos no guardan información acerca de los cables, si no que son los cables quienes identifican a dos nodos distintos como llaves foráneas.

Los cables, al igual que los nodos, obtienen su id de una secuencia. Ambos también comparten un rango de validez. Este rango nos permite saber si el objeto estaba activo en un punto específico en el tiempo. El cable guarda una ruta especificada como un LineString sobre EPSG:4326, las id de los nodos a su inicio y final, el tipo de cable que es (entre cable y fibra óptica) e información sobre la última vez que este fue manipulado.

Tanto los nodos como los cables tienen sus geometrías indexadas para poder acelerar el tiempo de las consultas que se hagan sobre ellas. Una consulta preguntando por los nodos cercanos a un punto en específico tomó alrededor de 0.8 segundos, mientras que luego de indexados los datos, la query demoraba 0.05, una mejora del 93 %.

### 3.3.3. Divisiones administrativas

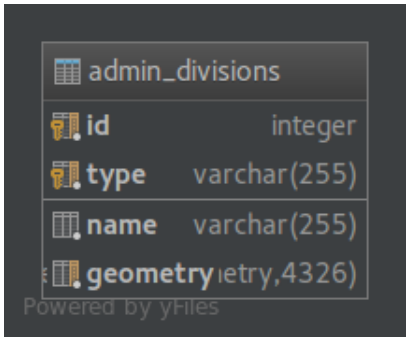


Figura 3.4: Modelo de divisiones administrativas

Las divisiones administrativas son los niveles de agregación en que se separará al país. Cada área está definida por una geometría, que acepta conjuntos de geometrías. Estas geometrías fueron obtenidas del sitio de la Biblioteca del Congreso Nacional de Chile[5]. Hay 3 tipos de división administrativa: Región, Provincia y Comuna, con las regiones estando compuestas de provincias, y las provincias compuestas de comunas.

Intuitivamente se puede pensar que la solución óptima tendría que venir con un modelo normalizado. Se optó por mantener todas las divisiones en una misma tabla, y agregar una columna extra que marca su tipo. La primera razón detrás de esto es que las búsquedas que se hacen en el sistema incluyen queries con datos espaciales, que requieren hacer productos cruz y buscar datos dentro de subdivisiones. Estas queries se ocupan frecuentemente, y hay un sobrecosto innecesario cuando se necesita que la aplicación de una buena experiencia de usuario. La segunda razón para mantener todas las geometrías en una sola tabla, es que hacen las queries mas fáciles de leer y de entender.

### 3.3.4. Historial del grafo

Como se explica en la sección 4.1.2, se usará una tabla de historia motivada por SCD tipo 4. Podemos ver en la figura 3.5, la tabla cables_history tiene los mismos campos que la tabla original cables, además datos nuevos. Estos corresponden al nombre de acción que gatilló el guardar la fila en la tabla de historial, el timestamp en que se gatilló la acción y el usuario que hizo la acción.

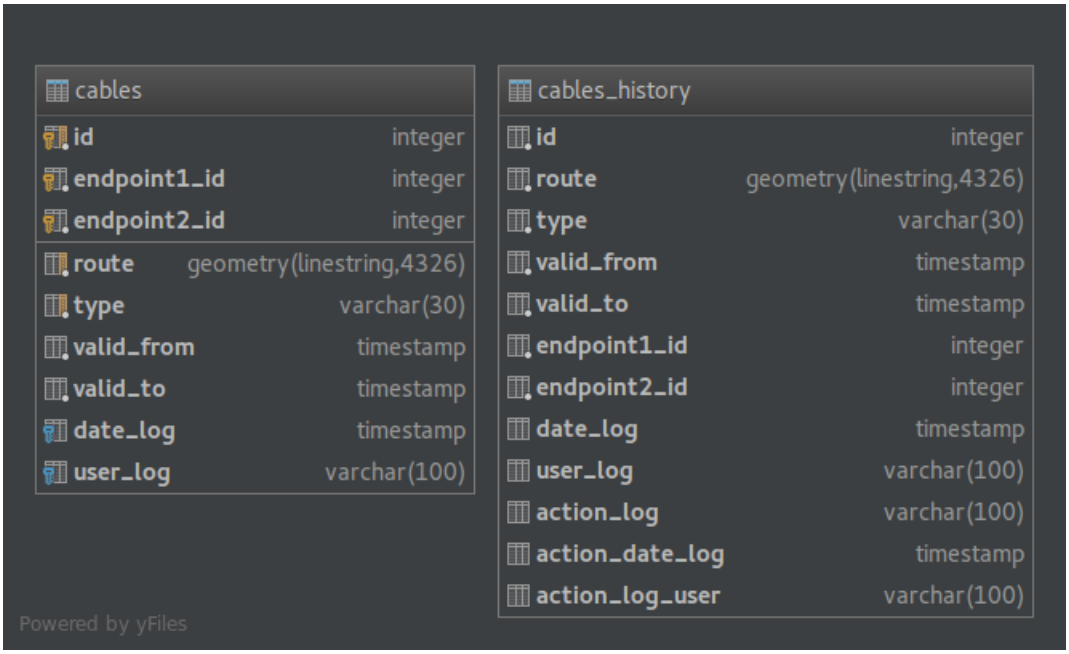


Figura 3.5: Historial de cable

De forma similar, la tabla de historial de nodos 3.6 guarda la información de los nodos cuando estaban vivos, el usuario que provoca la acción y el timestamp en que sucede. No se guarda cual fue la acción provocada, dado que esta tabla no es accedida diréctamente por los usuarios ni por el controlador.

La tabla de acciones tiene información acerca de cuando fue hecha la acción, que acción en particular fue, que usuario la provocó y que cables fueron afectados.

## 3.4. Front-End

En esta sección se presentará incrementalmente los features y tecnologías del Front-End.

### 3.4.1. Menú de navegación

En el menú de navegación se pueden apreciar casos básicos de las tecnologías usadas en desarrollo web.

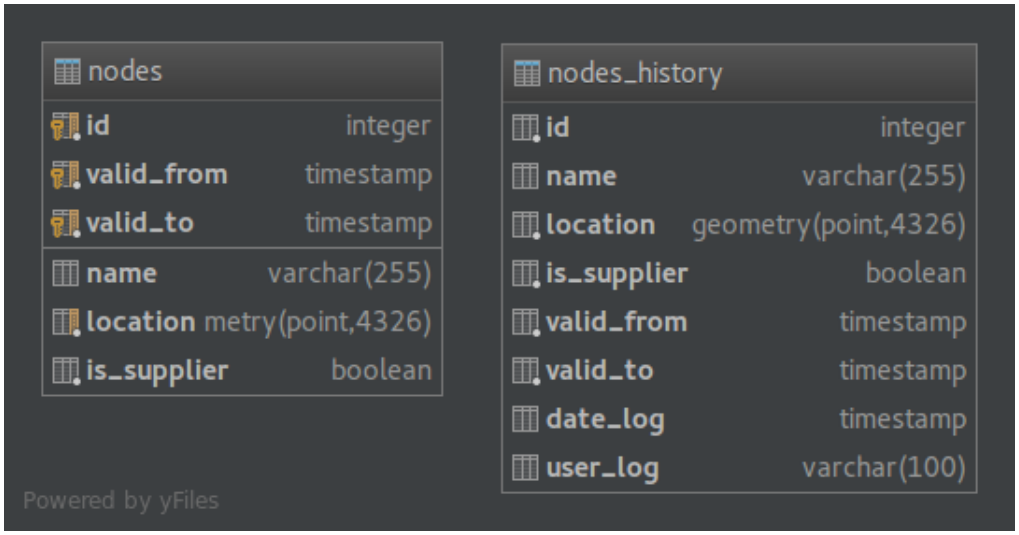


Figura 3.6: Historial de nodo

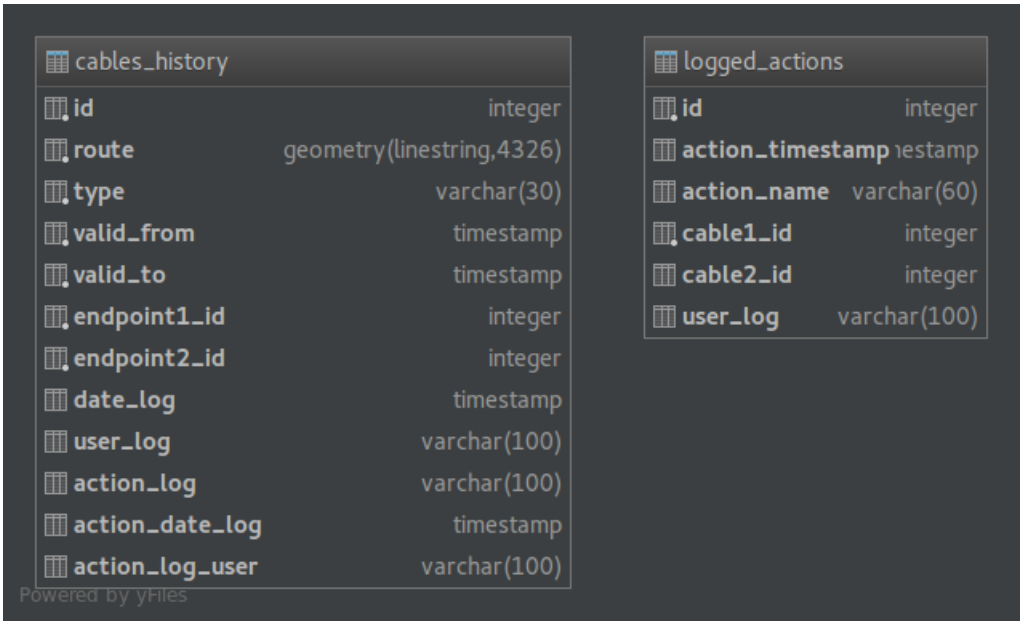


Figura 3.7: Historial de acciones

```
1 <div id="mySidenav" class="sidenav"></div>
2 <div class="hamburger">
3     <button title="Menu de opciones" onclick="openNav();" type="
        button"
4             class="btn btn-success btn-circle btn-lg"><i class="
                glyphicon glyphicon-menu-hamburger"></i></button>
5 </div>
```

Usando HTML se define la estructura de la barra de navegación y el botón hamburguesa 3.8 para abrirlo. HTML[4] es un lenguaje de markup, donde no se puede hacer programación.

[4]Hypertext Markup Language

Funciona como la estructura de la vista con la que el usuario deberá interactuar. Podemos ver en la linea 3 que el botón hamburguesa tiene un atributo onclick. Este llamará a la función openNav, previamente definida en JavaScript, quien se encarga de abrir la barra.

```
1 function openNav() {
2     $("#mySidenav").width("250px");
3 }
```

JavaScript es el lenguaje de programación que hace que las páginas web puedan ser interactivas, y posee múltiples librerías, como es jQuery. En la linea 2, jQuery selecciona el elemento con id mySideNav, y le da un ancho de 250 pixeles. De esta forma se abre el menú de navegación.

```
1 .sidenav {
2     width: 0; /* 0 width - change this with JavaScript */
3     position: fixed; /* Stay in place */
4     z-index: 1001; /* Stay on top */
5     transition: 0.5s; /* 0.5 second transition effect to slide in
          the sidenav */
6 }
```

Como vimos, el menú de navegación está hecho en HTML. Para darle estilo a los elementos, se hace uso de CSS[5]. En este ejemplo, se usa .sidenav para referirse a todos los elementos con clase sidenav, y se les define el estilo. El menú originalmente no se puede ver. Esto es porque tiene 0 de ancho. Propiedades interesantes que provee CSS son por ejemplo, el poder dejar la posición de un elemento fijo, la duración de las transiciones (cuando cambia el ancho del menú en este ejemplo concreto) y si estará por debajo o por sobre otros elementos del documento, definido por z-index.

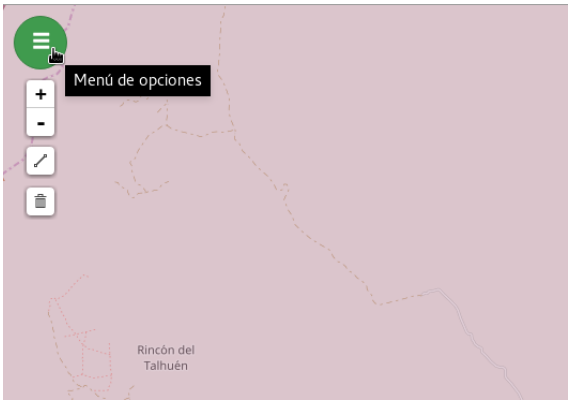


Figura 3.8: Botón hamburguesa.

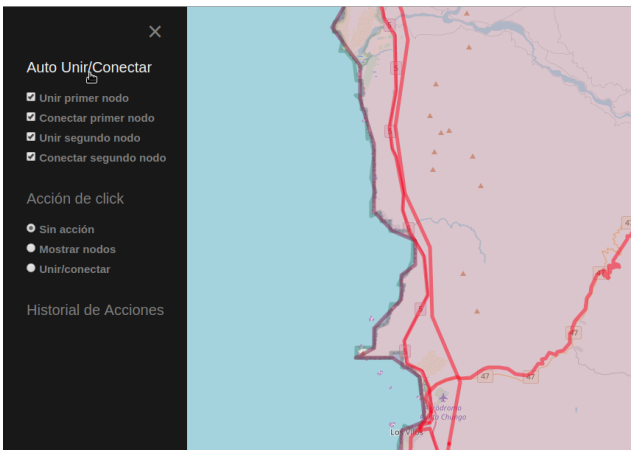


Figura 3.9: Menu de opciones

[5]Cascading Style Sheets

### 3.4.2. Búsqueda de área

Para trabajar sobre el mapa y poder visualizar los cables, hay que elegir el área en que se quiere trabajar. Primero se selecciona el nivel administrativo sobre el que se va a trabajar, siendo las opciones Provincia o Comuna, y luego se puede proceder a elegir un área en particular. La primera opción para elegirla, es usar la barra de búsqueda de la figura 6.5. En esta barra se muestra un listado de las áreas.

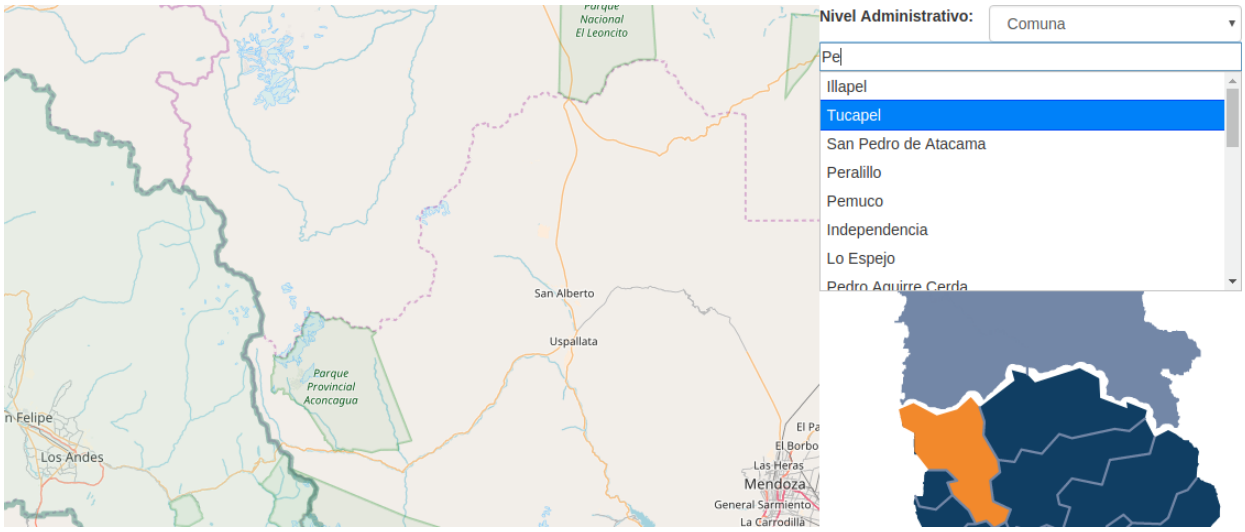


Figura 3.10: Barra de búsqueda

```
<div id="adminLevelSelector">
   <label id="admin-label">Nivel Administrativo:</label>
   <select id="adminLevel" name="adminLevel" class="form-control">
       <option value="Provincia">Provincia</option>
       <option value="Comuna">Comuna</option>
   </select>
</div>
<input type="text" id="tags" placeholder="Insertar comuna/
   provincia a buscar">
```

La barra de búsqueda inicialmente solo tiene un texto de placeholder. La funcion de autocompletar se maneja usando un widget de jQuery, autocomplete.

```
$(function () {
    $("#tags").autocomplete({
        source: function (request, response) {
            var administrative_level = $("#adminLevel
               option:selected").val();
            $.getJSON('/searchSuggestion', {administrative_level :
                administrative_level, term: request.term},
               response);
        },
        delay: 500
    }).on( "autocompleteselect", function( event, ui ) {
        dispatch.searchZoom(ui.item.value);
```

```
10      });
11  });
```

Cuando el usuario tipea sobre la barra, autocomplete va a buscar sobre una fuente. Esta puede ser un arreglo de string, arreglo de objetos en formato JSON[6] o una función. En este caso se usa una función que usando jQuery, va a buscar un JSON, con el que llena las opciones para elegir. En la linea 5 se puede ver como se usa getJSON, un wrapper de un request GET, pasando como parámetros extra el nivel administrativo que se quiere usar y lo que se lleva escrito.

```
1  @app.route('/searchSuggestion', methods=['GET'])
2  def area_suggestion():
3      partial = request.args['term']
4      partial = universalize_string(partial)
5      searching_for = request.args['administrative_level']
6      results = []
7      if searching_for == "Comuna":
8          for values in commune_codes.values():
9              if partial in universalize_string(values):
10                 results.append(values)
11     else:
12         for values in prov_codes.values():
13             if partial in universalize_string(values):
14                 results.append(values)
15     return json.dumps(results)
```

En el backend, se maneja la solicitud de los datos, retornandolos como un objeto JSON para devolverlo a autocomplete.

Cuando el usuario selecciona la opción desde autocompletar, se envía un evento searchZoom hacia dispatch, quien dentro del minimapa de polígonos se encargará de hacer el resto de la selección.

### 3.4.3. Minimapa de poligonos

Para representar el mapa de las regiones, provincias y comunas de Chile, se toman los polígonos conseguidos de la Biblioteca del Congreso Nacional[5] y luego se usa la librería d3[1] para mostrarlos. Los polígonos están en formato TopoJSON, una extensión de GeoJSON, un estandar diseñado para representar features[7] geográficas simples como modelos de geometrías 2-dimensionales usados por sistemas de geoinformación.

En el mapa de polígonos se definen dos tipos de divisiones, primaria y secundaria. La división primaria sería la de regiones, mientras que secundarias serían provincia o comuna, dependiendo de la elegida previamente.

[6] JavaScript Object Notation

[7] Feature se usará para referirse a elementos tanto en d3 como en leaflet

```
1  PolygonMap.prototype.loadMap = function () {
2      var mapReference = this;
3      d3.json(this.primaryDivisionURL,
4          function (country) {
5              var primaryDivision = topojson.feature(country,
                   country.objects.objects);
6              mapReference.path_object.projection(
                   mapReference.computeProjection(primaryDivision));
7              mapReference.g.append("g")
8                  .attr("id", "primaryDivision")
9                  .selectAll("path")
10                 .data(primaryDivision.features)
11                 .enter().append("path")
12                 .attr("d", mapReference.path_object)
13                 .attr("id", function (d) {
14                     return d.id;
15                 })
16                 .on("click", onPrimaryDivisionClick)
17             d3.selectAll("#primaryDivision").selectAll("path")
18                 .on("mouseover", onPrimaryDivisionMouseover);
19         ...
20         })}
```

Como fue mencionado, se usa la librería d3 para poder enlazar datos al código html. En la linea 3 se hace uso por primera vez de la librería. Se entrega una URL y una función callback. Se hace un request GET en la URL, y luego la respuesta se entrega como argumento a la función callback que luego es ejecutada. En este caso, se hace uso de una función anónima como callback. Se hace uso de las principales características de d3 desde la linea 9. Se seleccionan todos los elementos de tipo path, y los enlaza con los datos obtenidos previamente. Luego de esto en la linea 11 se usa una *selección enter*, donde se especifica que hacer si hay mas datos que elementos de tipo path. Aquí se utiliza para crear elementos path y poblarlos con la información ingresada en data, como son la id y la geometría que lo representa.

También podemos ver que se agregan listeners a los elementos para que respondan a eventos mouseover y click.

```
1  function onPrimaryDivisionClick(d) {
2      removeSecondaryAreaHighlight();
3      if (d && mapReference.centeredFeature !== d) {
4          ...
5          mapReference.zoom = scale /
               mapReference.path_object.projection().scale();
6          mapReference.centeredFeature = d;
7          // Notify observers
8          for (var i = 0; i <
               mapReference.primaryDivisionSelectedObververs.length; i
               ++) {
9              mapReference.primaryDivisionSelectedObververs[i](d);
10         }
11     }
```

```
12      else {
13          pos_x = mapReference.width / 2;
14          pos_y = mapReference.height / 2;
15          mapReference.zoom = 1;
16          mapReference.centeredFeature = null;
17      }
18      mapReference.g.selectAll("path")
19          .classed("active", mapReference.centeredFeature &&
                function (d) {
20                  return d === mapReference.centeredFeature;
21              });
22      mapReference.g.selectAll("#secondaryDivision").selectAll("path
            ")
23          .classed("active", maface partspReference.centeredFeature
                &&
24              function (d) {
25                  return mapReference.centeredFeature.id ==
                        d.properties["region_id"];
26              });
27      mapReference.g.transition()
28          .duration(750)
29          .attr("transform", "translate(" + mapReference.width / 2 +
                "," + mapReference.height / 2 + ")scale(" +
30              mapReference.zoom + ")translate(" + -pos_x + "," +
                    -pos_y + ")")
31          .style("stroke-width", 1.5 / mapReference.zoom + "px");
32  }
```

En caso de hacerse click sobre una región, se checkea si esta era la área centrada previamente y si es válida. En caso de serlo, se centra el mapa sobre la feature, y se avisa a los observadores. En caso de ser un área inválida (el background del mapa) se resetea el zoom y la posición. Desde la linea 18 podemos ver como se hace uso de los selectores de d3 para activar la feature que debe ser centrada junto a las areas secundarias. Luego se hace la animación del zoom del mapa de polígonos. El resultado de esto se puede ver en 3.11, 3.12 y 3.13

De manera similar, al pasar el mapa por encima de las áreas se llama a las funciones que hacen el destacado de las áreas en el minimapa, como en 3.12 y 3.13.

Los polígonos y funcionalidades para las áreas secundarias son creadas de forma muy similar. Ambas tienen un observador que dibuja el contorno del área en el mapa del terreno. La única importante diferencia entre ellas es que al hacerse click sobre un área secundaria, uno de los observadores se comunica con el mapa del terreno, quien se encarga de cargar el grafo que representa la red de internet chilena sobre el área seleccionada.

Figura 3.11: Minimapa de terreno, nivel región.

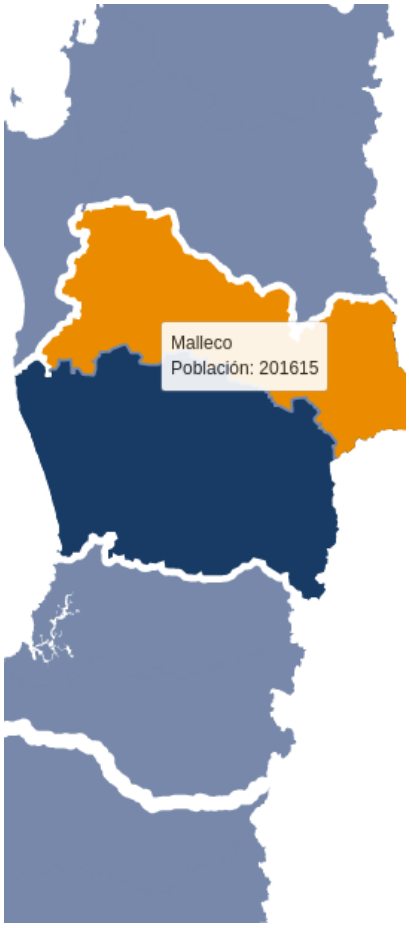


Figura 3.12: Minimapa de terreno, nivel provincia.

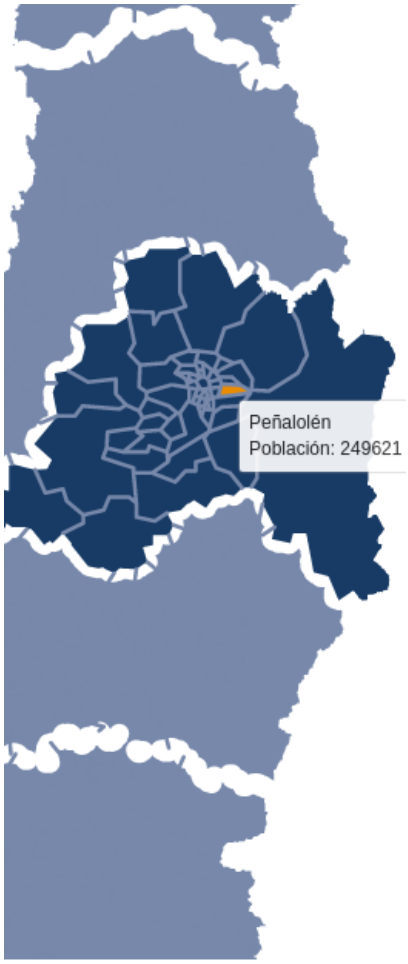


Figura 3.13: Minimapa de terreno, nivel comuna.

### 3.4.4. Mapa del terreno

El mapa del terreno funciona usando Leaflet para dibujar sobre el mapa. Se usa L.geoJson para recibir una lista de features y una función que les asigna el estilo. En este caso, no se entregan las features pero si la función. En genral, lo que se hace es asignar los handlers de los eventos de interacción con el mouse.

```
1 var TerrainMap = function (containerId) {
2     ...
3     mapReference = this;
4     this.geoJsonLayer = L.geoJson(null, function (feature, layer)
        {
5         onEachFeature: function (feature, layer) {
6             if (!(feature.properties.hasOwnProperty("region_name")
                  ||
7             if (feature.properties.hasOwnProperty("feat_id")) {
8                 layer.on({
9                     "click": function (event) {
```

```
10                          if (!previewRollbackActive && clickAction
                               == 'showNodes') {
11                              $.post("/get_nodes_from_cable",
12                                  {cable_id: feature.properties["
                                       feat_id"]},
13                                  function (data) {
14                                      self.tempMarkers.eachLayer(
15                                          function (layer) {
16                                              self.drawnItems.removeLayer
                                                   (layer);
17                                          }
18                                      );
19                                      self.geoJsonLayer.addData(
                                           data.node1);
20                                      self.geoJsonLayer.addData(
                                           data.node2);
21                                  }
22                      });}},
23                      "mouseover": function (event) {
24                          layer.setStyle(HighlightLineStyle);
25                          ...
26              }})}},
27          style: function (feature) {...}
28      }).addTo(this.map);
29  ...}
```

Por ejemplo, en la linea 11 podemos ver que al haber un click sobre un cable, se pregunta al servidor cuales son los nodos a los que el cable está conectado, y los agrega al mapa. Dado que ya está hecha la función para dar estilo y funcionalidad a los cables, lo que falta es agregar los datos de los cables al mapa. Como se mencionó antes, esto lo hacen los observadores registrados al minimapa.

Estos observadores hacen 3 trabajos distintos. El primero, es delinear áreas en el mapa, de región para la división primaria, y de provincia o comuna en la división secundaria. También se encargan de hacer zoom a la feature seleccionada, y lo más importante, es que cargan los cables sobre el mapa.

```
1  var getLines = function (feature) {
2      terrainMap.removeGeoJsonFeatures(function (f) {
3          return f.properties.hasOwnProperty("feat_id");
4      })
5      activeArea = feature;
6      commune_id = feature.properties.id;
7          url = getLinesURL + commune_id + "/lineas";
8          d3.json(url, function (error, feat) {
9              terrainMap.addGeoJsonFeature(feat);
10         });
11     }
12 }
```

La función getLines se llama cuando una división secundaria es clickeada. Esta se preocupa de remover todos los cables previamente existentes en el mapa, y a través de d3 se agregan una a una las features al mapa del terreno.

El otro punto importante en el mapa del terreno, es el plugin Leaflet.Draw para poder hacer la manipulación de los cables. Gracias a este plugin, se pueden crear y eliminar cables del grafo de forma interactiva.

```
1  var controlOptions = {
2          edit: {
3              featureGroup: self.drawnItems,
4              remove: true
5          },draw: {
6              polyline: { shapeOptions: {color: '#ff0000'
7  }}}}
8  var drawControl = new L.Control.Draw(controlOptions);
9  this.map.addControl(drawControl);
10 this.map.on('draw:created', function (e) {
11     self.drawnItems.addLayer(layer);
12     $.post("/create_cable",
13         {layer: JSON.stringify(layer.toGeoJSON())},
14         function (data) {
15             self.drawnItems.removeLayer(layer);
16             terrainMap.addGeoJsonFeature(layer.toGeoJSON());
17             ...
18 });
```

En el control se puede definir que tipo de edición se quiere aceptar, el estilo de los elementos creados por este, y también define las funciones que se gatillan cuando se crean y borran cables. En la linea 12 podemos ver que al ser creada la linea, se agrega temporalmente al mapa (para efectos de lo que ve el usuario), y cuando se recibe la función de éxito de cable creado esta se borra, y se agrega apropiadamente al mapa del terreno, donde se les dará estilo y funcionalidad a las features.

### 3.4.5. Historial de Acciones

El historial de acciones se puede acceder a través del menú de navegación. Este muestra el historial de las acciones que se han hecho sobre el grafo. Al seleccionar una de ellas se tiene 2 opciones importantes. Una es hacer un preview de como se veía el mapa en el timestamp seleccionado. La otra, es aprobar un rollback. Adicionalmente, se puede navegar por las múltiples páginas del historial de acciones.

## 3.5. Lógica del Controlador

### 3.5.1. Cables como features

Vimos en la sección de mapa del terreno que para dibujar el grafo en el mapa, primero se hace un request al servidor para conseguir los cables.

```
def get_lines_by_commune(commune_id):
    commune_name = get_commune_name_by_id(commune_id)
    if commune_name is None:
        resp = jsonify({'status': 404, 'message': 'commune not found'})
        resp.status_code = 404
        return resp
    sql = text("SELECT row_to_json(collection) FROM
       (SELECT 'FeatureCollection' As type,
         array_to_json(array_agg(feat)) As features
       FROM (SELECT
          'Feature' As type,
          ST_AsGeoJSON(route)::json AS geometry,
          row_to_json(SELECT l FROM
            (SELECT id AS feat_id, type AS feat_type) As l) As properties
          FROM cables as feat
          WHERE ST_Intersects(route, (SELECT geometry FROM
          admin_divisions WHERE name='" + commune_name + "')))
             as feat) As collection")
    result = db.engine.execute(sql)
    json = {}
    for row in result:
        json = row[0]
    return jsonify(json)
```

Para esto, se toman los cables que intersecten con el área secundaria que se pide, se transforma la información guardada como geometrías a geoJSON y se empaqueta todo como un featureCollection listo para ser retornado al front-end. Sería preferible evitar el uso directo de consultas hacia el servidor, sin pasar antes por el modelo provisto por Flask. Se justifica el no hacerlo en este caso dado que la consulta no modifica la base de datos, manteniendo su integridad intacta, y dado que simplifica la creación del FeatureCollection que el front-end espera.

### 3.5.2. Manipulación de los cables

Vimos anteriormente que un cable era creado gráficamente con Leaflet.Draw, y que para persistirlo se hacía un post hacia el servidor. La url es capturada, y luego se ejecuta el código que las guarda en la base de datos.

```
def create_new_cable(coords, time_now=datetime.now(), given_session=session):
```

```
    initial_node = new_node(coords[0],
        time_now=time_now, given_session=given_session)
    final_node = new_node(coords[-1],
        time_now=time_now, given_session=given_session)
    cable = new_cable(coords, initial_node,
        final_node, time_now=time_now, given_session=given_session)
    ...
    return json.dumps({'success': True,
        'new_id': cable.id, 'cable_endpoints':
        "["+str(cable.endpoint1_id)+","+str(cable.endpoint2_id)+"]",
        'near_nodes': near_nodes, 'layerinfo': layerinfo})
```

Los nodos son creados primero para satisfacer las llaves foráneas del cable, luego este es creado y se retorna un json al front-end. Podemos notar que en las funciones que hacen cambios en la base de datos, se entrega como parametro una sesión. Esta sesión es usada para hacer testing de la API.

```
def cable_connect(id_cable, cable_endpoint, join_node_id,
    time_now=datetime.now(), given_session=session):
    cable = given_session.query(Cables)
        .filter(Cables.id == id_cable).first()
    ...
    update_cable(cable.id, json.dumps(cable_json),
        "UPDATE/CONNECT", time_now=time_now, given_session=given_session)
    log_action(action_name="UPDATE/CONNECT", cable1_id=cable.id,
                action_timestamp=time_now, given_session=given_session)
    delete_node_if_not_connected(cable_endpoint_node.id,
        time_now=time_now, given_session=given_session)
    return json.dumps({'cableid': id_cable,
        'newcable': get_line_as_feature(id_cable, given_session),
        'action': 'connect'})
```

Dentro de la lógica para conectar o unir cables, se modifica el nodo en particular que se quiere cambiar (sea borrar si se unen dos cables, o reemplazar si se conecta a uno ya existente) y se ajusta el linestring que representa al cable para que coincida con el cambio.

Cada vez que se hace una acción que modifique la base de datos, esta es guardada con log_action, y el valor anterior del cable es también guardado en la tabla de historial. Se modifica el valor del timestamp del rango de validez de los cables (dentro de update_cable), para reflejar que en este momento es cuando deja de estar activo.

### 3.5.3. Tablas de Historial

El trabajo mas importante para las tablas de historial es el hacerse cargo de los previews de saltos en el tiempo (como se veía el grafo en un instante en el tiempo) y de los rollbacks.

```
def database_rollback_to(rollback_timestamp, given_session=session):
```

```
    to_be_revived_nodes = given_session.query(NodesHistory) \
        .filter(NodesHistory.valid_from <= rollback_timestamp,
                NodesHistory.valid_to >= rollback_timestamp).all()
    ...
    time_now = datetime.now()
    rollback_log = "ROLLBACK: deleted " + len(to_be_deleted_cables)
        + ", revived " + len(to_be_revived_cables)
    log_action(action_name=rollback_log, cable1_id=None, cable2_id=None,
        action_timestamp=time_now, given_session=given_session)
    for cable in to_be_deleted_cables:
        delete_cable(cable.id, delete_reason="ROLLBACK_DELETE",
            log_delete_action=False, time_now=time_now,
            given_session=given_session)
    ...
    rollback_data = dict()
    rollback_data["deleted_cables_amount"] = len(to_be_deleted_cables)
    ...
    return json.dumps(rollback_data)
```

Se obtienen todos los datos que se vean afectados: los cables y nodos que deben revivir, como también los nodos y cables que deberán ser eliminados. Podemos ver en la linea 9 que se hace un log con información de alto nivel acerca del rollback. Dado que un rollback afecta múltiples cables, se pasa el parametro log_delete_action=False para que no se guarde una acción por cable afectado. Los elementos que cambian son guardados en su tabla de historial correspondiente, independientemente de si la acción va a ser guardada o no. Esto solo evita que la tabla de log de acciones se llene con múltiples filas que representan la misma acción en el mismo timestamp.

### 3.5.4. Log de acciones y Paginación

Dado que los rollbacks y previews a estos requieren un instante en el tiempo al que viajar, es en el historial de acciones que se guarda esta información. Vimos previamente que las templates usadas por flask para crear los html que el usuario recibirá. Estas templates pueden también usar argumentos extra para llenar el contenido. Esto es utilizar en particular para llenar la tabla de historial de acciones que verá el usuario, para seleccionar cuando hacer un rollback.

```
<table class="table table-bordered table-hover">
...
{% for action in actions.items %}
    <tr><td>
        <div class="radio-button-text">
            <label for="{{ loop.index }}">{{ action.action_name }}
               </label>
        </div>
    </td>
    ...
```

```
10      </tr>
11  {% endfor %}
12  </table>
```

En el caso de la tabla de acciones, se entrega el argumento actions, quien es usado para llenar la tabla y los botones que verá el usuario. Gracias al uso de SQLAlchemy se pueden recibir los datos paginados, para así no tener que transferir todos los datos al mismo tiempo.

## 3.6. Tests unitarios

Bugs en el software son problemas recurrentes en el ciclo de vida de un sistema. Una buena forma para mitigarlos es hacer tests sobre el código. De esta forma, luego de hacer modificaciones sobre el software se pueden correr los tests para asegurar que lo testeado siga funcionando correctamente.

Se opta por hacer tests unitarios sobre la API del software.

Al iniciar en el proyecto se buscan extensiones para poder hacer testing unitario del sistema. La idea es hacer uso de la API, y que luego de correr los tests la base de datos volviera al estado de antes de correr el test. No se logra encontrar una extensión que diréctamente provea esta funcionalidad, por lo que esta se emula creando y luego destruyendo una base de datos solo para testing que es creada a partir de un template.

```
1     def setUp(self):
2         app.config.from_object('config.SuperUserConfig')
3         self.engine = create_engine(app.config['DATABASE_URI'])
4         self.conn.execute("CREATE DATABASE resilienciatest WITH TEMPLATE
5             resilienciatemplate OWNER postgres")
6         app.config.from_object('config.TestingConfig')
7         self.db = create_engine(app.config['DATABASE_URI'])
8         sess = sessionmaker()
9         sess.configure(bind=self.db)
10        self.session = sess()
11
12    def tearDown(self):
13        self.session.close()
14        self.conn.execute("commit")
15        self.conn.execute("SELECT *, pg_terminate_backend(pid) FROM
16            pg_stat_activity WHERE datname='resilienciatest'")
17        self.conn.execute("DROP DATABASE resilienciatest")
18        self.conn.close()
```

Antes de iniciar cada test se crea esta base de datos y se crea una sesión que conecta a ella, y al terminar de correr, se cierra a la fuerza las conexiones que estén usándola para luego botarla. Actualmente esto produce un overhead de alrededor de 1.5 segundos por test.

Se hace testing sobre las partes críticas de la API, como son el creado de cables, sus

modificaciones, borrado, log de acciones y rollback, entre otros.

```
1  def test_createCable(self):
2      new_cable_route = [[-72.33498, -40.60352], [-72.2450, -40.560240]]
3      current_cable_amount = self.session.query(Cables).count()
4      location = from_shape(LineString(new_cable_route), srid=4326)
5      non_existant_cable = self.session.query(Cables)
6          .filter(Cables.route == location).first()
7      self.assertIsNone(non_existant_cable)
8
9      create_new_cable(new_cable_route, self.session)
10     newly_created_cable = self.session.query(Cables)
11         .filter(Cables.route == location).first()
12     self.assertIsNotNone(newly_created_cable)
13     self.assertEqual(self.session.query(Cables).count(),
14         current_cable_amount+1)
```

Se puede ver en la linea 9 que se pasa la sesión de testing a la API para que no modifique la base de datos de desarrollo.

# Capítulo 4

# Evaluación de solución

Este capítulo se divide en dos secciones. La primera refiere al rendimiento de la solución en términos técnicos, y la segunda refiere a la usabilidad de esta.

## 4.1. Rendimiento técnico

### 4.1.1. Servidor

Se puede evaluar el rendimiento técnico de la solución de distintas formas. La primera, es medir el cambio en el número de elementos en la base de datos al hacer uso del sistema. En la tabla 4.1, se aprecia el cambio con distintas acciones.

Tabla 4.1: Número de filas en la tabla correspondiente

| Acción | Cables | Cables Historial | Nodos | Nodos Historial | Log Acciones |
|---|---|---|---|---|---|
| 1. Sin accion | 1738 | 206 | 1925 | 194 | 319 |
| 2.Crear 10 cables | 1748 | 206 | 1945 | 194 | 329 |
| 3.Crear 10, conectar 1 | 1758 | 216 | 1955 | 204 | 349 |
| 4.Crear 10, conectar 2 | 1768 | 236 | 1955 | 224 | 379 |
| 5.Borrar 10 cables | 1758 | 246 | 1955 | 224 | 389 |
| 6.Crear 10, autojoin 2 | 1748 | 286 | 1935 | 264 | 439 |
| 7.Rollback acción 1 | 1738 | 297 | 1925 | 275 | 440 |
| 8.Rollback acción 6 | 1748 | 298 | 1935 | 276 | 442 |
| 9.Rollback completo | 1 | 2045 | 6 | 2205 | 443 |
| 10.Rollback acción 6 | 1748 | 2045 | 1935 | 2205 | 444 |

Podemos ver que todas las acciones menos rollback crean $\mathcal{O}(1)$ filas nuevas en la base de datos. Los valores difieren dependiendo de la acción. Al crear 10 cables, se crean 20 nodos (2 por cable) y se guardan 10 acciones de creado, en cambio al crear 10 usando autojoin en ambas puntas, para cada acción se crea 1 cable, se crean 2 nodos, se hacen 2 join, que

incluyen cada uno 1 delete intermedio. Rollback en cambio, crea $\prime(n)$ filas nuevas en función del tamaño total de cables en la base de datos.

Podemos analizar también el gasto en espacio de la base de datos, para ver si su uso es sostenible en el tiempo.

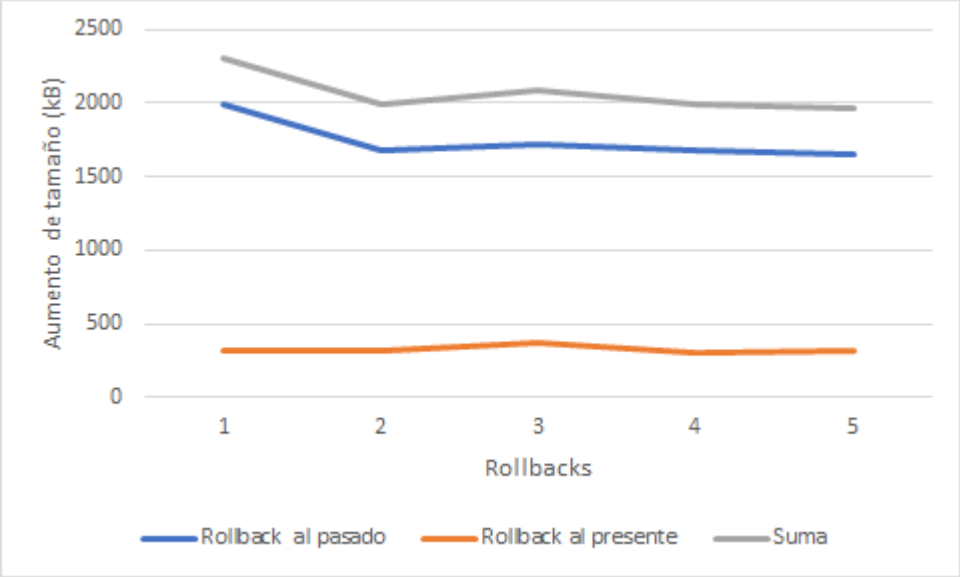


Figura 4.1: Aumento del tamaño en la base de datos. Los rollbacks al pasado son al instante en que existía solo un cable, y al presente son revertir al estado original.

Podemos ver en la figura 4.1 el comportamiento de la base de datos. Se hicieron 10 rollbacks consecutivos. Alternándose, uno fue al instante en que solo existía un cable, y el siguiente al estado original. El aumento del tamaño de la base de datos es diréctamente proporcional al numero total de cables que deben ser movidos al historial luego de un rollback. Considerando que el promedio de este cambio es de 2079 kB, y también el uso objetivo del software creado, podemos asumir que los rollbacks al inicio no serán comunes, y que el uso de espacio es negligible.

Podemos notar que el espacio que usa la base de datos no es siempre la misma, a pesar de que el número de cables sea consistente con las operaciones hechas. Dado que el número de cables es consistente y no hay un aumento notorio del tamaño de la base de datos no se investiga más al respecto.

Cabe destacar que el uso de previews de rollback no hace uso intensivo de la base de datos, dado que no debe escribir sobre ella si no solo recolectar la información de las tablas.

### 4.1.2. Usuario

Dado que el sistema será usado en distintos lugares de Chile, se asume que las conexiones a internet de los usuarios no serán siempre de calidad. Es por esto que se hace énfasis en que haya una baja transferencia de datos entre el usuario y el servidor.

La operación que más transferencia de datos, es la conseguir los cables en un área determinada. Se usaron 2 dataset para esta sección. El primer dataset, son los datos reales del proyecto. El segundo dataset, son los datos de prueba que se usaron durante la primera parte del desarrollo del sistema. El segundo dataset es considerablemente mas grande que el original.

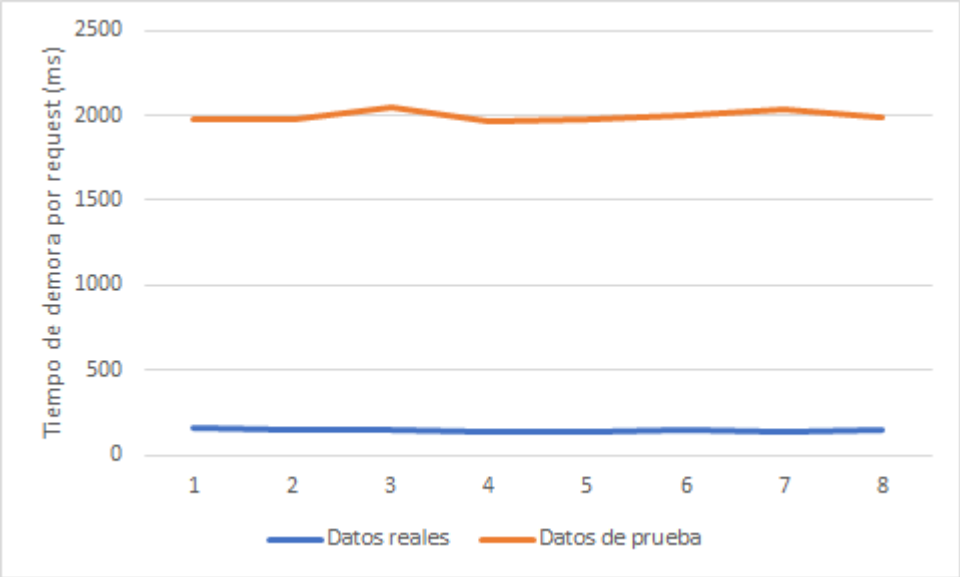


Figura 4.2: Tiempo de demora por request de getLines en Santiago.

Se usa Santiago como área para hacer las pruebas, dado que es la que contiene más cables. Podemos ver en la figura 4.2 el tiempo que demora hacer las request en los dos dataset. El dataset falso contiene mas de 4000 cables, mientras que El dataset real tiene 72 cables en Santiago, mientras que el dataset falso contiene mas de 4000, casi 60 veces el tamaño real. Aún así, el promedio por request fue menor a 2.1 segundos. El tamaño de los datos transferidos por request es de 329 kB para el dataset real y 1.9MB para el dataset falso.

Para ver si el uso del software podría escalar su uso a países mas grandes, podemos hacer una comparación con China, el país con más población del mundo. En Santiago hay cerca de 5 millones de habitantes, mientras que en Shanghai, la ciudad con mayor población del mundo tiene 25 millones. La población total en Chile es menor a los 19 millones de habitantes, mientras que en China supera los 1400 millones. Shanghai tiene 5 veces la población de Santiago, y China tiene alrededor de 80 veces la población de Chile. Haciendo la suposición de que la cantidad de cables crece proporcional a la cantidad de habitantes, se estima que el uso de datos y tiempo de espera de los request son suficientemente pequeños como para trabajar con ellos.

## 4.2. Usabilidad

A lo largo del proyecto se hizo múltiples pruebas de usuario. Por comodidad estas fueron hechas por 3 miembros de la familia del desarrollador. Estas fueron dirigidas para recibir

feedback acerca de como mejorar la experiencia de uso. El mayor resultado de las pruebas de usuario fue el desarrollo de la barra de navegación. Aquí se pudo agrupar las funciones del sistema, de forma que el usuario pudiera explorar de forma intuitiva la interfaz y se pudiera dar un uso correcto y satisfactorio.

Los usuarios manifiestan disgusto al tener que esperar las operaciones de rollback, pero con el tiempo establecido no se pudo hacer una solución satisfactoria para el problema.

Una solución para mitigar el efecto es hacer un indicador de que hay una operación en curso, para que el usuario sepa cuando se está esperando al servidor.

# Capítulo 5

# Conclusiones

En este capítulo se presentan conclusiones finales sobre el trabajo realizado en la memoria, así como propuestas de trabajo futuro y posibles extensiones a la memoria.

## 5.1. Conclusiones

Se cumplió satisfactoriamente con el objetivo general establecido al inicio de la memoria, habiendo implementado un sistema que apoya la creación, visualización, edición y mantención del grafo que representa la red física del internet chileno, cumpliéndose así los casos de uso y requisitos definidos previamente. Una guía de uso del sistema puede encontrarse en el Anexo A.

### Cumplimiento de objetivos específicos

Se considera que los objetivos específicos fijados al comienzo del desarrollo de la memoria fueron cumplidos. La herramienta nos permite visualizar el mapa gracias al uso del mapa del terreno que usa Leaflet, y se dio también soporte para hacer recepción de datos nuevos, tanto como la modificación de los ya existentes.

El uso del historial de información se considera un éxito, cumpliendo la función de hacer rollback manteniendo la integridad de los datos. El uso de la operación rollback toma un tiempo considerable, que provoca problemas de usabilidad para los usuarios. Si se desea continuar el desarrollo del sistema y aplicarlo para otros países, se deberá optimizar el back-end para disminuir estos tiempos.

La sugerencia de conexiones fue implementada, pero se considera necesario buscar mas formas para ayudar con el curado de los datos.

Dado que los datos reales usados en el software fueron recolectados manualmente, en

estos momentos múltiples partes del grafo están disconexas y no hay una forma simple de visualizarlo.

## 5.2. Trabajo Futuro

A continuación se presentan propuestas de trabajo futuro:

- Crear una opción para marcar las distintas componentes conexas del grafo.
- Optimizar el uso de la base de datos durante un rollback para evitar tiempos de espera excesivos.
- Crear indicadores de loading.
- Dar soporte en dispositivos móviles.
- Dar soporte para imágenes a lo largo de los cables[1].
- Extensión para Leaflet.Draw para conectar cables de forma interactiva.
- Implementar creado de múltiples cables de forma consecutiva.

[1]Para referenciar el lugar físico donde pasa el cable

# Capítulo 6

# Bibliografía


[1] D3, data driven documents. https://d3js.org/.

[2] Gephi, plataforma para visualización de grafos. https://gephi.org/.

[3] Leaflet draw, extensión para dibujar en leaflet. https://github.com/Leaflet/Leaflet.draw.

[4] Leaflet, librería para mapas interactivos. http://leafletjs.com/.

[5] Mapas poligonales de áreas administrativas en chile. https://www.bcn.cl/siit/mapas_vectoriales/index_html.

[6] Pkcs #5: Password-based cryptography specification version 2.0. https://www.ietf.org/rfc/rfc2898.txt.

[7] Postgis, objetos geográficos y espaciales para postgresql. http://postgis.net/.

[8] Slowly changing dimensions tipo 4. https://en.wikipedia.org/wiki/Slowly_changing_dimension#Type_4:_add_history_table.

# Anexo

## A. Manual de Uso

El sistema a nivel de usuario consiste de una única vista principal. Esta a su vez está compuesta por tres secciones principales. El mapa del terreno, el minimapa de regiones y menú de opciones.


Figura 6.1: Vista principal

## Minimapa de regiones

Para visualizar el estado del grafo en una región en específico, se selecciona un nivel administrativo entre Provincia y Comuna. Estos dictarán el área geográfica del mapa a usar. Luego se puede elegir un área en particular del mapa a representar. Usando el buscador de

áreas se puede buscar por texto el área a representar, mientras que usando el minimapa de regiones se podrá seleccionar interactivamente.

Figura 6.2: Minimapa de terreno, nivel región.

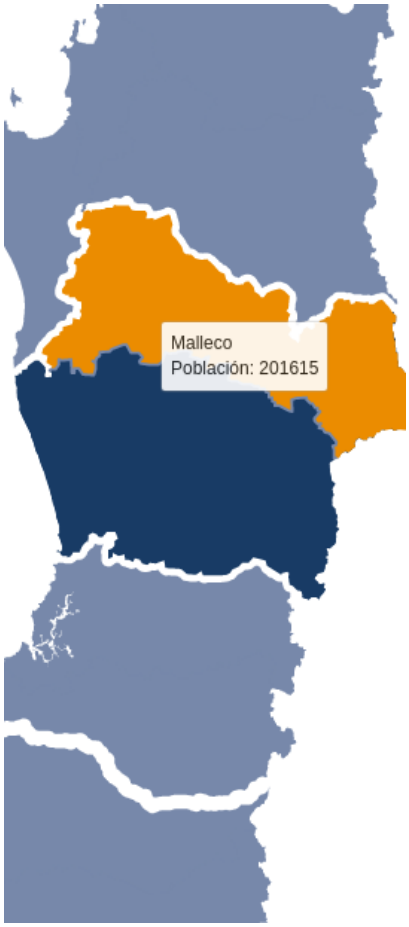


Figura 6.3: Minimapa de terreno, nivel provincia.

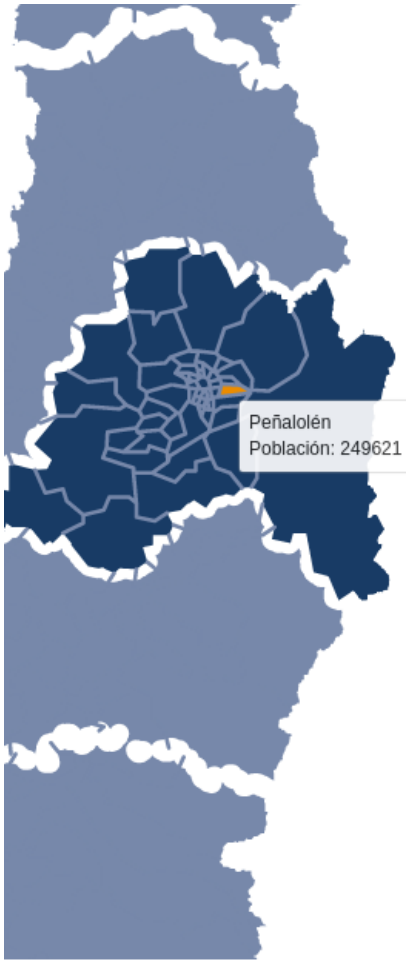


Figura 6.4: Minimapa de terreno, nivel comuna.

En el minimapa de terrenos 6.2 se selecciona una región del país sobre la que trabajar, que luego da opciones para seleccionar un área secundaria. Se puede elegir entre nivel de agregación Provincia (figura 6.3) y Comuna (figura 6.4). El minimapa está hecho usando la librería d3js[1].

Otra forma de seleccionar un área es usando el buscador. Usando jQuery se puede buscar por nombre un área sobre la que trabajar. Esta se comunica con el minimapa de figuras y con el mapa del terreno para mostrar el grafo (figura 6.5).

## Mapa de terreno

Cuando un área es seleccionada, sea por la barra de búsqueda o por el minimapa de regiones, se dibuja el grafo del área seleccionada sobre el mapa del terreno de figura 6.6

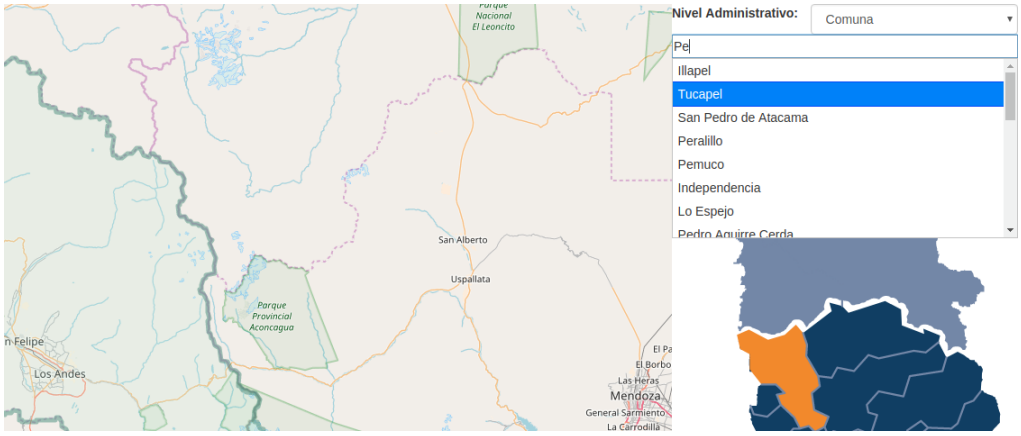


Figura 6.5: Barra de búsqueda

provisto por OpenStreetMap, usando Leaflet[4], una librería para creación de aplicaciones interactivas con mapas.

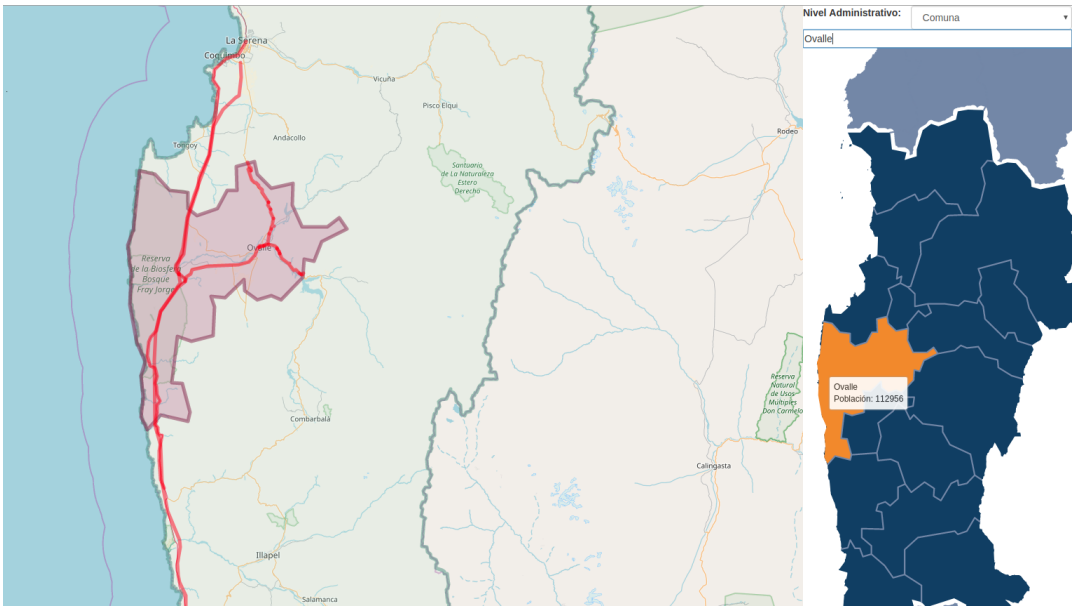


Figura 6.6: Mapa del terreno

En el mapa del terreno con el grafo dibujado, se puede empezar a trabajar sobre este. Los dos usos provistos son de creación y borrado de aristas. Estas aristas están representadas por polilíneas[1].

Para crear un cable nuevo, se uso del control provisto por la extensión de Leaflet, Leaflet.Draw[3]. Los cables que se creen pueden o no conectarse automáticamente al nodo mas cercano, dependiendo del usuario. También se podrá extender una linea existente si esta no está en un nodo conectado a mas aristas[2].

[1]Una polilínea es una secuencia conectada de segmentos de linea, creados como un único objeto.

[2]Si se uniera en un nodo conectado a mas aristas, estas no quedarían conectadas a la linea anterior, potencialmente quedando aristas no conectadas a un nodo

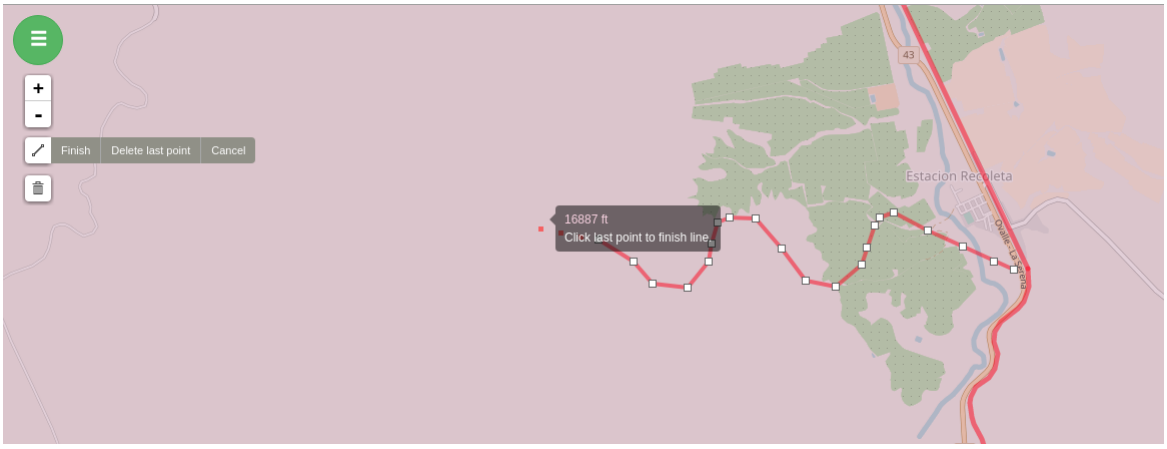


Figura 6.7: Creación de cable


Figura 6.8: Cable creado

Adicionalmente se pueden borrar cables existentes.


Figura 6.9: Interfaz para unir/conectar.

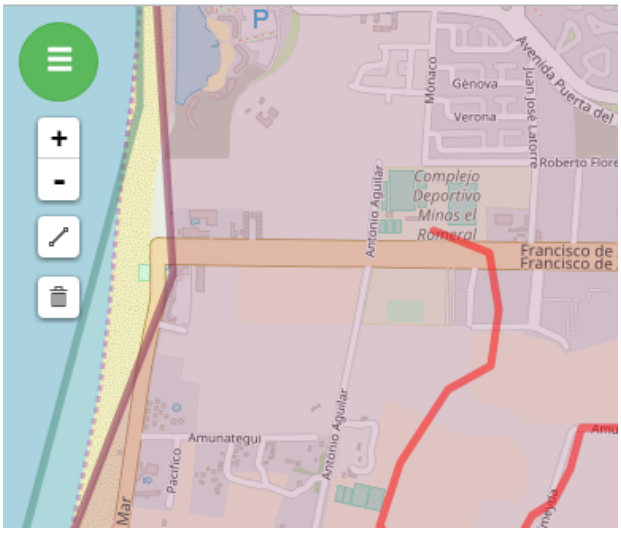

Figura 6.10: Resultado.

Vimos que la creación de cables puede automáticamente unir o conectar a la red existente. Dado que no solo es necesario extender la red creando cables nuevos, si no también modificar

la existente, se provee la opción de unir y conectar manualmente cables existentes. Esto se puede apreciar en las figuras 6.11, 6.12 y 6.13.

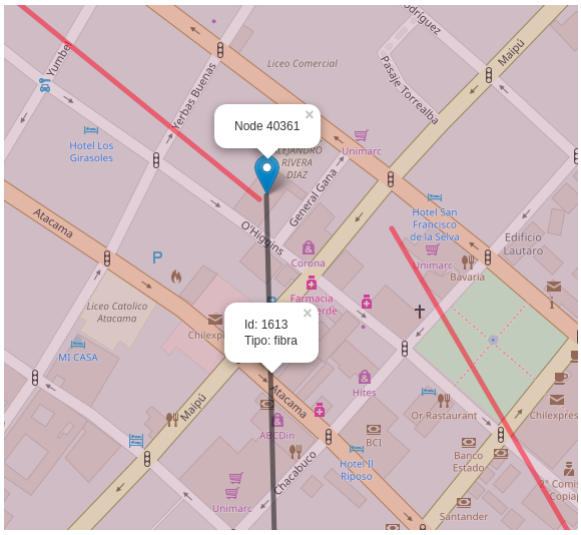


Figura 6.11: Mostrar nodos del cable.

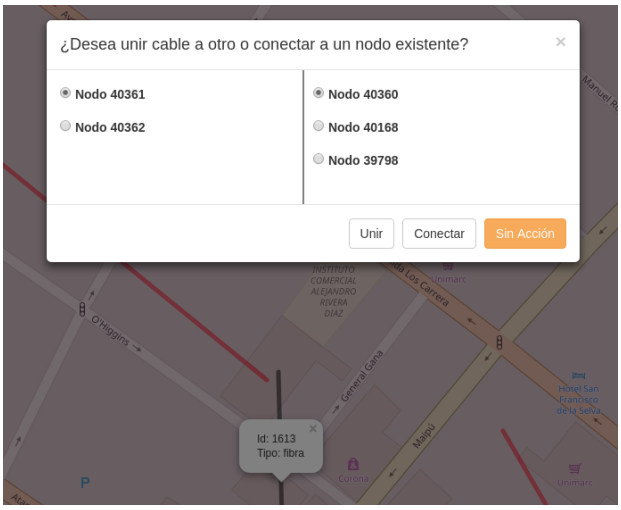


Figura 6.12: Interfaz para unir/conectar.

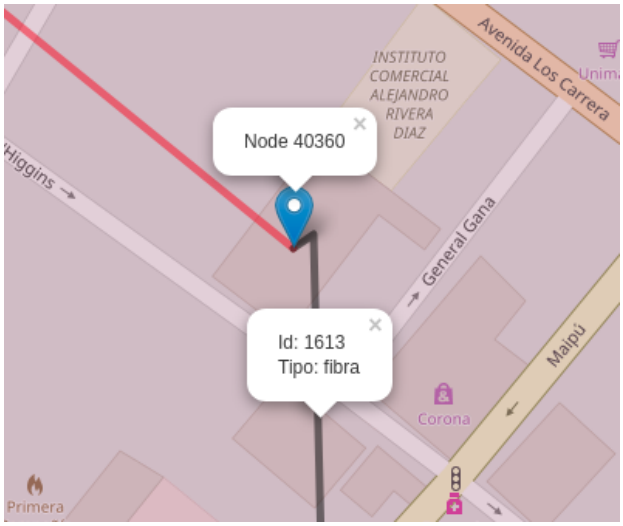


Figura 6.13: Resultado.

# Menú de opciones

Se puede acceder al menú de opciones haciendo click en el botón hamburguesa 6.14. En el menú de opciones 6.15 se puede seleccionar el comportamiento esperado de auto unir y conectar los cables creados, al igual que el comportamiento de los clicks sobre el mapa, como mostrar los nodos de un cable y conectar cables manualmente.

Además se puede acceder al historial de acciones 6.16, donde se podrá hacer un preview del grafo en un punto en el tiempo, y hacer un rollback sobre este. De esta forma, el grafo elimina los cables que no existian en ese tiempo y revive los que están en la tabla de historial y estuvieron activos en ese tiempo.

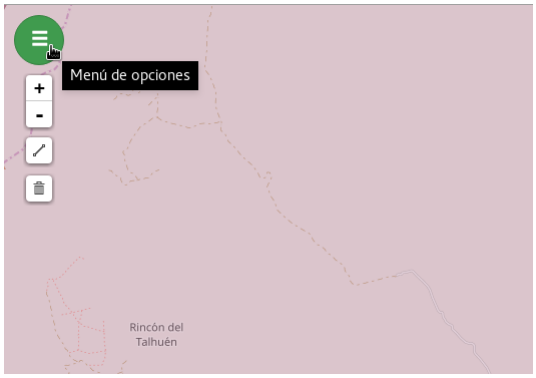


Figura 6.14: Botón hamburguesa

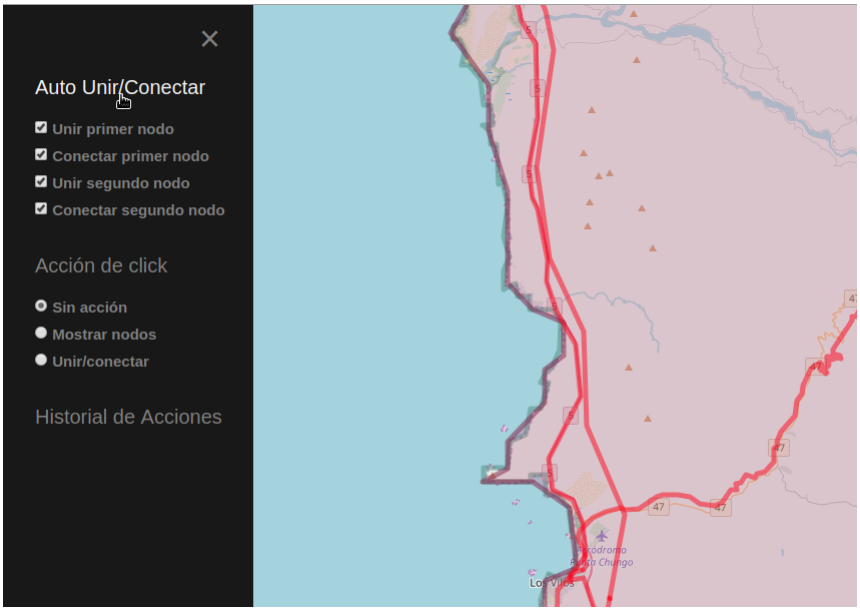


Figura 6.15: Menu de opciones

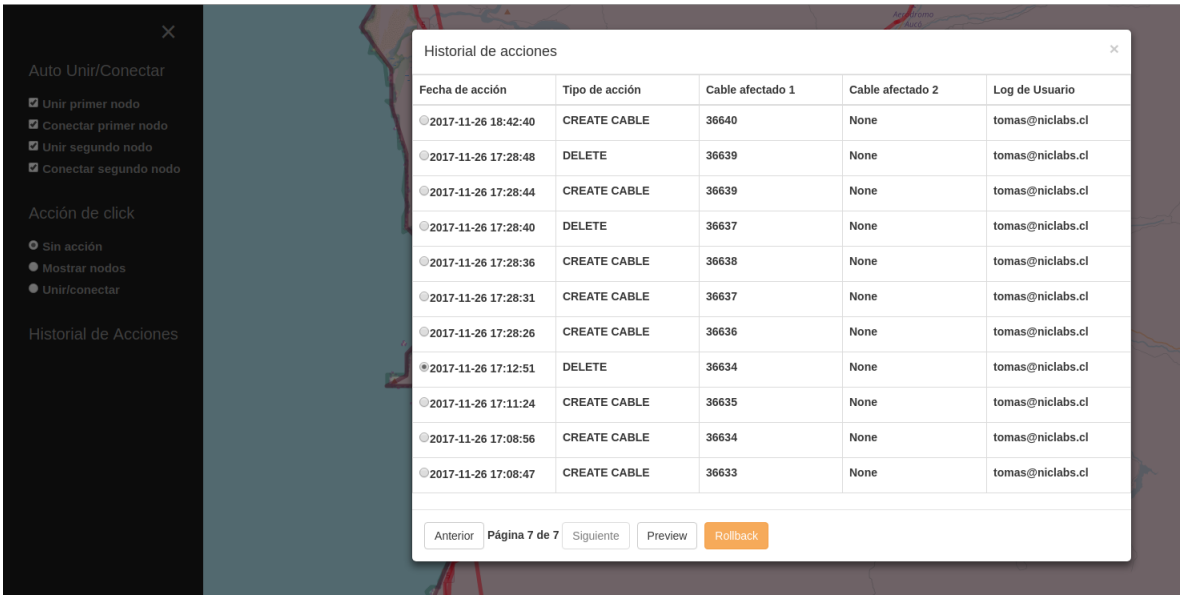


Figura 6.16: Historial de acciones