UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería de Telecomunicación

Simulación y visualización de redes de agua utilizando la librería WNTR de Python en una aplicación web.

Trabajo Fin de Grado

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

AUTOR/A: Zarzo Toutoundji, Vicente

Tutor/a: López Patiño, José Enrique

CURSO ACADÉMICO: 2022/2023

Resumen

En este proyecto elaborado durante el desarrollo de unas prácticas en una empresa, se va a implementar un nuevo módulo a una aplicación web ya existente, en la cual, aquellos usuarios (clientes) que dispongan de una red de distribución de aguas y tengan implementados en esta una serie de medidores desarrollados por la propia empresa, van a poder realizar una simulación actual de su red de aguas gracias a la librería WNTR (Water Network Tool for Resilience) de Python. Además de la simulación, estos clientes también van a tener disponible una visualización de la red, pudiendo obtener los parámetros de interés de los elementos que la componen: uniones, válvulas, tuberías, tanques y depósitos. El desarrollo del trabajo constará de la parte de simulación realizada con Python, de la parte de almacenado de los datos de la simulación, se utilizará PHP para la conexión con la API y la base de datos, y se utilizará JavaScript para la visualización de la red simulada.

Palabras clave: Front-end, Back-end, Laravel, Vue.js, Python, Web, API, Epanet, WNTR, Simulación.

Resum

En aquest projecte elaborat durant el desenvolupament d'unes pràctiques en una empresa, s'implementarà un nou mòdul a una aplicació web ja existent, en la qual, aquells usuaris (clients) que disposen d'una xarxa de distribució d'aigües i tinguen implementats en aquesta una sèrie de mesuradors desenvolupats per la pròpia empresa, podran realitzar una simulació actual de la seua xarxa d'aigües gràcies a la llibreria WNTR (Water Network Tool for Resilience) de Python. A més de la simulació, aquests clients també tindran disponible una visualització de la xarxa, podent obtindre els paràmetres d'interés dels elements que la componen: unions, vàlvules, canonades, tancs i depòsits. El desenvolupament del treball constarà de la part de simulació realitzada amb Python, de la part d'emmagatzemat de les dades de la simulació, s'utilitzarà PHP per a la connexió amb la API i la base de dades, i s'utilitzarà JavaScript per a la visualització de la xarxa simulada.

Paraules clau: Front-end, Back-end, Laravel, Vue.js, Python, Web, API, Epanet, WNTR, Simulació.

Abstract

In this project developed during the course of an internship in a company, a new module will be implemented to an existing web application, in which those users who have a water distribution network and have implemented in it a series of meters developed by the company itself, can perform a current simulation of their water network thanks to the WNTR (Water Network Tool for Resilience) library of Python. In addition to the simulation, these customers will also have a visualization of the network, obtaining the parameters of interest of the elements that compose it: connections, valves, pipes, tanks, and reservoirs. The development of the work will consist of the simulation part carried out with Python, the simulation data storage part, PHP will be used for the connection with the API and the database, and JavaScript for the visualization of the simulated network.

Keywords: Front-end, Back-end, Laravel, Vue.js, Python, Web, API, Epanet, WNTR, Simulation.

Índice

1. Introducción

Este proyecto de fin de grado ha sido desarrollado durante la realización de unas prácticas en *Datakorum Solutions S.L*, una compañía que desarrolla dispositivos electrónicos y ofrece soluciones *IoT End-to-End* en las áreas de agua, energía y movilidad. Específicamente el proyecto se enmarca en el área de agua y consiste en la implementación de un nuevo módulo a la aplicación web Baltoro. [1]

Para contextualizar *Baltoro*, esta aplicación permite al usuario gestionar y visualizar los dispositivos de *Datakorum* implementados en su red. Además, cada dispositivo dispone de gráficos diarios, semanales y/o mensuales de los datos que estos generan, dependiendo del tipo de dispositivo.

El nuevo módulo desarrollado para la aplicación permite a aquellos clientes que disponen de una red de distribución de aguas con nuestros dispositivos incorporados, obtener y visualizar una simulación de su red hidráulica. La simulación se realiza de forma automática y dinámica gracias a la librería de Python *WNTR* (*Water Network Tool for Resilience*).

1.1 Detalles aplicación web desarrollada

Para el desarrollo del nuevo módulo de la aplicación ha sido necesario el uso de distintos lenguajes de programación, cada uno ha sido utilizado para una parte especifica. Por un lado, tenemos la simulación la cual ha requerido el uso de Python, concretamente de la librería *WNTR* ya comentada. Por otro lado, nos encontramos con el guardado de los datos de la simulación en una base de datos *MySQL*, para ello se ha utilizado *PHP* para la conexión con la *API* y la base de datos. Y, por último, para la parte de visualización de la simulación ha sido necesario utilizar *JavaScript* para crear la vista.

2. Objetivos y finalidad

2.1 Objetivos del proyecto

El proyecto tiene como objetivo añadir un nuevo módulo a la aplicación web *Baltoro* para que aquellos clientes que dispongan de una red de aguas, puedan realizar simulaciones de su red gracias a los medidores desarrollados por *Datakorum* incorporados en esta.

Como meta en el proyecto se ha fijado la obtención y visualización de simulaciones de forma dinámica y automática de aquellos modelos fiables de redes que dispone la empresa.

En el proyecto no se van a realizar simulaciones a futuro, si no que se pretende que cada vez que se reciban los datos de los dispositivos se realice una simulación que muestre el estado actual de la red en un tiempo cero.

2.2 Objetivos personales

Durante el desarrollo del trabajo me he marcado los objetivos de profundizar y mejorar todos aquellos conocimientos obtenidos durante la carrera universitaria que puedan serme de utilidad.

Además de consolidar mis conocimientos existentes, también busco adquirir nuevos conocimientos sobre herramientas y lenguajes de programación que no he utilizado todavía y que me van a ser necesarios para el desarrollo del proyecto.

2.3 Finalidad

La finalidad actual que tiene el desarrollo de este nuevo módulo de la aplicación, es proporcionar al cliente/usuario la posibilidad de monitorizar al completo su red y poder observar los valores de los elementos de esta con la configuración y parámetros de entrada[1] actuales.

[1] Parámetros que no depende de la simulación, si no que se establecen inicialmente en cada elemento para poder realizarla. Son parámetros como las longitudes, los diámetros, alturas, etc.

3. Planificación y metodología

Para realizar este Trabajo de Fin de Grado y poder lograr los objetivos planteados ha sido necesario seguir una serie de pasos durante el desarrollo de las prácticas:

- Aprendizaje de las herramientas y tecnologías utilizadas en la aplicación.
- Aprendizaje de la estructura que presenta la aplicación.
- Búsqueda de información y documentación sobre la aplicación de *Epanet* y su librería de Python *WNTR*.
- Desarrollo de la simulación:
    - Aprendizaje de *Python*.
    - Creación *scripts* simulación.
    - Mejora de código.
- Desarrollo de la conexión simulación-visualización:
    - Diseño de las tablas necesarias en la base de datos.
    - Creación del *script PHP* para el almacenamiento de modelos.
    - Elaboración del *script PHP* para el almacenamiento de las simulaciones.
    - Creación de las funciones en *PHP* que permiten obtener los datos de la simulación en la clase de la vista.
    - Aprendizaje de *JavaScript* para la creación de la vista de la visualización de la simulación.
    - Diseño de la vista de simulación.
    - Mejora constante de la vista de la simulación.
- Corrección de errores y optimización de código.

Una vez adquiridos los conocimientos necesarios para iniciar el desarrollo, se ha procedido a realizar de manera simultánea y progresiva todas las partes del módulo de la aplicación (simulación, almacenado y visualización). Esta forma de trabajo ha permitido verificar los resultados de cada parte en cada versión de código, asegurando una funcionalidad correcta y una integración adecuada de todas las partes.

4. Tecnologías y herramientas utilizadas

Durante el desarrollo del proyecto ha sido necesario utilizar diversas herramientas y tecnologías relacionadas con la programación. Para hablar de ellas se ha optado por hacer una diferenciación en la parte del desarrollo que se han utilizado: *Front end* y *back end*.

4.1 Front end

En un entorno de programación, el *front end* se refiere a la parte de un sistema software que es visible por el usuario y con la que estos pueden interactuar directamente. Para el desarrollo de esta parte ha sido necesario utilizar *JavaScript, HTML y CSS*. [2]

4.1.1 JavasScript

JavasScript es un lenguaje de programación de alto nivel orientado a objetos, utilizado para agregar interactividad a las páginas y aplicaciones web, permitiendo realizar acciones dinámicas, animaciones, etc. Ha sido utilizado para la creación de la vista que contiene la visualización de la simulación. [3]

Para agilizar y facilitar el desarrollo se ha utilizado el *framework* progresivo *Vue.js* de *JavaScrips*. *Vue.js* es ideal para crear interfaces de usuario dinámicas y aplicaciones de una sola página. Además, facilita la manipulación del DOM[2] (*Document Object Modal*) de forma declarativa y ofrece funcionalidades útiles, como el enrutamiento, para mejorar la experiencia del usuario al navegar por la ampliación. [4]

```
document.getElementById("sidemenu").style.width = "20rem";
document.getElementById("sidemenu").style.padding = "0.75rem 0.75rem 0 0.75rem";
document.getElementById("menu").style.marginRight = "20rem";
```

Figura 1: Ejemplo de manipulación del DOM.

[2] Es una interfaz de programación para los documentos HTML y XML, que nos permite crear, modificar o eliminar elementos del documento.

4.1.2 HTML

HTML (*Hyper Text Markup Language*) es el componente más básico de la web. Proporciona la estructura básica a los elementos que componen una página web, como encabezados, párrafos, tablas, etc.

Utiliza una serie de marcas como <head>, <tittle>, <body>, <img> entre otras, para etiquetar texto, imágenes y diversos tipos de contenidos para mostrarlo en un navegador web. [6]

```
<div id="legend"
        class="baltoro-container-2 w-fit h-fit box-border z-50 ml-4 mb-11 left-0 bottom-0
                fixed flex flex-col py-1 px-2 text-gray-light text-sm hidden">
</div>
```

Figura 2:Código HTML para la leyenda del mapa.

4.1.3 CSS

CSS (*Cascading Style Sheets*) es el lenguaje utilizado para describir la presentación y el diseño de los elementos de una página web. Con CSS, es posible definir propiedades como el color, el tamaño y tipo de la fuente, los márgenes, los bordes y otros estilos para mejorar la apariencia visual. [5]

Para facilitar el desarrollo y agilizar el proceso de estilización, se ha utilizado el *framework* de CSS **Tailwind CSS**. Este *framework* proporciona un conjunto de clases predefinidas que permiten aplicar estilos de manera más rápida y sencilla.

```
.baltoro-container-2{
    border-radius: 0.375rem;
    box-shadow: 0 10px 15px -3px rgb(0 0 0 / 0.25);
    background: rgba(0,42,80,0.85);
    border-width: 0px;
}
```

Figura 3:Clase CSS utilizada para definir el estilo del contendor de la leyenda del mapa.

4.2 Back end

Back end es el término que se utiliza para definir la parte de un sistema que se ocupa del procesamiento, la gestión de datos y la comunicación entre distintas partes del sistema, es decir, se encarga de la lógica de una página web. En otras palabras, el *back end* incluye todo aquello que no es visible por el usuario de una aplicación web. [2]

Los lenguajes de programación *PHP* y *Python* han sido utilizados en esta parte del desarrollo.

4.2.1 PHP

PHP (*Hypertext Pre-Processor*) es un lenguaje de programación situado en el lado del servidor que permite la generación de páginas web dinámicas, que se generan a tiempo a real en función de la interacción del usuario. [7]

Tiene un amplio soporte para conectarse y manipular bases de datos. En definitiva, su uso favorece la conexión entre el servidor y la interfaz del usuario.

En el proyecto, *PHP* permite la conexión entre la base de datos y la vista para obtener y poder mostrar los datos de la simulación.

```php
public function GET_data_network($idgroup_selected){

    $junctions = array();
    $pipes = array();
    $valves = array();

    Log::channel('privateAPI')->info('EpanetController GET_data_network', [
        'idgroup_selected' => $idgroup_selected
    ]);


    $junctionsData = DB::select('SELECT j.id, j.idclient, j.idnode, j.GPS_lat, j.GPS_lng, j.type, j.iddevice, jd.demand, jd.head, jd.pressure, j.elevation, jd.date
        FROM baltoro.EPANET_junctions j
        INNER JOIN baltoro.EPANET_junctions_data jd ON j.idnode = jd.idnode
        WHERE jd.idclient = '.$idgroup_selected.' AND jd.date = (SELECT MAX(date)
        FROM baltoro.EPANET_junctions_data WHERE idclient = '.$idgroup_selected.')');

    foreach($junctionsData as $junction){
        $junctions[] = $junction;
    }
```

Figura 4: Función PHP mediante la cual obtenemos en los valores de la simulación en la vista.

Para la parte de *back end*, utilizamos el *framework Laravel* de *PHP*. Laravel es uno de los *frameworks* más populares utilizados para el desarrollo de aplicaciones web robustas.

Sigue un patrón de diseño conocido como MVC (Modelo-Vista-Controlador) y proporciona una amplia gama de herramientas para facilitar el desarrollo de aplicaciones web. [8]

4.2.2 Python

Python es un lenguaje de programación de alto nivel caracterizado por su amplia biblioteca estándar, que proporciona una gran cantidad de módulos y funciones que facilitan el desarrollo de aplicaciones. *Python* es un lenguaje de programación multiparadigma, esto significa que permite varios estilos de programación. Es ampliamente utilizado en aplicaciones web, desarrollo software, ciencia de datos y el *machine learning*. [9]

En el proyecto este lenguaje ha sido utilizado para crear los *scripts* encargados de realizar la simulación de los modelos de red de distribución de agua. Ha sido necesario el uso de la librería *WNTR*, cuyas características y funcionalidades serán explicadas más adelante.

4.2.3 MySQL

Para el almacenamiento de los datos de la simulación se ha utilizado MySQL, que es un sistema de gestión de bases de datos relacionales de código abierto. Permite almacenar, organizar y gestionar grandes cantidades de datos de manera eficiente.

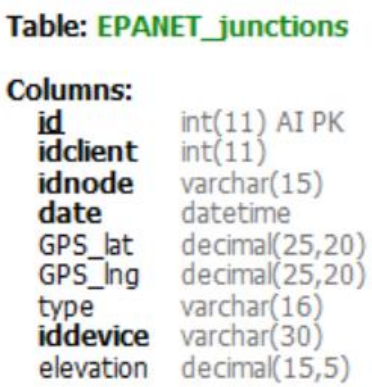


Figura 5: Estructura de la tabla Epanet_junctions, que contiene los datos necesarios de los nodos para la simulación.

Además de herramientas y lenguajes de programación utilizados para el desarrollo de código, también ha sido necesario aprender conocimientos sobre la administración y navegación en un servidor *Ubuntu* (donde se encuentra alojado el *middleware*[3] de *Datakorum*) para poder ejecutar de forma dinámica el *script* de simulación utilizando la herramienta *cron*.

Cron es un administrador regular de procesos en segundo plano utilizado en el sistema operativo Ubuntu. [10]

Los procesos/archivos a ejecutar y la hora a la que deben hacerlo están definidos en el fichero de *crontab*, de forma que cron revisa constantemente las tareas establecidas en *crontab* y las va ejecutando si corresponde. En el proyecto ha sido utilizado para ejecutar los *scripts* de *Python* encargados de la simulación, para así obtener las simulaciones cada quince minutos.

Figura 6: Definición ejecución del archivo de simulación modelSim.py.

Por último, para la edición de código de los diversos lenguajes de programación se ha utilizado *Visual Studio Code*, elegido por su utilización a lo largo de diversos proyectos llevados a cabo en la universidad.

[3] El middleware es un software que se sitúa entre sistema operativo y aplicación. Mediante el cual se pueden comunicar varias aplicaciones entre sí. Actúa como un puente entre tecnologías, herramientas y bases de datos para que se puedan integrar en un único sistema.

5. Epanet y WNTR

Con el objetivo de poder realizar la sección de simulación de las redes de agua, ha sido necesario entender cómo funciona el software *Epanet*, además de la librería *WNTR* de Python relacionado con este último.

Epanet es un software desarrollado por la Agencia de Protección Ambiental de Estados Unidos (EPA) con la colaboración de distintas instituciones entre las que se encuentra la *UPV* (Universitat Politècnica de Vàlencia), que permite realizar simulaciones del comportamiento hidráulico y de la evolución de la calidad de agua en redes de suministro a presión. Cuando se habla de red se hace referencia al conjunto de tuberías, válvulas, bombas, tanques, nodos y depósitos. [11]

En este proyecto se ha puesto el foco en el análisis del comportamiento hidráulico.

Mediante Epanet, se pueden realizar análisis hidráulicos de redes partiendo de las características físicas de los elementos que las componen, tales como la longitud y diámetro de las tuberías, la elevación del agua en los tanques y depósitos. Es fundamental conocer la demanda inicial de cada uno de los nodos de la red, para poder obtener una simulación correcta y precisa.

El programa brinda la capacidad de hacer un seguimiento a lo largo de la simulación de todos los elementos que componen la red. Así, se pueden obtener datos importantes, como las presiones en los nudos, la velocidad del agua en las tuberías, el estado (abierto/cerrado) de las válvulas y tuberías a medida que transcurre la simulación, entre otros.

Esta capacidad de análisis detallado proporciona una visión clara del comportamiento hidráulico y de calidad del agua en la red, lo que resulta crucial para el diseño, mantenimiento y mejora de los sistemas de distribución de agua.

Para la realización del proyecto no ha sido necesario entrar en profundidad en el funcionamiento de Epanet, ya que es el propio cliente el que proporciona el modelo de su red, debido a que hay parámetros y conocimientos que escapan a mi conocimiento y al de la empresa y requiere de un experto en hidráulica. Pero si ha sido necesario entender el funcionamiento de forma básica para la posterior utilización de la librería de Python WNTR.

Los pasos básicos a seguir para la utilización de Epanet son:

1. Dibujar o importar una representación de la red del sistema de distribución a simular. Figura7.
2. Editar las propiedades de los objetos que conforman el sistema. Figuras 9 y 10.
3. Describir como trabaja el sistema.
4. Determinar las opciones de análisis. Figura 8.
5. Iniciar el análisis hidráulico o de calidad de agua.
6. Obtener y visualizar los resultados.

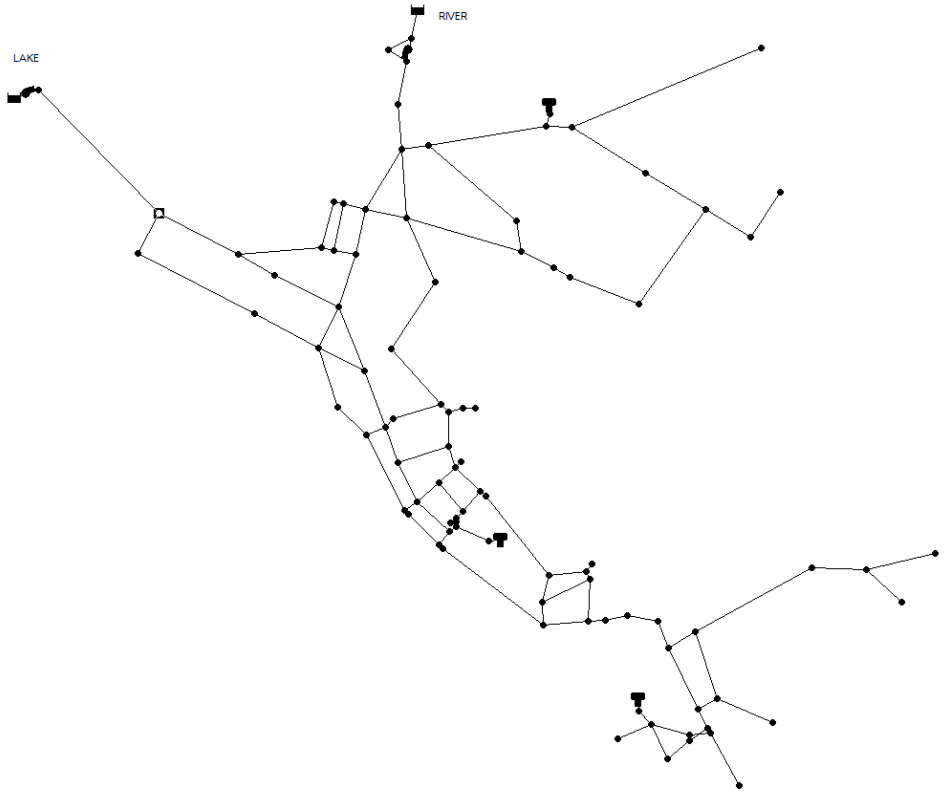


Figura 7: Modelo de red de ejemplo utilizado para el desarrollo.

Epanet dispone de multitud de parámetros configurables a gusto del usuario para llevar a cabo la simulación de la forma en que se desee.

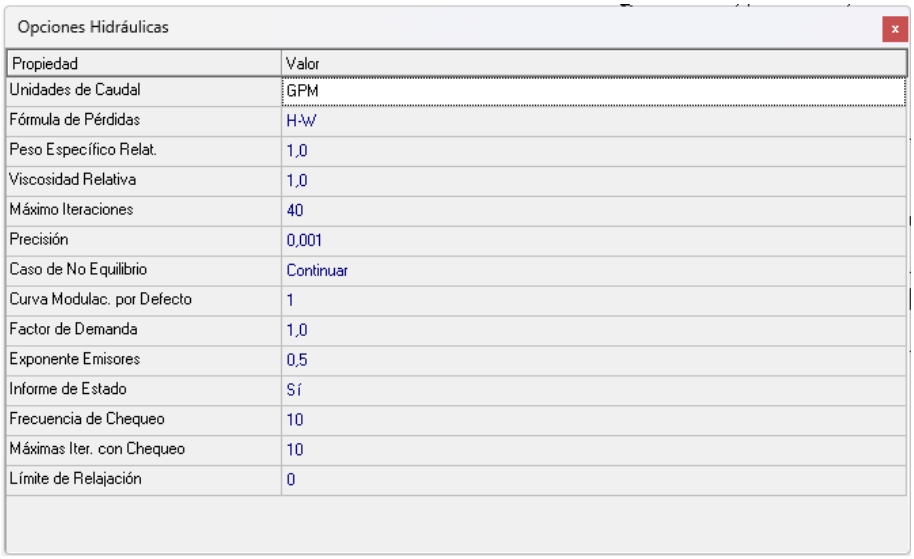


Figura 8: Opciones hidráulicas Epanet

En la figura 8 se pueden observar las opciones hidráulicas que se pueden modificar, podemos destacar entre ellas las unidades de caudal, que como su propio nombre indica define las unidades del caudal y de la demanda de los nodos de la red, este parámetro tiene diferentes posibles valores según estés utilizando las unidades del Sistema Internacional o unidades americanas, siendo estos algunos de ellos:

- LPS: litros por segundo.
- LPM: litros por minuto.
- CMH: metros cúbicos por hora.
- GPM: galones por minuto.

También es relevante señalar la fórmula de pérdida de cargas, que es la disminución de presión que sufre el agua a medida que fluye a través de las tuberías debido a las pérdidas existentes en esta. Es importante seleccionar correctamente este parámetro porque dependiendo del tipo de tuberías del modelo, puede ser más precisa una u otra fórmula. Entre las distintas fórmulas disponibles para el cálculo de la pérdida de carga tenemos la fórmula de *Hazen-Williams* (H-W), que se basa en el diámetro de la tubería, la seguridad interna y la velocidad de flujo.

Nudo de Caudal 103

Propiedad	Valor
*ID Nudo de Caudal	103
Coordenada X	12,96
Coordenada Y	21,31
Descripción	
Etiqueta	
*Cota	43
Demanda Base	133,2
Curva Modul. Demanda	
Tipos de Demanda	1
Coeficiente del Emisor	
Calidad Inicial	
Intensidad de la Fuente	
Demanda Actual	Sin Valor
Altura Total	Sin Valor
Presión	Sin Valor
Calidad	Sin Valor

Figura 9: Propiedades de los nodos de la red.

Tubería 177

Propiedad	Valor
*ID Tubería	177
*Nudo Inicial	159
*Nudo Final	161
Descripción	
Etiqueta	
*Longitud	2000
*Diámetro	30
*Rugosidad	141
Coef. Pérdidas Menores	0
Estado Inicial	Abierta
Coef. Reacción en el Medio	
Coef. Reacción en la Pared	
Caudal	Sin Valor
Velocidad	Sin Valor
Pérdida Unitaria	Sin Valor
Factor Fricción	Sin Valor

Figura 10:Parámetros de las tuberías de la red.

Todos estos pasos que se deben seguir cuando se pretende realizar un análisis, no han sido necesarios en mi caso, debido a que es el propio cliente el que nos proporciona un modelo correcto de su red.

En cuanto al análisis contamos con dos posibilidades, análisis en régimen permanente y análisis de periodo extendido.

Como su propio nombre indica, en el análisis en régimen permanente las demandas de los nodos permanecen intactas, es decir, no varía con el tiempo.

Por otro lado, con el análisis de periodo extendido se logran obtener simulaciones más realistas gracias a la creación de curvas de modulación. Estas curvas son patrones que permiten que la demanda en los nodos varie de forma periódica a lo largo del día.

Para lograr entender por completo un modelo de red y diversas características que se van a comentar más adelante es necesario explicar de forma breve que es cada elemento de la red:

- Conexiones(nodos/nudos/uniones/*junctions*): Son puntos en la red donde se unen las líneas por donde entra o sale agua de la red.
- Embalses (*Reservoirs*): Son nudos que representan una fuente externa infinita o un sumidero para el sistema. Se utilizan para modelizar lagos, ríos y conexiones a otros sistemas.
- Tanques (*Tanks*): Los tanques son nudos con capacidad de almacenamientos, en los cuales el volumen de agua almacenada puede variar con el tiempo.
- Tuberías (*Pipes*): Son líneas que llevan el agua de un punto de la red a otro. La dirección de caudal va desde el extremo con altura piezométrica[4] mayor hacia el extremo de meno altura. Los extremos de las tuberías son siempre nudos.
- Valvulas (Valves): Son líneas que limitan la presión y el caudal en puntos específicos de la red. Dentro de estas existen diversos tipos, entre los cuales encontramos:
    - Válvulas reductoras de presión.
    - Válvulas sostenedoras de presión.
- Bombas (*Pumps*): Son elementos que aportan energía al fluido incrementando su altura piezométrica.

[4] Suma del potencial de elevación y del potencial de presión en un punto de un líquido. Se expresa en unidades de altura.

Todos los conocimientos sobre Epanet redactados anteriormente han sido obtenidos de su manual de usuario. [12]

Debido a que Epanet es una aplicación de escritorio no nos sirve para realizar simulaciones automáticas y dinámicas de todos los modelos que disponemos, es por ello que decidimos utilizar ***WNTR***.

WNTR es una librería de Python diseñada para simular y analizar sistemas de redes de distribución de aguas. *WNTR* está basado en el software Epanet, ambos tienen la misma finalidad, son utilizados para el modelado y la simulación de redes de agua.

Podríamos decir que *WNTR* es Epanet convertido en una librería de Python, debido a esto se recomienda que previo a la utilización de esta librería se debe tener un conocimiento básico de Epanet y estar familiarizado con este.

Para poder realizar la simulación es necesario disponer de un archivo que almacena todas las características y parámetros del modelo de red a simular. Este archivo es generado mediante Epanet.

En este archivo tiene un formato específico para poder ser utilizado por la librería *WNTR*. Este se divide en secciones y en cada sección se describen distintos contenidos. Las secciones que conforman este archivo son:

Componentes de la red	**Funcionamiento del sistema**	**Calidad del agua**	**Opciones e informes**	**Plano y etiquetas de red**
[TITLE]	[CURVES]	[QUALITY]	[OPTIONS]	[COORDINATES]
[JUNCTIONS]	[PATTERNS]	[REACTIONS]	[TIMES]	[VERTICES]
[RESERVOIRS]	[ENERGY]	[SOURCES]	[REPORT]	[LABELS]
[TANKS]	[STATUS]	[MIXING]		[BACKDROP]
[PIPES]	[CONTROLS]			[TAGS]
[PUMPS]	[RULES]			
[VALVES]	[DEMANDS]			
[EMITTERS]				

Tabla 1: Secciones archivo ".inp" para la simulación.

Algunas de las secciones más destacables son:

- [JUNCTIONS]: En esta sección se describe los nodos de la red. Para cada nodo se especifica su identificador, la elevación, y la demanda.
- [RESERVOIRS]: Aquí se definen los embalses de agua. Se especifica su identificador y la altura del nivel del agua con respecto al punto de referencia.
- [PIPES]: En esta sección se detallan las tuberías que conforman la red. Se indica el identificador, el nodo inicio y nodo fin, la longitud y diámetro de la tubería, la rugosidad y el estado (abierto/cerrado).
- [VALVES]: Aquí es donde se definen las características de las válvulas. Teniendo un identificador, un nodo de inicio y un nodo final, el diámetro y el tipo de válvula entre otros.
- [DEMANDS]: Es un suplemento de la sección [JUNCTIONS] para definir las demandas en los nudos.
- [COORDINATES]: En esta sección se definen las coordenadas geográficas de los nodos de la red, siendo esto vital para la posterior visualización de la simulación.
- [OPTIONS]: Define las distintas opciones de simulación como las unidades o la fórmula para calcular las pérdidas de caga.

La información sobre la librería *WNTR* ha sido extraída de su página web de documentación. [13]

Durante el desarrollo de este proyecto se han utilizado diversos modelos de redes para hacer pruebas con las simulaciones, llegando a utilizar desde los modelos de clientes hasta prototipos de ejemplo proporcionados en la documentación de *WNTR*.

```
[RESERVOIRS]
;ID              Head            Pattern
 River           220.0                   ;
 Lake            167.0                   ;

[TANKS]
;ID              Elevation       InitLevel       MinLevel        MaxLevel        Diameter        MinVol          VolCurve        Overflow
 1               131.9           13.1            .1              32.1            85              0                               ;
 2               116.5           23.5            6.5             40.3            50              0                               ;
 3               129.0           29.0            4.0             35.5            164             0                               ;

[PIPES]
;ID              Node1           Node2           Length          Diameter        Roughness       MinorLoss       Status
 20              3               20              99              99              199             0               Open    ;
 40              1               40              99              99              199             0               Open    ;
 50              2               50              99              99              199             0               Open    ;
 60              River           60              1231            24              140             0               Open    ;
 101             10              101             14200           18              110             0               Open    ;
 103             101             103             1350            16              130             0               Open    ;
 105             101             105             2540            12              130             0               Open    ;
 107             105             107             1470            12              130             0               Open    ;
 109             103             109             3940            16              130             0               Open    ;
 111             109             111             2000            12              130             0               Open    ;
 112             115             111             1160            12              130             0               Open    ;
 113             111             113             1680            12              130             0               Open    ;
 114             115             113             2000            8               130             0               Open    ;
 115             107             115             1950            8               130             0               Open    ;
 116             113             193             1660            12              130             0               Open    ;
```

Figura 11: Secciones [Reservoirs], [Tanks] y [Pipes] del modelo de ejemplo Net3.inp

WNTR ha sido utilizada en dos *scripts* de Python. En uno se emplea facilitar el almacenamiento eficiente de los datos del modelo (parámetros de entrada) en la base de datos, lo que permite una gestión organizada y accesible de la información. En el segundo *script*, *WNRT* se encarga de la simulación de los modelos y del posterior almacenamiento de los resultados en la base de datos.

6. Dispositivos Datakorum

En este punto se van a comentar las características de los dispositivos de *Datakorum* cuyos datos utilizamos para las simulaciones de los modelos de redes de los clientes.

Estos dispositivos proporcionan información sobre la presión, la calidad del agua, el caudal y el consumo, dependiendo obviamente, del tipo de dispositivo.

Si el dispositivo está funcionando de forma correcta realiza el envío de una trama que contine los datos generados, con un periodo de quince minutos. En función del dispositivo la trama de envío de datos tiene un formato diferente.

Por norma general, en este tipo de mensaje se indica que tipo de sensor ha generado los datos (sensor de presión y/o sensor de consumo), el valor de los datos generados y la fecha de los datos.

Además de este tipo de mensajes para el envío de datos, los dispositivos también generan y envían tramas de información sobre el estado del dispositivo y de la red, y de registros de errores, entre otras. Este tipo de tramas permiten monitorizar el dispositivo, pudiendo saber si se producen errores en este y que tipo de errores suceden.

Las figuras que se muestran a continuación no forman parte del desarrollo del nuevo módulo de la aplicación, pero se añaden a este proyecto para que se vea de forma clara los datos que puede proporcionar un dispositivo de agua de *Datakorum*:

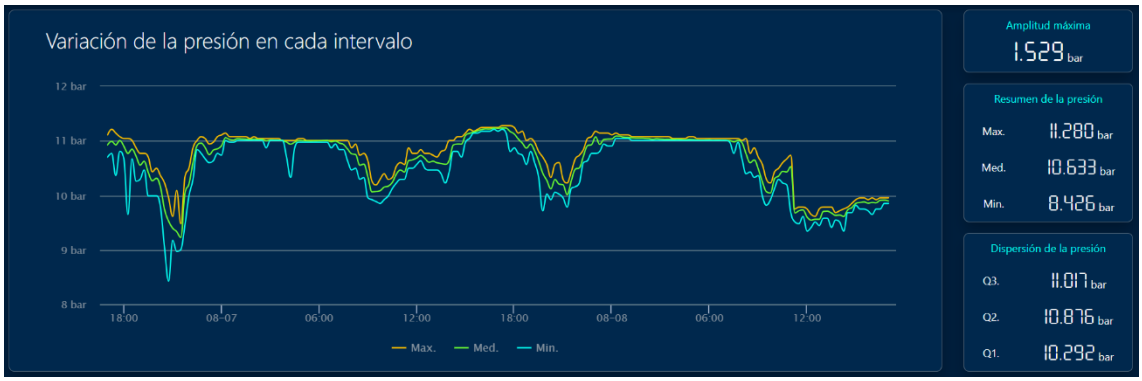


Figura 12: Variación de la presión en un dispositivo.

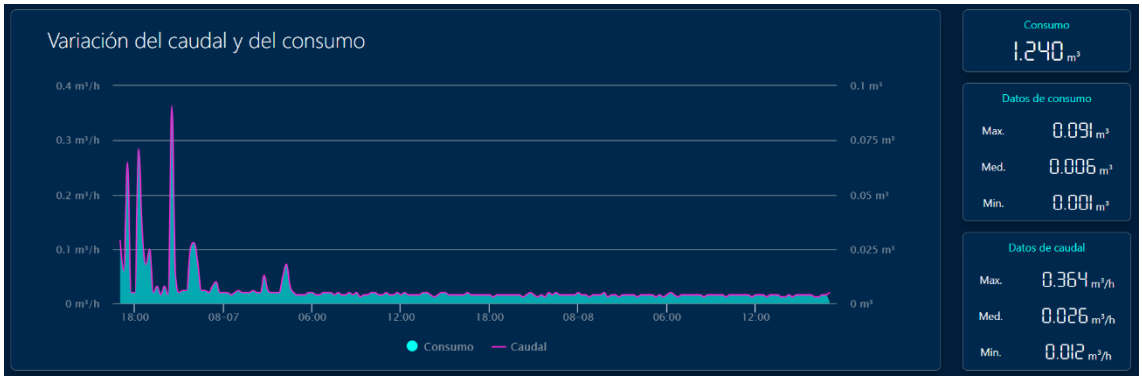


Figura 13: Variación del caudal y del consumo de un dispositivo.

En este caso, serán los valores de los últimos consumos de cada dispositivo de la red los que se utilicen para la simulación. Los consumos se utilizarán como demandas iniciales de los respectivos nodos de las redes de los clientes.

Figura 14: Dispositivo PIPE20P.

7. Análisis de la aplicación

7.1 Requisitos

Para poder realizar y visualizar una simulación de una red de distribución de agua deben darse una serie de requisitos por parte del cliente/usuario:

- El cliente debe de disponer de un modelo de red adecuado y sin errores.
- El cliente debe de tener integrados los medidores en todos los nodos de la red, para disponer de información en cada punto de la red.[5]

Cuantos más dispositivos se tengan incorporados dentro de la red, más monitorizada estará, lo que proporcionará simulaciones más precisas.

Además, como se indica en el primer requisito, es necesario que el modelo sea preciso y concreto en comparación con la red hidráulica real, si no es muy probable que los resultados de la simulación sean erróneos.

7.2 Esquemas

Para comprender el nuevo módulo de la aplicación, se han creado una serie de esquemas que representan diversas situaciones y casos que pueden suceder. Estos esquemas se han diseñado para ilustrar de forma clara los diferentes flujos que pude tomar la aplicación respecto a las simulaciones

[5] Este requisito no se incluye para aquellos modelos de ejemplo utilizados durante el proyecto, en los cuales no disponemos dispositivos integrados debido a que no son redes reales.

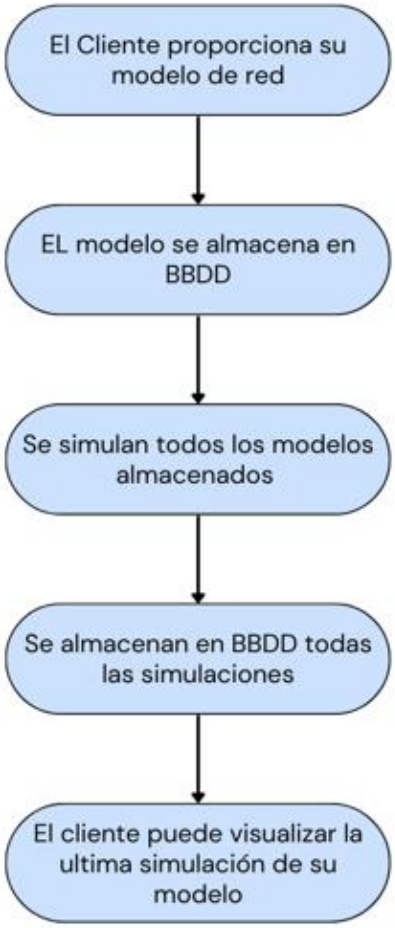


Figura 16: Flujo que sigue la aplicación cuando se dispone de un nuevo modelo.

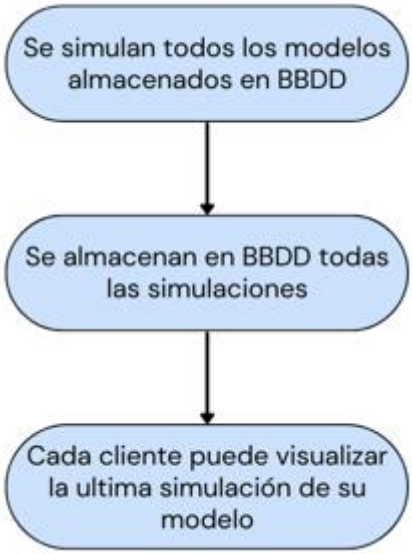


Figura 15. Flujo que sigue la aplicación de forma general.

Existen dos flujos distintos pero similares que sigue la aplicación:

1. Cuando un nuevo cliente proporciona su modelo de red o cuando un cliente, ya existente, proporciona un nuevo modelo o una modificación del anterior.
    - En este caso se almacenarán los datos del modelo en base de datos y posteriormente se simularán todos los modelos uno tras otro (de todos los clientes). Según vayan terminando las simulaciones estas se almacenarán en base de datos, haciendo posible que el cliente las visualice mediante la aplicación web.
2. Cuando no se incorpora ningún modelo nuevo.
    - Aquí directamente se simula los modelos con la frecuencia de tiempo establecida en *crontab*, posteriormente se almacenan todas las simulaciones y, por último, estas son visibles por los clientes.

8. Diseño y desarrollo del módulo de la app

En esta sección, se va a explicar el desarrollo de cada una de las partes que componen el nuevo módulo de la aplicación, para ello es necesario separar y diferenciar cada parte, aunque estén todas relacionas y dependan unas de otras.

8.1 Simulación

Esta es la piedra angular del desarrollo de la aplicación, ya que sin la simulación no tendría sentido el proyecto. Todos los conocimientos aprendidos sobre *Epanet* y *WNTR* son utilizados aquí.

El proceso de simulación consta de la ejecución de dos *scripts* de *Python* diferentes, cada uno con una función. En ambos *scripts*, en cada ejecución se llevan a cabo las tareas respectivas para todos los modelos existentes en la base de datos, es decir, no se ejecutan una vez para cada modelo.

En primer lugar, se ejecutará el *script* para almacenar los datos de interés de los modelos (*loadDeviceData.py*), datos que no dependen de la simulación, si no que la simulación depende de estos, como las longitudes de las tuberías y demandas iniciales de los nodos entre otros.

```
cursor.execute("SELECT * FROM epanet_models")
resultados = cursor.fetchall()

if  resultados:
    modelos = resultados

    if len(modelos) > 0:
        # creamos los CSV con las columnas correspondientes a cada elemento
        columnsJunctions = ['idclient', 'idnode', 'date', 'GPS_lat', 'GPS_lng', 'type', 'iddevice', 'elevation']
        columnsPipes = [ 'idclient', 'idpipe', 'date', 'start_node_name', 'end_node_name', 'length', 'diameter', 'roughness', 'minor_loss', 'status']
        columsValves = [ 'idclient', 'idvalve', 'date', 'start_node_name', 'end_node_name', 'diameter', 'setting', 'minor_loss', 'status']

        for model in modelos:

            dispositivos = []

            logging.info("Data loading of the model "+ str(model[1]) + " has started")
            file_name = model[2]
            idgroup = model[3]
            reference_1 = model[4]
            GPS_lat_1 = model[5]
            GPS_lng_1 = model[6]
            reference_2 = model[7]
            GPS_lat_2 = model[8]
            GPS_lng_2 = model[9]

            dfJ = pd.DataFrame(columns=columnsJunctions)
            dfJ.to_csv(routeCSV + str(idgroup) +'_EPANET_junctions.csv', index=False)

            dfP = pd.DataFrame(columns=columnsPipes)
            dfP.to_csv(routeCSV + str(idgroup) +'_EPANET_pipes.csv', index=False)

            dfV = pd.DataFrame(columns=columsValves)
            dfV.to_csv(routeCSV + str(idgroup) + '_EPANET_valves.csv', index=False)

            #wn = wntr.network.WaterNetworkModel('C:/MAMP/htdocs/haltoroplatform/epanet/models/' + file_name )
            wn = wntr.network.WaterNetworkModel([redacted]models/' + file_name )
            #wn = wntr.network.WaterNetworkModel('/var/www/html/public/epanet/models/' + file_name )

            longlat_map = {reference_1:(float(GPS_lng_1), float(GPS_lat_1)), reference_2: (float(GPS_lng_2), float(GPS_lat_2))} #example.inp


            wn2 = wntr.morph.convert_node_coordinates_to_longlat(wn, longlat_map)

            if(idgroup != 0):
                dispositivos = getIdDevices(idgroup)

            getDataSim()

            logging.info("Data loading of the model:  "+ str(model[1]) + " has ended")


        conexion.close()
        logging.info('All the models has ended \n\n')
    else:
        logging.warning('There are not models to simulate \n\n')
```

Figura 17: Script loadDeviceData.py

Como ya se ha comentado anteriormente, para realizar una simulación es necesario disponer de un archivo ".inp" que contenga el modelo de aquella red que se desea simular, indicándose en este, todos los parámetros necesarios de cada uno de los elementos de la red.

Todos aquellos archivos ".inp" de las redes que se van a simular están almacenados en el servidor de la aplicación, de forma que en el *script* de simulación se hace una petición a la base de datos para obtener el nombre de los modelos disponibles.

id	name	file_name	idgroup	reference_1	GPS_lat_1	GPS_lng_1
1	Modelo example	example.inp	556	Main_WTW	43.39751100000000000000	-80.53297500000000000000
[redacted]						
3	Modelo Net3	Net3.inp	557	10	43.39751100000000000000	80.53297500000000000000

Figura 18: Tabla Epanet_models

En la figura 18 se observan lo registros de la tabla *Epanet_models*, en la cual cada uno de los registros tiene un campo "file_name" que almacena el nombre del archivo del modelo de ese grupo/cliente.

Si tras hacer la consulta a la base de datos se encuentran modelos, se generará para cada uno de estos un archivo csv para cada tipo de elemento de la red (uniones, tuberías y válvulas), este archivo será utilizado para almacenar la información de cada modelo.

Cada tipo de csv tendrá unas columnas específicas, como se muestra en la figura 17, ya que no todos los elementos tienen los mismos campos.

```
junctions = {'idclient':idgroup, 'idnode': idnode, 'date': date, 'GPS_lat':GPS_lat, 'GPS_lng':GPS_lng, 'type':type, 'iddevice': iddevice, 'elevation':elevation}

dfJ = pd.read_csv(routeCSV + str(idgroup) +'_EPANET_junctions.csv')

new_data = pd.DataFrame(junctions)
df = pd.concat([dfJ, new_data], ignore_index=True)
df.to_csv(routeCSV + str(idgroup) +'_EPANET_junctions.csv', mode='w', header=True, index=False)
```

Figura 19: Almacenamiento de los datos de las Junctions en el archivo csv en loadDeviceData.py

En la figura 19 se muestra cómo se almacenan los datos de las uniones en el csv. Este proceso se repetirá para cada elemento y modelo, por ejemplo, para uno de los modelos de simulación utilizados durante el desarrollo con el identificador idgroup igual a 557, se generarán los archivos 557_Epanet_junctions.csv, 557_Epanet_pipes.csv y 557_Epanet_valves.csv.

idclient	idnode	date	GPS_lat	GPS_lng	type	iddevice	elevation
557	10	2023-07-28 06:45:26.983250	43.39751100098435	2.532974999992446	Junction	NULL	44.8056
557	15	2023-07-28 06:45:26.983250	42.58021489304524	0.8306211650850488	Junction	NULL	9.7536
557	20	2023-07-28 06:45:26.983250	42.76695134677149	1.4394275640187648	Junction	NULL	39.3192
557	35	2023-07-28 06:45:26.983250	43.510895424166165	0.9053333283520568	Junction	NULL	3.81
557	40	2023-07-28 06:45:26.983250	43.48572703523979	0.7932340231352433	Junction	NULL	40.20312
557	50	2023-07-28 06:45:26.983250	43.538868081598565	0.17725440307306467	Junction	NULL	35.5092
557	60	2023-07-28 06:45:26.983250	42.83439693639282	1.856887941274413	Junction	NULL	0.0
557	601	2023-07-28 06:45:26.983250	42.88072146054374	1.8830141576179282	Junction	NULL	0.0
557	61	2023-07-28 06:45:26.983250	42.87487317017395	1.8262343760929909	Junction	NULL	0.0
557	101	2023-07-28 06:45:26.983250	43.42722627542949	2.064815257753658	Junction	NULL	12.8016
557	103	2023-07-28 06:45:26.983250	43.516410212201755	2.0359937099151058	Junction	NULL	13.1064
557	105	2023-07-28 06:45:26.983250	43.38677723255971	1.8271976984324791	Junction	NULL	8.6868
557	107	2023-07-28 06:45:26.983250	43.36926360478182	1.7141141888606786	Junction	NULL	6.7056
557	109	2023-07-28 06:45:26.983250	43.4536618426623	1.6867868214429116	Junction	NULL	6.18744

Figura 20: Datos almacenados en 557_EPANET_junctions.csv.

Posterior a la ejecución de este *script*, se ejecutaría *modelsSim.py*. Este se encarga de la simulación de los modelos.

Su estructura es similar a *loadDeviceData* pero en este, se generan nuevos archivos csv diferentes a los anteriores, ya que en estos se almacenan los parámetros obtenidos tras la simulación, valores de presiones y demandas entre otros.

```
junctions = { 'idclient':idgroup, 'idnode': idnode, 'iddevice': iddevice, "date":str(date), 'demand':demand, 'head':head,  'pressure':pressure}
dfJ = pd.read_csv(routeCSV + str(idgroup) +'_EPANET_junctions_data.csv')

new_data = pd.DataFrame(junctions)
df = pd.concat([dfJ, new_data], ignore_index=True)

df.to_csv(routeCSV + str(idgroup) +'_EPANET_junctions_data.csv', mode='w', header=True, index=False)
```

Figura 21: Almacenamiento de los datos simulados de las Junctions en el archivo csv en modelsSim.py

En este caso, para el modelo de ejemplo con el idgroup 557 se generarán 557_Epanet_junctions_data.csv, 557_Epanet_pipes_data.csv y 557_Epanet_valves_data.csv.

```
idclient,idnode,iddevice,date,demand,head,pressure
557,10,NULL,2023-07-28 06:48:09.532671,0.0,44.35553,-0.45007
557,15,NULL,2023-07-28 06:48:09.532671,140.8173278,38.34726,28.59366
557,20,NULL,2023-07-28 06:48:09.532671,0.0,48.1584,8.8392
557,35,NULL,2023-07-28 06:48:09.532671,371.8031734,44.42247,40.61247
557,40,NULL,2023-07-28 06:48:09.532671,0.0,44.196,3.99288
557,50,NULL,2023-07-28 06:48:09.532671,0.0,42.672,7.1628
557,60,NULL,2023-07-28 06:48:09.532671,0.0,63.70645,63.70644
557,601,NULL,2023-07-28 06:48:09.532671,0.0,92.18788,92.18787
557,61,NULL,2023-07-28 06:48:09.532671,0.0,92.18788,92.18787
557,101,NULL,2023-07-28 06:48:09.532671,57.8107305,44.35553,31.55393
557,103,NULL,2023-07-28 06:48:09.532671,40.5390378,44.34608,31.23968
557,105,NULL,2023-07-28 06:48:09.532671,41.1994681,44.7536,36.0668
557,107,NULL,2023-07-28 06:48:09.532671,16.6295266,44.75164,38.04603
557,109,NULL,2023-07-28 06:48:09.532671,70.4259239,44.34624,38.1588
557,111,NULL,2023-07-28 06:48:09.532671,43.1990314,44.53408,41.48607
```

Figura 22: Datos almacenados en 557_EPANET_junctions_data.csv.

En ambos *scripts*, para aquellos modelos que sean de clientes de *Datakorum* y tengan integrados en su red nuestros dispositivos, se obtendrá el *iddevice* del dispositivo de *Datakorum* que corresponde a cada nodo de la red, relacionando así dispositivo real y nodo del modelo.

Establecer esta relación, nos permite a la hora de hacer la simulación sustituir las demandas de cada nodo por los consumos reales y actuales de su correspondiente dispositivo. Logrando así que la simulación se realice con los valores reales actuales de la red, obteniendo una simulación realista del momento actual en el que se reciben los datos del dispositivo.

8.2 Almacenamiento de los datos

Una vez realizada la simulación de todos los modelos y habiendo obtenido los csv que almacenan la simulación, así como los csv que almacenan la información de los elementos de la red, se proceden a almacenar todos estos datos en la base de datos.

Para ello se utilizan dos *scripts PHP*, uno para cada tipo de csv (información y simulación) que se encargarán de conectarse con la base de datos y realizar la instrucción "LOAD DATA", que lee las filas de los archivos csv a una gran velocidad y las va insertando en el formato correcto en las respectivas tablas.

```php
try{
    foreach ($tablesName as $key => $table) {

        $filename = $idgroup.'_'.$table.'.csv';

        $tempFile = '/var/lib/mysql-files/'.$filename;

        $file = fopen($tempFile, 'w');

        $curl = curl_init($endpoint.'/'.$filename);

        curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
        curl_setopt($curl, CURLOPT_SSL_VERIFYPEER, false);
        curl_setopt($curl, CURLOPT_FILE, $file);

        $response = curl_exec($curl);
        $error = curl_error($curl);

        curl_close($curl);
        fclose($file);

        echo "LOAD DATA INFILE '".$tempFile."' INTO TABLE baltoro.".$tablesName[$key]." FIELDS TERMINATED BY ',' ENCLOSED BY '\"' LINES TERMINATED BY '\\n' IGNORE 1 LINES (".$columnsNames[$key].")";

        $query = $db->prepare("LOAD DATA INFILE '".$tempFile."' INTO TABLE baltoro.".$tablesName[$key]." FIELDS TERMINATED BY ',' ENCLOSED BY '\"' LINES TERMINATED BY '\\n' IGNORE 1 LINES (".$columnsNames[$key].")");
        $query->execute();

        unlink($tempFile);

    }
}catch (PDOException $e) {
    // Si ocurre un error, deshace la transacción
    $db->rollBack();
    echo "Error: " . $e->getMessage();
}
```

Figura 23: Script PHP encargado de almacenar los datos.

Se realizará dicha instrucción seis veces por modelo, tres en el *script* encargado de almacenar la información y las otras tres en el *script* que se encarga de almacenar la simulación. Cada una de esas tres ejecuciones corresponden a los CSV de los distintos elementos de la red: uniones, tuberías y válvulas.

Independientemente del modelo, cada elemento se almacenará en una tabla especifica. Los nodos se guardarán en *EPANET_junctions* y *EPANET_junctions_data*, las tuberías en *EPANET_pipes* y *EPANET_pipes_data*, y las válvulas en *EPANET_valves* y *EPANET_ valves _data*.

id	idclient	idnode	date	GPS_lat	GPS_lng	type	iddevice	elevation
1330	557	10	2023-07-28 07:50:23	43.39751100098435000000	2.53297499999244600000	Junction	NULL	44.80560
1331	557	15	2023-07-28 07:50:23	42.58021489304524000000	0.83062116508504880000	Junction	NULL	9.75360
1332	557	20	2023-07-28 07:50:23	42.76695134677149000000	1.43942756401876480000	Junction	NULL	39.31920
1333	557	35	2023-07-28 07:50:23	43.51089542416616500000	0.90533332835205680000	Junction	NULL	3.81000
1334	557	40	2023-07-28 07:50:23	43.48572703523979000000	0.79323402313524330000	Junction	NULL	40.20312
1335	557	50	2023-07-28 07:50:23	43.53886808159856500000	0.17725440307306467000	Junction	NULL	35.50920
1336	557	60	2023-07-28 07:50:23	42.83439693639282000000	1.85688794127441300000	Junction	NULL	0.00000
1337	557	601	2023-07-28 07:50:23	42.88072146054374000000	1.88301415761792820000	Junction	NULL	0.00000
1338	557	61	2023-07-28 07:50:23	42.87487317017395000000	1.82623437609299090000	Junction	NULL	0.00000
1339	557	101	2023-07-28 07:50:23	43.42722627542949000000	2.06481525775365800000	Junction	NULL	12.80160
1340	557	103	2023-07-28 07:50:23	43.51641021220175500000	2.03599370991510580000	Junction	NULL	13.10640
1341	557	105	2023-07-28 07:50:23	43.38677723255971000000	1.82719769843247910000	Junction	NULL	8.68680
1342	557	107	2023-07-28 07:50:23	43.36926360478182000000	1.71411418886067860000	Junction	NULL	6.70560
1343	557	109	2023-07-28 07:50:23	43.45366184266230000000	1.68678682144291160000	Junction	NULL	6.18744
1344	557	111	2023-07-28 07:50:23	43.42166336432434000000	1.49185055421569500000	Junction	NULL	3.04800

Figura 25: Tabla bd EPANET_junctions.

id	idclient	idnode	iddevice	date	demand	head	pressure
3638	557	10	NULL	2023-07-28 07:53:08	0.0000000	44.35553	-0.45007
3639	557	15	NULL	2023-07-28 07:53:08	140.8173278	38.34726	28.59366
3640	557	20	NULL	2023-07-28 07:53:08	0.0000000	48.15840	8.83920
3641	557	35	NULL	2023-07-28 07:53:08	371.8031734	44.42247	40.61247
3642	557	40	NULL	2023-07-28 07:53:08	0.0000000	44.19600	3.99288
3643	557	50	NULL	2023-07-28 07:53:08	0.0000000	42.67200	7.16280
3644	557	60	NULL	2023-07-28 07:53:08	0.0000000	63.70645	63.70644
3645	557	601	NULL	2023-07-28 07:53:08	0.0000000	92.18788	92.18787
3646	557	61	NULL	2023-07-28 07:53:08	0.0000000	92.18788	92.18787
3647	557	101	NULL	2023-07-28 07:53:08	57.8107305	44.35553	31.55393
3648	557	103	NULL	2023-07-28 07:53:08	40.5390378	44.34608	31.23968
3649	557	105	NULL	2023-07-28 07:53:08	41.1994681	44.75360	36.06680
3650	557	107	NULL	2023-07-28 07:53:08	16.6295266	44.75164	38.04603
3651	557	109	NULL	2023-07-28 07:53:08	70.4259239	44.34624	38.15880
3652	557	111	NULL	2023-07-28 07:53:08	43.1990314	44.53408	41.48607

Figura 24: Tabla bd EPANET_junctions_data.

8.3 Visualización de la simulación

En cuanto se realizan las simulaciones y se almacenan en la base de datos, ya se puede visualizar las redes simuladas en la aplicación web. Esta es la única parte desarrollada para la aplicación web que es visible para el usuario.

Para la visualización de las simulaciones se ha realizado una vista con vue.js, en esta se cargará el modelo de red simulado correspondiente a cada usuario/cliente. Para ello al cargar la vista se realizarán varias consultas a la base de datos para obtener los parámetros deseados.

```
$junctionsData = DB::select('SELECT j.id, j.idclient, j.idnode, j.GPS_lat, j.GPS_lng, j.type, j.iddevice, jd.demand, jd.head, jd.pressure, j.elevation, jd.date
    FROM baltoro.EPANET_junctions j
    INNER JOIN baltoro.EPANET_junctions_data jd ON j.idclient = jd.idclient   AND j.idnode = jd.idnode
    WHERE jd.idclient = '.$idgroup.' AND jd.date = (SELECT MAX(date)
    FROM baltoro.EPANET_junctions_data WHERE idclient = '.$idgroup.')');

foreach($junctionsData as $junction){
    $junctions[] = $junction;
}
```

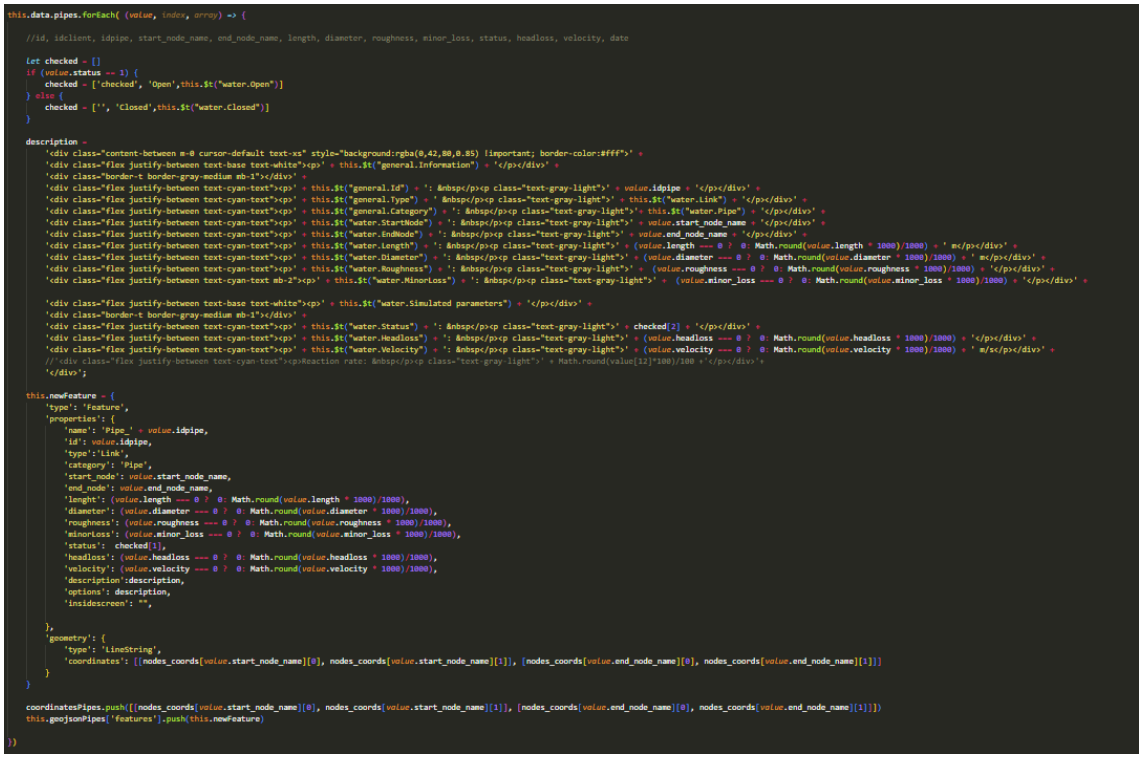
Figura 26: Query para la obtención de los parámetros de las uniones.

De la misma forma que en la figura 26 se obtienen los parámetros de las uniones, se obtendrían los parámetros de las tuberías y de las válvulas.

Tras obtener todos los datos del modelo en la vista, se procede a crear un mapa mediante la librería *mapbox*[6]. Este mapa aparecerá centrado en la localización establecida para cada cliente/usuario y tendrá un estilo diseñado expresamente para *Baltoro* mediante *CSS*.

Debido a que se va a representar el modelo en su localización geográfica real, es necesario codificar los elementos de la red en el formato *GeoJson*, formato que permite representar elementos geográficos sencillos.

[6] Mapbox proporciona la capacidad de crear mapas dinámicos, eficaces y personalizados.



```
this.data.pipes.forEach( (value, index, array) => {

    //id, idclient, idpipe, start_node_name, end_node_name, length, diameter, roughness, minor_loss, status, headloss, velocity, date

    let checked = []
    if (value.status == 1) {
        checked = ['checked', 'Open',this.$t("water.Open")]
    } else {
        checked = ['', 'Closed',this.$t("water.Closed")]
    }

    description =
        '<div class="content-between m-0 cursor-default text-xs" style="background:rgba(0,42,88,0.85) !important; border-color:#fff">' +
        '<div class="flex justify-between text-base text-white"><p>' + this.$t("general.Information") + '</p></div>' +
        '<div class="border-t border-gray-medium mb-1"></div>' +
        '<div class="flex justify-between text-cyan-text"><p>' + this.$t("general.Id") + ':  </p><p class="text-gray-light">' + value.idpipe + '</p></div>' +
        '<div class="flex justify-between text-cyan-text"><p>' + this.$t("general.Type") + '  </p><p class="text-gray-light">' + this.$t("water.Link") + '</p></div>' +
        '<div class="flex justify-between text-cyan-text"><p>' + this.$t("general.Category") + ':  </p><p class="text-gray-light">'+ this.$t("water.Pipe") + '</p></div>' +
        '<div class="flex justify-between text-cyan-text"><p>' + this.$t("water.StartNode") + ':  </p><p class="text-gray-light">' + value.start_node_name + '</p></div>' +
        '<div class="flex justify-between text-cyan-text"><p>' + this.$t("water.EndNode") + ':  </p><p class="text-gray-light">' + value.end_node_name + '</p></div>' +
        '<div class="flex justify-between text-cyan-text"><p>' + this.$t("water.Length") + ':  </p><p class="text-gray-light">' + (value.length === 0 ?  0: Math.round(value.length * 1000)/1000) + ' m</p></div>' +
        '<div class="flex justify-between text-cyan-text"><p>' + this.$t("water.Diameter") + ':  </p><p class="text-gray-light">' + (value.diameter === 0 ?  0: Math.round(value.diameter * 1000)/1000) + ' m</p></div>' +
        '<div class="flex justify-between text-cyan-text"><p>' + this.$t("water.Roughness") + ':  </p><p class="text-gray-light">' +  (value.roughness === 0 ?  0: Math.round(value.roughness * 1000)/1000) + '</p></div>' +
        '<div class="flex justify-between text-cyan-text mb-2"><p>' + this.$t("water.MinorLoss") + ':  </p><p class="text-gray-light">' +  (value.minor_loss === 0 ?  0: Math.round(value.minor_loss * 1000)/1000) + '</p></div>' +

        '<div class="flex justify-between text-base text-white"><p>' + this.$t("water.Simulated parameters") + '</p></div>' +
        '<div class="border-t border-gray-medium mb-1"></div>' +
        '<div class="flex justify-between text-cyan-text"><p>' + this.$t("water.Status") + ':  </p><p class="text-gray-light">' + checked[2] + '</p></div>' +
        '<div class="flex justify-between text-cyan-text"><p>' + this.$t("water.Headloss") + ':  </p><p class="text-gray-light">' + (value.headloss === 0 ?  0: Math.round(value.headloss * 1000)/1000) + '</p></div>' +
        '<div class="flex justify-between text-cyan-text"><p>' + this.$t("water.Velocity") + ':  </p><p class="text-gray-light">' + (value.velocity === 0 ?  0: Math.round(value.velocity * 1000)/1000) + ' m/s</p></div>' +
        //'<div class="flex justify-between text-cyan-text"><p>Reaction rate:  </p><p class="text-gray-light">' + Math.round(value[12]*100)/100 + '</p></div>'+
        '</div>';

    this.newFeature = {
        'type': 'Feature',
        'properties': {
            'name': 'Pipe_' + value.idpipe,
            'id': value.idpipe,
            'type':'Link',
            'category': 'Pipe',
            'start_node': value.start_node_name,
            'end_node': value.end_node_name,
            'lenght': (value.length === 0 ?  0: Math.round(value.length * 1000)/1000),
            'diameter': (value.diameter === 0 ?  0: Math.round(value.diameter * 1000)/1000),
            'roughness': (value.roughness === 0 ?  0: Math.round(value.roughness * 1000)/1000),
            'minorLoss': (value.minor_loss === 0 ?  0: Math.round(value.minor_loss * 1000)/1000),
            'status':  checked[1],
            'headloss': (value.headloss === 0 ?  0: Math.round(value.headloss * 1000)/1000),
            'velocity': (value.velocity === 0 ?  0: Math.round(value.velocity * 1000)/1000),
            'description':description,
            'options': description,
            'insidescreen': "",

        },
        'geometry': {
            'type': 'LineString',
            'coordinates': [[nodes_coords[value.start_node_name][0], nodes_coords[value.start_node_name][1]], [nodes_coords[value.end_node_name][0], nodes_coords[value.end_node_name][1]]]
        }
    }

    coordinatesPipes.push([[nodes_coords[value.start_node_name][0], nodes_coords[value.start_node_name][1]], [nodes_coords[value.end_node_name][0], nodes_coords[value.end_node_name][1]]])
    this.geojsonPipes['features'].push(this.newFeature)
})
```

Figura 27: Creación del objeto geojsonPipes.

Tras crear los objetos *geojson* de cada elemento de la red (gejsonJuncions, geojsonTanks, geojsonReservoirs, geojsonPipes y geojsonValves) se procede a generar una fuente de datos para el mapa de cada uno de los elementos y para cada una de estas se creará una nueva capa en el mapa.

Las capas son elementos visuales que representan datos geoespaciales de una fuente de datos. En estas se define el estilo y la forma de visualización de los elementos en el mapa. Por ejemplo, los nodos en el mapa tienen una forma circular y son de color azul, como se puede observar en la figura 28.

```
map.addSource('junctions', {
    'type': 'geojson',
    'data': this.geojsonJunctions
});

map.addLayer({
    'id': 'junctions',
    'type': 'circle',
    'source': 'junctions',
    'layout': {
        'visibility': 'visible'
    },
    'paint': {
        'circle-radius': 6,
        'circle-color': '#166EB2',
        "circle-stroke-width": 1,
        "circle-stroke-color": '#fff'
    }
});
```

Figura 28 : Fuente de datos y capa correspondiente a los nodos.

```
map.addSource('pipes', {
    'type': 'geojson',
    'data': this.geojsonPipes,
});

map.addLayer({
    'id': 'pipes',
    'type': 'line',
    'source': 'pipes',
    'layout': {
        'line-join': 'round',
        'line-cap': 'round',
        'visibility': 'visible'
    },
    'paint': {
        'line-color': '#0D03FF',
        'line-width': 3,
        'line-opacity': 0.8,
    }
});
```

Figura 29: Fuente de datos y capa correspondiente a las tuberías.

Este proceso se realiza para cada elemento y posteriormente se realiza la lógica que registra la posición de ratón, de forma que cuando se clique o el ratón se posicione dentro de cualquier elemento se muestre un pop-up con los datos de este.

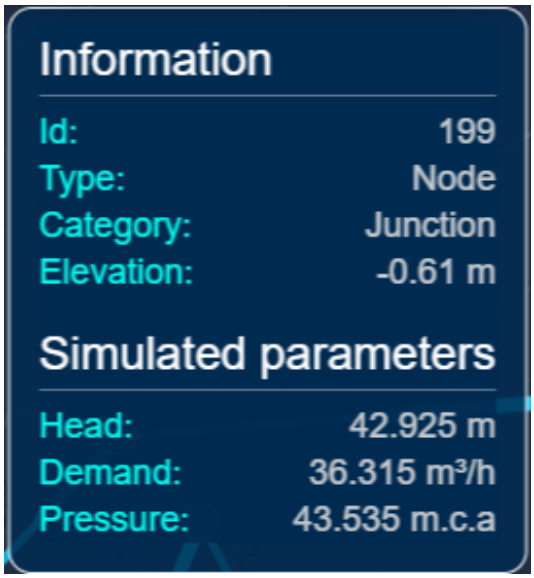


Figura 30: Pop-up correspondiente a un nodo.

Además de los elementos, ha sido necesario añadir una capa más al mapa, se trata de una animación añadida a las tuberías que representa el agua fluyendo a través de estas.

Para la velocidad de la animación se ha tenido en cuenta la longitud de cada tubería y la velocidad a la que fluye el agua a través de estas. La velocidad de animación no es realista, es decir, el tiempo que tarda en animar una tubería completa no es $t = \frac{distancia}{velocidad}$, pero cada tubería tiene una velocidad de animación que guarda relación con dicha fórmula.

```
map.addSource('pipes-animation', {
    'type': 'geojson',
    'data': this.geojson
});

map.addLayer({
    'id': 'pipes-animation',
    'type': 'line',
    'source': 'pipes-animation',
    'layout': {
        'line-cap': 'round',
        'line-join': 'round',
        'visibility': 'visible'
    },
    'paint': {
        'line-color': '#0DD3FF',
        'line-width': 5
    }
});
```

Figura 31: Fuente de datos y capa correspondiente a la animación de las tuberías.

Con la vista ya creada y todos los elementos dispuestos en el mapa, la simulación ya estaría disponible para el usuario.

En la figura 32 se muestra la red del modelo de ejemplo Net3, identificado en la aplicación con el *idclient* 557.

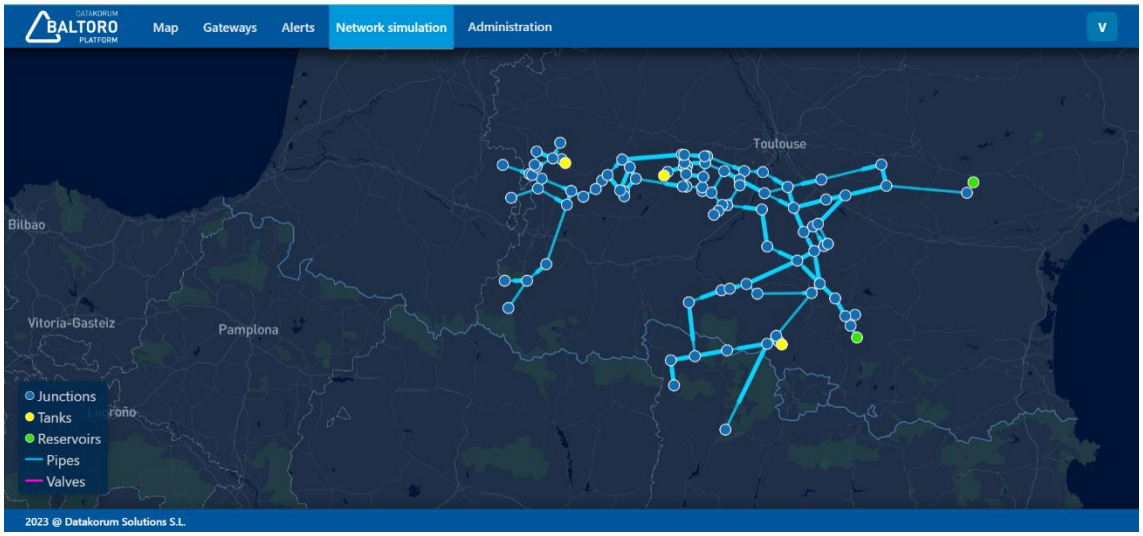


Figura 32: Red hidráulica del modelo de ejemplo Net3.

En la esquina inferior izquierda de la vista aparece una leyenda que contiene los nombres e iconos correspondientes a cada uno de los elementos. Si se pulsa en cualquier nombre de la leyenda, el correspondiente elemento desaparecerá del mapa y si se vuelve a pulsar sobre el mismo, este vuelve a aparecer.

Esto nos permite obtener una mejor visualización de la red en caso de que esta disponga de muchos elementos y la vista no sea clara.

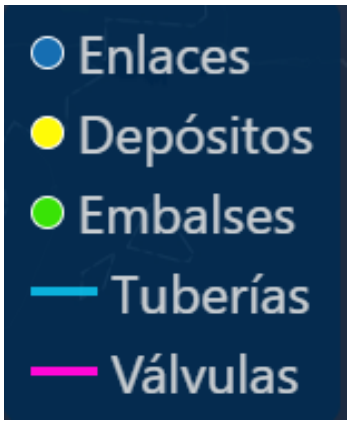


Figura 33: Leyenda de los elementos del mapa.

9. Estado actual

En este punto se va a exponer el estado actual del módulo de la aplicación, estado en el cual se va a presentar y defender el Trabajo de Fin de Grado.

Actualmente los modelos de redes de agua que tenemos proporcionados por los clientes no son todo lo precisos que deberían, lo que no nos permite realizar simulaciones realistas y precisas de sus redes. Por lo que se han utilizado los modelos de ejemplo que proporciona *WNTR* para lograr realizar simulaciones de forma más precisa.

Debido a que los modelos de redes de nuestros clientes están en constante evolución y mejora, ahora mismo no se realizan simulaciones a futuro, es decir, actualmente cogemos los últimos consumos de nuestros dispositivos, los sustituimos en las demandas iniciales de sus respectivos nodos y se realiza la simulación con un tiempo de duración igual a cero.

Este proceso se repite cada quince minutos, debido a que los dispositivos realizan envíos con dicha periodicidad. Así pues, cada cuarto de hora, tendremos una simulación nueva de cada una de las redes disponibles.

Este tipo de simulación nos permite únicamente visualizar el estado y los valores de cada punto de la red en un momento preciso, en un tiempo cero.

10. Actualizaciones futuras

Se pretende seguir avanzando en el desarrollo de este módulo ya incorporado a la aplicación web *Baltoro* y en este punto se van a comentar posibles actualizaciones que pude recibir el módulo de la aplicación realizado en este proyecto.

Una vez se consiga un modelo de red de un cliente lo más preciso posible, se pretende realizar simulaciones con un periodo de quince minutos igual que ahora, pero estas simulaciones tendrán una duración de quince minutos. De forma que cuando se visualice una simulación se pueda seleccionar el tiempo en el que se quieren ver los resultados, variando los valores de este entre $t = 0$, tiempo en que empieza la simulación y $t = 15$, tiempo el que finaliza la simulación.

También se está planteando incorporar simulaciones más prolongadas en el tiempo, aunque puede que los resultados obtenidos sean algo menos certeros.

Con estos nuevos cambios que afectarán a la simulación, se añadirán nuevas funcionalidades a la vista:

- El usuario podrá elegir el tiempo en el que desea ver la simulación.
- Cada elemento dispondrá de gráficas de tiempo específicas, en las cuales se podrá observar la variación del valor de ciertos parámetros. Por ejemplo, las uniones tendrán una gráfica para observar como varía la presión a lo largo de la simulación.
- Un buscador en la vista, de forma que se pueda buscar los dispositivos en la red a través de su identificador *iddevice*.

Todos estos cambios añaden todavía más funcionalidad a la aplicación y permiten al usuario tener un control más amplio de su red, ya que además de tener simulaciones temporales va a disponer de gráficos en los cuales ver la evolución de ciertos parámetros.

11.Conclusiones

Una vez finalizado el desarrollo del trabajo podemos extraer una serie de conclusiones.

En primer lugar, en cuanto a los objetivos marcados antes de empezar con el proyecto podemos decir que estos se han cumplido de forma satisfactoria, ya que he conseguido implementar un nuevo módulo funcional a la aplicación web *Baltoro*. Este nuevo módulo cumple con todos los requisitos establecidos como objetivos, es decir, actualmente este módulo es capaz de simular redes hidráulicas de manera automática y de mostrar estas simulaciones al usuario.

Respecto a las metas personales, afirmar con seguridad que también se han alcanzado con éxito, ya que he conseguido afianzar conceptos académicos debido a la utilización de estos durante el proyecto, además, he aprendido a utilizar nuevas herramientas y lenguajes de programación que serán de utilidad para mi futuro profesional.

Es importante destacar que la mayoría de las implementaciones comentadas en el punto anterior se van a introducir gradualmente en *Baltoro*, esto asegura que haya una continuación del desarrollo de este trabajo. De hecho, por parte de *Datakorum* se me ha comunicado que quieren que forme parte de este futuro, por lo que yo mismo voy a estar mejorando el desarrollo de este módulo de la aplicación.

En resumen, me siento satisfecho con el trabajo de realizado ya que los objetivos se han cumplido con creces y con los conocimientos que ha aprendido a lo largo del desarrollo de este proyecto.

12.ODS

Los Objetivos de Desarrollo Sostenible [14], son un conjunto de 17 objetivos adoptados por todos los miembros de las Naciones unidas en 2015. Estos objetivos forman parte de la Agenda 2030 para el Desarrollo Sostenible, que es un plan de acción global destinado a erradicar la pobreza, proteger el planeta y asegurar la prosperidad para el año 2030.

Los ODS abarcan una amplia gama de áreas clave sobre las que actuar, incluyen la erradicación de la pobreza, la igualdad de género, la acción climática, la educación de calidad, la paz y la justicia entre otros.

Figura 34: Objetivos de Desarrollo Sostenible desarrollados por las Naciones Unidas.

Todos estos objetivos están integrados, la acción en un área afectara a los resultados de las otras áreas. La idea inicial de los ODS es que con la colaboración de países, ciudadanos y empresas se logre un mundo más sostenible y justo en todos los ámbitos.

A continuación, se van a describir aquellos objetivos que guardan relación con este trabajo de fin de grado:

- 6. **Agua limpia y saneamiento**: El objetivo es asegurar el acceso a servicios de saneamiento adecuados y equitativos y mejorar la calidad del agua mediante la reducción de la contaminación.
    - Mediante las simulaciones de redes de distribución se pueden obtener parámetros que indican el grado de contaminación del agua, pudiendo saber cómo van a afectar posibles modificaciones de la red a la calidad del agua. Además, mediante las simulaciones y los datos de consumo de la red, se pueden realizar estudios sobre modificaciones de parámetros y elementos de la red para evitar sobrecostes y proporcionar a cada punto de la red solo la demanda necesaria.
- 9. **Industria, innovación e infraestructuras**: El objetivo es promover la industrialización sostenible y fomentar la innovación para lograr un desarrollo económico inclusivo y sostenible.
    - El nuevo módulo implementado en *Baltoro* permite simular modelos de redes de agua de forma dinámica y automática, lo que permite al cliente tener información y control sobre toda su red de agua.
- 12**: Producción y consumo sostenibles**: Tiene como objetivo fomentar la eficiencia en el uso de recursos y energía en los procesos de producción y promover prácticas de consumo sostenible.
    - Como se ha comentado a lo largo del proyecto, mediante los dispositivos de *Datakorum* se pueden monitorizar los consumos de agua actuales y con las simulaciones se puede obtener una aproximación de los consumos futuros, lo que permite saber en que puntos de la red está habiendo un consumo excesivo, ya sea por pérdidas y problemas en la red o por un mal uso del agua. Controlando estos puntos de exceso consumo se pueden reducir costes y el desperdicio de agua.

13.Anexo

13.1 Índice de figuras

13.2 Índice de tablas

14. Bibliografía

[1] Datakorum Solutions S.L (2018). https://www.datakorum.com/site/.

[2] Coppola, M. HubSpot, (2022/2023). "Frontend y backend: qué son y en qué se diferencian y ejemplos". https://blog.hubspot.es/website/frontend-y-backend.

[3] Coppola, M. HubSpot, (2023). "Qué es JavaScript, para que sirve y cómo funciona". https://blog.hubspot.es/website/que-es-javascript#que-es.

[4] (s.f). "¿Qué es Vue.js?". https://es.vuejs.org/v2/guide/.

[5] Mdn web docs. "Tecnología para desarrolladores web – HTML: Lenguaje de etiquetas de hipertexto". (2023). https://developer.mozilla.org/es/docs/Web/HTML.

[6] Mdn web docs. "Tecnología para desarrolladores web - CSS". (2023). https://developer.mozilla.org/es/docs/Web/CSS.

[7] Epitech Spain. (2021). "Qué es PHP y para qué sirve este lenguaje de código abierto". https://www.epitech-it.es/que-es-php/.

[8] Atube Vera, R. OpenWebinars (2021). "Qué es Laravel: Características y ventajas" https://openwebinars.net/blog/que-es-laravel-caracteristicas-y-ventajas/

[9] Amazon Web Services. (s.f). "¿Qué es Python?". https://aws.amazon.com/es/what-is/python/.

[10] Alejandro. DesdeLinux. (2019). "Cron y crontab, explicados". https://blog.desdelinux.net/cron-crontab-explicados/.

[11] Instituto de ingeniería del Agua y Medio ambiente. (s.f). "Programa para la simulación hidráulica y de calidad del agua en redes de distribución de agua a presión". https://www.iiama.upv.es/iiama/es/transferencia/software/epanet-esp.html.

[12] Rossman, L.A. (s.f). "EPANET 2 MANUAL DE USUARIO". http://www.instagua.upv.es/epanet/descargas/ManualEPANETv2E.pdf.

[13] Water Network Tool for Resilience. (s.f). https://wntr.readthedocs.io/en/stable/index.html#.

[14] Organización de las Naciones Unidas. (s.f). "La Agenda para el Desarrollo Sostenible". https://www.un.org/sustainabledevelopment/es/development-agenda/.