UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Diseño e implementación de una aplicación Web para la visualización de datos provenientes de Ensayos de Respuesta Térmica

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: José Vicente Hurtado Gimeno

Tutores: Lenin Guillermo Lemus Zúñiga

Miguel Ángel Mateo Pla

Curso 2016 - 2017

Resumen

Un pozo de intercambio de calor (*Borehole heat exchanger,* BHE inglés) es una instalación geotérmica que genera una gran cantidad de información. Este volumen de información requiere un sistema informático para su almacenamiento y presentación. En este trabajo se diseña una aplicación Web y una Web Api para dar solución a los dos problemas anteriores y facilitar el análisis y estudio de la información obtenida.

Palabras clave: Pozo Térmico, ensayos de Respuesta Térmica, TRT, eficiencia energética, geotérmica, Web Api, MongoDB, Angular.

Abstract

A *Borehole heat exchanger* is a thermical installation who generate a lot of information. This information needs a computer system to save and present the data. In this project we designed a Web application and a Web Api to solve the two problems and make more easier to analyze.

Key Words: *Borehole heat exchanger*, Thermal Response Test, energy efficiency, geothermal, Web Api, MongoDB, Angular.

A mis padres, amigos, profesores y especialmente a Chelo.....

Bendita paciencia.

Tabla de contenidos

1 Introducción 7
1.1 Objetivos 10
2 Análisis 11
3 Diseño - Implementación 15
3.1 Pasarela 15
3.1.1 Estructura de la aplicación. 16
3.1.2 Librerías utilizadas. 17
3.1.3 Formulario de la aplicación 18
3.2 Web Api MongoDB – NodeJS 19
3.2.1 Estructura de la Web Api con Mongoose 20
3.2.2 Estructura de la Web Api con MongoDB 20
3.2.3 Librerías utilizadas. 21
3.2.4 Peticiones Web Api 24
3.3 Web Api SQL Server - .NET 25
3.3.1 Estructura de la Web Api 25
3.3.2 Librerías Utilizadas. 26
3.3.3 Controladores Web Api 28
3.3.4 Modelos de simulación. 30
3.4 Aplicación Web 31
3.4.1 Estructura de la Aplicación. 32
3.4.2 Librerías Utilizadas 35
3.4.3 Páginas y Gráficos. 36
4 Pruebas 41
Conclusiones 42
Bibliografía 44

1 Introducción

Los pozos de intercambio de calor [1] [2] [3] (*borehole heat exchanger* o BHE en inglés) son instalaciones geotérmicas, que persiguen mejorar la eficiencia energética de los sistemas de climatización. Para el correcto diseño de estos sistemas es necesario conocer las características del terreno. Existen dos métodos para determinar estas características.

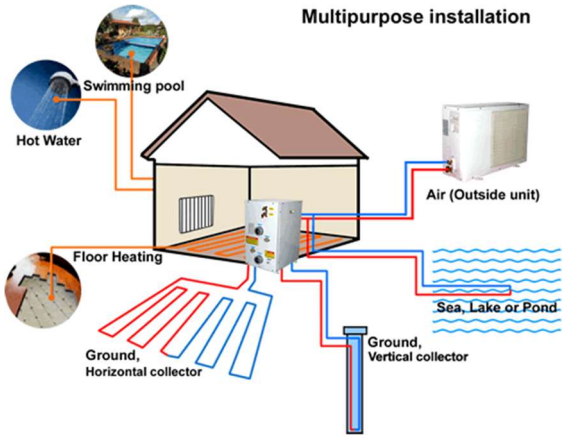

Fig. 1 Instalación multipropósito.

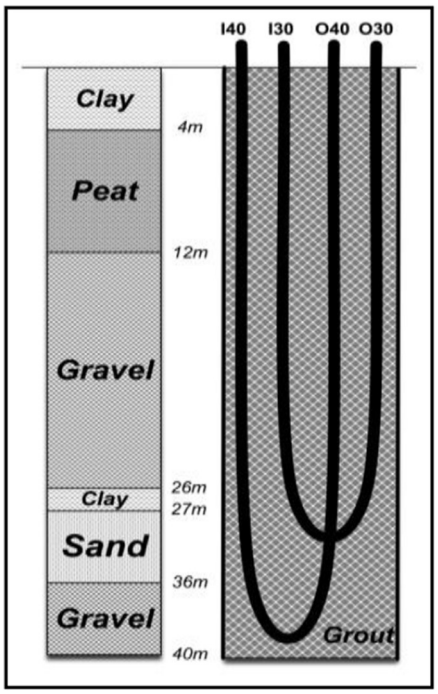

Fig. 2 Profundidad Pozo

El primero de estos métodos se basa en el análisis de catas del terreno en laboratorios especializados. La extracción de estas catas es compleja y cara, como también es muy costoso el análisis.

El segundo método se basa en la realización de test de respuesta térmica (*Thermal Response Test* o TRT en inglés). Un TRT se realiza creando un BHE más pequeño y barato que el que se supone se necesita. Sobre esa instalación a escala, se realiza un ensayo controlado en el que se inyecta un fluido calentado y se mide la temperatura de salida del mismo. Conociendo las temperaturas de entrada y salida al BHE, el caudal y las dimensiones del BHE, se pueden estimar parámetros importantes del subsuelo que se pueden utilizar para el diseño del resto de la instalación geotérmica.

Los valores obtenidos por un ensayo son capturados a través de un controlador lógico programable PLC (*Programmable Logic Controller* o PLC en inglés) instalado en el BHE. Este PLC almacena en un fichero la información medida en el ensayo. Al estar almacenada la información en un fichero esta debe ser tratada, formateada y almacenada en un SGBD (Sistema gestor de Base de Datos), para su posterior consulta, estudio y comparación con modelos de simulación obtenidos a partir de los datos del PLC.

Con los datos obtenidos, se hace necesario, su tratamiento, almacenamiento y posterior muestra a través de una aplicación Web. Con ello conseguiremos que los estudios realizados con los datos obtenidos del PLC y de los modelos de simulación se haga una interpretación mucho más cómoda.

En nuestro caso hemos obtenido los datos de prueba del TRT instalado en el Campus de Vera de la UPV, que está enfrente del edificio 8H.

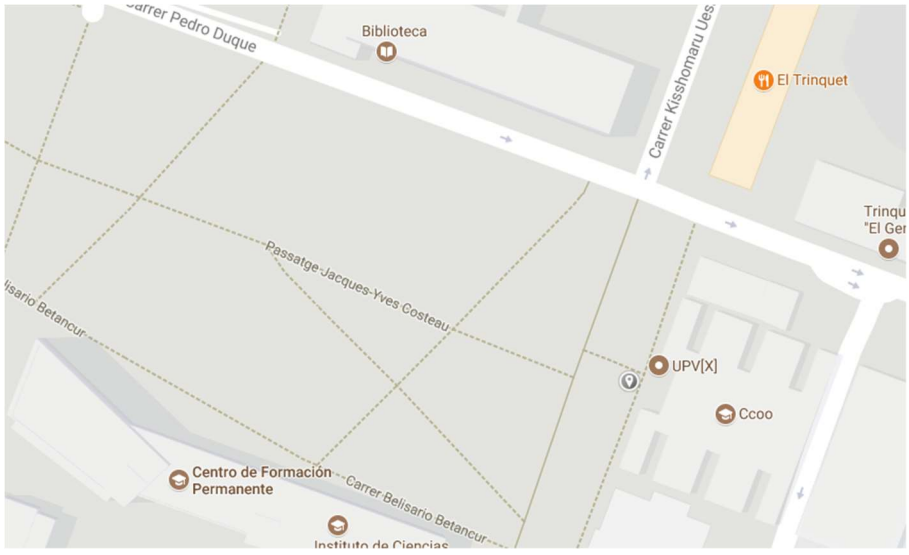

Fig. 3 Ubicación TRT Campus de Vera.

Fig. 4 Instalación Campus de Vera

Fig. 5 Instalación Campus de Vera Fontanería.

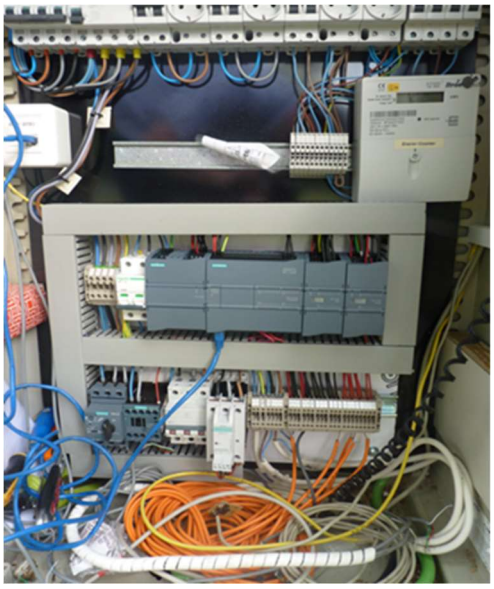

Fig. 6 Instalación Campus de Vera PLC.

1.1 Objetivos

Los objetivos principales de este TFG son:

1. Entender cómo funciona un sistema para realizar ensayos de respuesta térmica.
2. Entender el funcionamiento del sistema ERT instalado en el campus de Vera.
3. Tratar la información obtenida del PLC.
4. Almacenar los datos en un SGBD.
5. Crear un servicio Web para obtener los datos almacenados en el SGBD.
6. Diseñar una página Web que muestre en tiempo real información importante de la instalación usando el servicio web desarrollado en 3
7. Implementar modelos de simulación del comportamiento de un BHE..
8. Dibujar gráficas que permitan comparar los datos de la instalación con los modelos desarrollados.

El recorrido que se describe en el presente documento es el siguiente.

En un primer lugar daremos una definición de lo que es un TRT y un BHE, con esta definición describiremos el problema planteado, descrito anteriormente.

Posteriormente daremos un análisis al problema planteado y de qué forma vamos a solucionarlo.

Describiremos la tecnología utilizada para llevar a cabo la solución, y mostraremos tanto trozos de código de la implementación, como páginas de la propia aplicación.

Más tarde veremos los datos utilizados para las pruebas, estos mismos datos son los que se ven reflejados en las capturas de la propia aplicación.

Finalizaremos con las conclusiones del proyecto, así como una visión personal del mismo.

2 Análisis

Para poder facilitar el análisis en los estudios de los TRT´s, debemos facilitar al usuario una herramienta para la captura de los datos obtenidos de las lecturas del PLC, su automatización y su posterior presentación.

Partiendo del fichero en el cual el PLC va escribiendo sus resultados, iniciaremos el análisis de la solución que le vamos a dar al problema.

Este fichero escrito por el PLC, es un fichero de texto plano en formato CSV [4] (Comma-Separated Values en inglés), en el cual encontraremos un identificador de la prueba, fecha en la cual se realizó, hora en la que se realizó, temperatura de entrada del agua, temperatura de salida del agua, temperatura ambiente, temperatura del terreno, caudal y un valor de control correspondiente al valor proporcional a la potencia con la que se está inyectando el agua.

```
Record,Date,UTC Time,IDA1,IDA2,RET1,RET2,AMB,TERR,PRES,CAUDAL,ACC_CONTROL
     1, 4/20/2017, 9:12:08,  2.115596E+01,  2.110532E+01,  2.097240E+01,  2.101671E+01,  1.737341E+01,  2.186918E+01,  3.277271E+00,  3.428819E-01,  0.000000E+00
     2, 4/20/2017, 9:15:08,  2.116862E+01,  2.109899E+01,  2.095975E+01,  2.101671E+01,  1.777127E+01,  2.184658E+01,  3.280527E+00,  3.404948E-01,  0.000000E+00
     3, 4/20/2017, 9:18:08,  2.117495E+01,  2.109899E+01,  2.096607E+01,  2.102937E+01,  1.774233E+01,  2.186353E+01,  3.271846E+00,  3.404948E-01,  0.000000E+00
     4, 4/20/2017, 9:21:08,  2.116862E+01,  2.109899E+01,  2.096607E+01,  2.101671E+01,  1.763744E+01,  2.189742E+01,  3.274016E+00,  3.404948E-01,  0.000000E+00
     5, 4/20/2017, 9:24:08,  2.117495E+01,  2.111798E+01,  2.095975E+01,  2.100405E+01,  1.763744E+01,  2.186353E+01,  3.276910E+00,  3.404948E-01,  0.000000E+00
     6, 4/20/2017, 9:27:08,  2.116862E+01,  2.112431E+01,  2.097240E+01,  2.100405E+01,  1.777850E+01,  2.190307E+01,  3.276186E+00,  3.404948E-01,  0.000000E+00
     7, 4/20/2017, 9:30:08,  2.118761E+01,  2.110532E+01,  2.096607E+01,  2.101671E+01,  1.747830E+01,  2.189742E+01,  3.283420E+00,  3.404948E-01,  0.000000E+00
     8, 4/20/2017, 9:33:08,  2.117495E+01,  2.111166E+01,  2.097240E+01,  2.101671E+01,  1.733001E+01,  2.190307E+01,  3.283420E+00,  3.404948E-01,  0.000000E+00
     9, 4/20/2017, 9:36:08,  2.118128E+01,  2.113697E+01,  2.097873E+01,  2.102304E+01,  1.719256E+01,  2.189742E+01,  3.277271E+00,  3.404948E-01,  0.000000E+00
    10, 4/20/2017, 9:39:08,  2.118128E+01,  2.112431E+01,  2.097240E+01,  2.102304E+01,  1.709129E+01,  2.186353E+01,  3.280527E+00,  3.404948E-01,  0.000000E+00
    11, 4/20/2017, 9:42:08,  2.118128E+01,  2.113065E+01,  2.098506E+01,  2.102937E+01,  1.709129E+01,  2.189178E+01,  3.286314E+00,  3.404948E-01,  0.000000E+00
    12, 4/20/2017, 9:45:08,  2.119394E+01,  2.113697E+01,  2.097873E+01,  2.102304E+01,  1.696470E+01,  2.186353E+01,  3.282697E+00,  3.404948E-01,  0.000000E+00
    13, 4/20/2017, 9:48:08,  2.119394E+01,  2.112431E+01,  2.097240E+01,  2.101671E+01,  1.698640E+01,  2.189178E+01,  3.283058E+00,  3.404948E-01,  0.000000E+00
    14, 4/20/2017, 9:51:08,  2.119394E+01,  2.112431E+01,  2.097873E+01,  2.103570E+01,  1.716724E+01,  2.189742E+01,  3.278718E+00,  3.404948E-01,  0.000000E+00
    15, 4/20/2017, 9:54:08,  2.118761E+01,  2.112431E+01,  2.097240E+01,  2.103570E+01,  1.733001E+01,  2.190307E+01,  3.273654E+00,  3.404948E-01,  0.000000E+00
```

Fig. 7 Fichero CSV del PLC.

Para poderle dar un uso a estos datos sin pérdida de información de los mismos, necesitamos formatearlos. Sobre todo, en el caso de la fecha y la hora, ya que según el SGBD en el que decidamos guardar la información, el propio gestor lo almacenara de una forma u otra. También es interesante fusionar la propia fecha con la hora, para facilitar filtros en un futuro sobre solo un campo de la tabla del SGBD y no sobre dos, que es como esta en el fichero original del PLC.

	Record	Date	UTC Time
1	1	2017-04-20 09:12:08.000	09:12:08
2	2	2017-04-20 09:15:08.000	09:15:08
3	3	2017-04-20 09:18:08.000	09:18:08

Fig. 8 Fecha y hora de la prueba fusionadas.

Una vez tengamos la información formateada, necesitaremos llevar los datos al SGBD, para así tenerlos automatizados para su posterior consulta. Este paso se podría realizar de forma automática importándolos con las distintas herramientas existentes, pero como

hemos explicado anteriormente, corremos el riesgo de pérdida de información de los mismos.

Con los datos almacenados, formateados y automatizados. Necesitamos dar una herramienta de acceso a los mismos. Hay múltiples formas de llevarlo a cabo, Procedimientos Almacenados (*Stored Procedures* o *SP* en inglés) [5] , Web Apis [6], etc..

Teniendo implementado este acceso a los datos, ya podremos desarrollar la aplicación Web encargada de mostrarlos, pintarlos en diferentes gráficas, así como filtrarlos de forma sencilla.

La solución que nosotros aportamos al problema anteriormente descrito se ve reflejada en la siguiente imagen en forma de esquema, que a continuación detallaremos con más detalle.

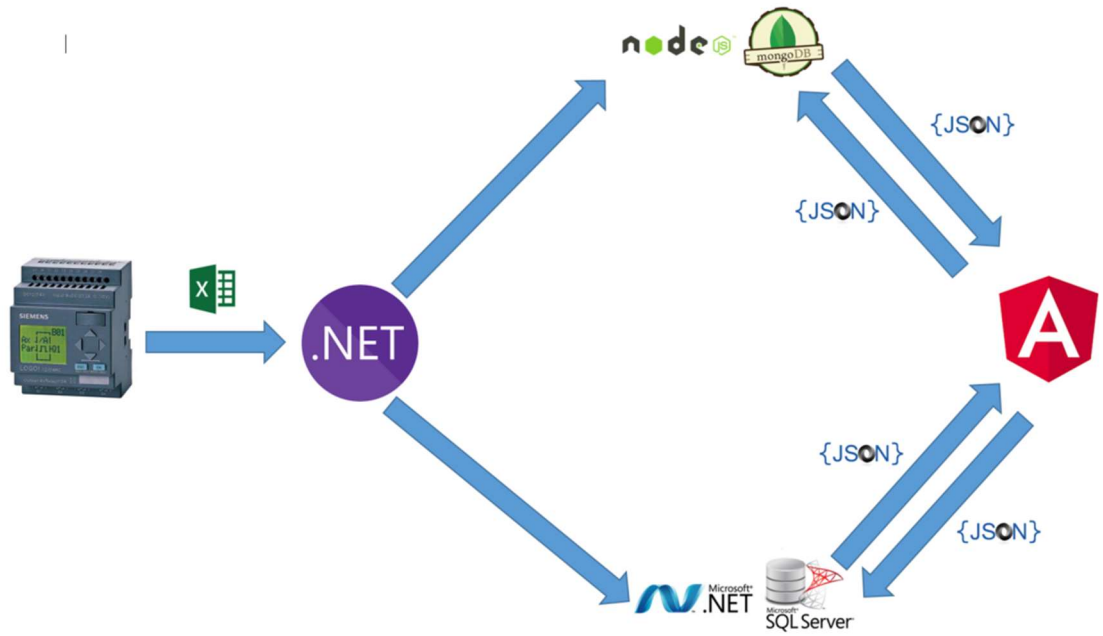

Fig. 9 Esquema de la aplicación.

En un primer lugar y conforme hemos explicado, necesitamos darle un formato a la información contenida en el CSV generado por el PLC. Para ello hemos decidido en una primera instancia, importar el documento a un fichero Excel. Gracias a ello los datos mantienen la misma información. La posterior importación nos resultara mucho más sencilla con una de las múltiples librerías existentes en el mercado de acceso a Excel.

Record	Date	UTC Time	IDA1	IDA2	RET1	RET2	AMB	TERR	PRES	CAUDAL	ACC_CONTROL
1	20/04/2017	9:12:08	2,12E+01	2,11E+01	2,10E+01	2,10E+01	1,74E+01	2,19E+01	3,28E+00	3,43E-01	0,00E+00
2	20/04/2017	9:15:08	2,12E+01	2,11E+01	2,10E+01	2,10E+01	1,78E+01	2,18E+01	3,28E+00	3,40E-01	0,00E+00
3	20/04/2017	9:18:08	2,12E+01	2,11E+01	2,10E+01	2,10E+01	1,77E+01	2,19E+01	3,27E+00	3,40E-01	0,00E+00
4	20/04/2017	9:21:08	2,12E+01	2,11E+01	2,10E+01	2,10E+01	1,76E+01	2,19E+01	3,27E+00	3,40E-01	0,00E+00
5	20/04/2017	9:24:08	2,12E+01	2,11E+01	2,10E+01	2,10E+01	1,76E+01	2,19E+01	3,28E+00	3,40E-01	0,00E+00
6	20/04/2017	9:27:08	2,12E+01	2,11E+01	2,10E+01	2,10E+01	1,78E+01	2,19E+01	3,28E+00	3,40E-01	0,00E+00
7	20/04/2017	9:30:08	2,12E+01	2,11E+01	2,10E+01	2,10E+01	1,75E+01	2,19E+01	3,28E+00	3,40E-01	0,00E+00
8	20/04/2017	9:33:08	2,12E+01	2,11E+01	2,10E+01	2,10E+01	1,73E+01	2,19E+01	3,28E+00	3,40E-01	0,00E+00
9	20/04/2017	9:36:08	2,12E+01	2,11E+01	2,10E+01	2,10E+01	1,72E+01	2,19E+01	3,28E+00	3,40E-01	0,00E+00

Fig. 10 Datos importados a Excel

Con el documento Excel generado y a través de una Pasarela, el logo de .NET morado en el esquema, insertaremos los datos en los distintos SGBD que hayamos decidido utilizar. Esta pasarela mantendrá la información de los mismos como la fusión de la fecha y la hora. Hay que vigilar en este caso en concreto y definir bien los campos de cada una de las tablas de la base de datos. Se puede dar el caso que decidamos coger como tipo de campo TimeSpan para la hora y que al mínimo problema de perdida de información de los datos los descarte y nos inserte como valores null en este campo.

La pasarela que nosotros hemos implementado inserta los datos en una base de datos NoSQL [7] como es MongoDB [8], y en una base datos relacional como es Microsoft SQL Server [9]. El porqué de implementar dos bases de datos diferentes se debe a querer dar una doble solución al acceso a los datos. Con la primera se da una solución de código abierto y con la segunda se da una solución más comercial.

	Record	Date	UTC Time	IDA1	IDA2	RET1	RET2	AMB	TERR	PRES	CAUDAL	ACC_CONTROL
1	1	2017-04-20 09:12:08.000	09:12:08	21,15596	21,10532	20,9724	21,01671	17,37341	21,86918	3,277271	0,3428819	0
2	2	2017-04-20 09:15:08.000	09:15:08	21,16862	21,09899	20,95975	21,01671	17,77127	21,84658	3,280527	0,3404948	0
3	3	2017-04-20 09:18:08.000	09:18:08	21,17495	21,09899	20,96607	21,02937	17,74233	21,86353	3,271846	0,3404948	0
4	4	2017-04-20 09:21:08.000	09:21:08	21,16862	21,09899	20,96607	21,01671	17,63744	21,89742	3,274016	0,3404948	0
5	5	2017-04-20 09:24:08.000	09:24:08	21,17495	21,11798	20,95975	21,00405	17,63744	21,86353	3,27691	0,3404948	0
6	6	2017-04-20 09:27:08.000	09:27:08	21,16862	21,12431	20,9724	21,00405	17,7785	21,90307	3,276186	0,3404948	0
7	7	2017-04-20 09:30:08.000	09:30:08	21,18761	21,10532	20,96607	21,01671	17,4783	21,89742	3,28342	0,3404948	0
8	8	2017-04-20 09:33:08.000	09:33:08	21,17495	21,11166	20,9724	21,01671	17,33001	21,90307	3,28342	0,3404948	0
9	9	2017-04-20 09:36:08.000	09:36:08	21,18128	21,13697	20,97873	21,02304	17,19256	21,89742	3,277271	0,3404948	0
10	10	2017-04-20 09:39:08.000	09:39:08	21,18128	21,12431	20,9724	21,02304	17,09129	21,86353	3,280527	0,3404948	0

Fig. 11 Datos Pruebas SQL Server

Una vez tengamos los datos almacenados, para poder acceder a ellos necesitaremos implementar una Web Api para cada uno de los SGBD, en los que hemos decidido almacenar los datos. El primero de ellos lo hemos implementado con NodeJS [10], logo de NodeJS junto con el de MongoDB en el esquema, y el segundo con .NET, log de .NET junto con el de SQLServer en el esquema. El porqué de hacerlo de esta forma, a parte del mismo motivo que hemos decidido con los SGBD, en este caso y con toda la documentación existente tanto para NodeJS - MongoDB como para, SQLServer - .NET [11], nos parecía complicar en demasía la solución al problema.

Una vez ya tengamos implementado el Web Api de acceso a los datos, implementaremos la aplicación en la cual mostraremos los datos obtenidos del PLC a través de diferentes gráficas. Para la implementación de la Aplicación Web, hemos optado por Angular 2 [12],

para que la comunicación con los distintos Web Apis no afecte al funcionamiento de la aplicación, se realizara en el formato JSON [13].

3 Diseño - Implementación

Como hemos explicado anteriormente en el análisis del problema, vamos a explicar la solución que hemos aportado con mayor detalle. Hemos optado por una solución multiplataforma, a excepción de la pasarela, para poder trasladar la aplicación al entorno de producción que más nos interese.

3.1 Pasarela

Para poder llevar la información desde el fichero CSV que rellena el PLC, hasta los distintos SGBD implementados, deberemos realizar dos pasos previos. Un primero, que se tendrá que realizar a manualmente, y se trata de trasladar este fichero CSV a un fichero Microsoft Excel. Como hemos explicado antes, no queremos perder calidad de la información extraída del PLC, así que será necesario hacer este paso previo.

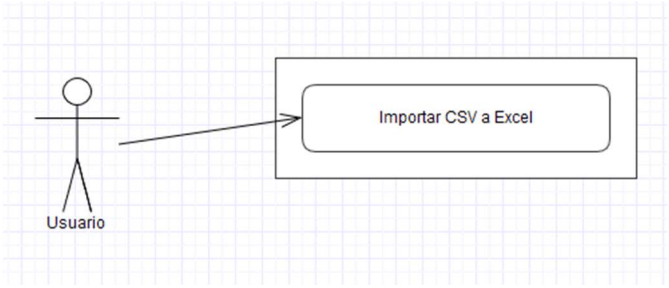

Fig. 12 Caso de Uso Importación Excel

Un segundo paso que consistirá en un aplicación, la cual hemos implementado basándonos en el la librería de Microsoft .NET [14] WinForms.

Fig. 13 Pasarela Windows Forms

En esta pasarela seleccionaremos el fichero Excel ya formateado que queremos almacenar en los distintos SGBD. Una vez cargado el fichero seleccionado pulsaremos

sobre que SGBD, en nuestro caso MongoDB o SQL Server, en el cual queremos guardar los datos. Es importante recordar que una vez actualicemos los datos con los del fichero que acabamos de cargar, se perderán todos los datos anteriormente guardados.

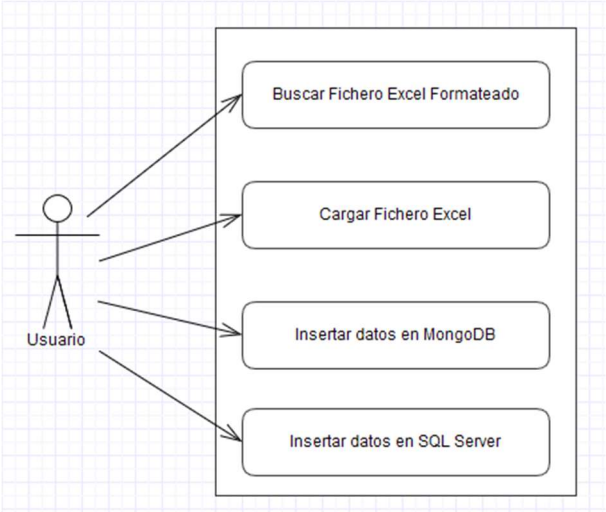

Fig. 14 Caso de uso Pasarela

3.1.1 Estructura de la aplicación.

La estructura de la aplicación está basada en dos proyectos de Visual Studio. En el primero contendrá las entidades utilizadas para la conversión de la información contenida en el documento Excel a los distintos SGBD (TFG.PozoTermico.Entity). Este proyecto será un proyecto de clases de Visual Studio [15].

El segundo proyecto contendrá las clases necesarias para mostrar los datos formateados así como el formulario utilizado por el usuario para seleccionar el documento CSV y convertirlo al SGBD seleccionado (TFG.PozoTermico.Excel.Win).

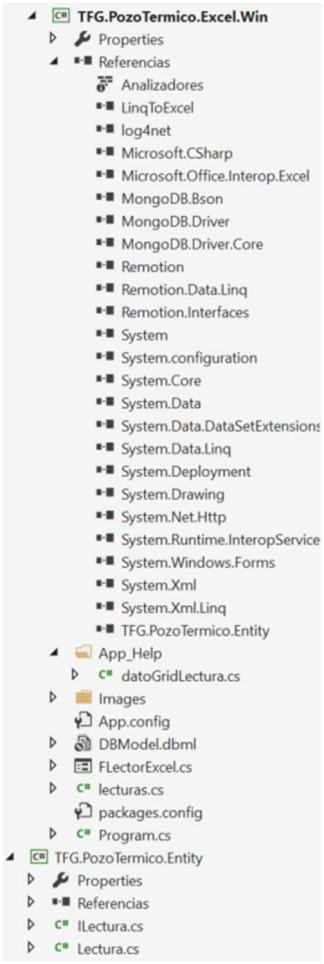

Fig. 15 Estructura de la Pasarela

3.1.2 Librerías utilizadas.

Para extraer los datos del documento Excel formateado proveniente del PLC, utilizaremos una librería para extraerlos de forma cómoda y sencilla.

En el caso que vayamos a insertar los datos en SQL Server, no necesitaremos ninguna librería, ya que .NET nos lo da de forma nativa.

Si vamos a insertar los datos en la base de datos MongoDB necesitaremos instalar un par de librerías que se encarguen de ello, y nos den una funcionalidad parecida a la que nos daría Microsoft ADO.NET [16] de forma nativa.

Tanto la librería de Excel como la librería de MongoDB, las hemos instalado en el proyecto utilizando el gestor de paquetes nativo de Visual Studio, NuGet [17].

3.1.2.1 *Linq to Excel*

Librería que accede a un fichero Excel indicándole la ruta donde se encuentra el fichero, y acceso con Linq [18] a los datos del documento. Como podemos observar con una simple consulta Linq podemos consultar la información leída del documento [19].

```
var documentoExcel = new
ExcelQueryFactory(ofdSeleccionarExcel.FileName);
var lecturaExcel = (from row in documentoExcel.Worksheet(hoja)
let item = new datoGridLectura
                    {
                        ID = row["Record"].Cast<int>(),
                  FechaLectura=row["Date"].Cast<DateTime>().AddTicks(Ti
            meSpan.Parse(row["UTC Time"].Cast<string>()).Ticks),
                        HoraLectura = TimeSpan.Parse(row["UTC
Time"].Cast<string>()),
                  ida1 = row["IDA1"].Cast<double>(),
                        ida2 = row["IDA2"].Cast<double>(),
                        ret1 = row["RET1"].Cast<double>(),
                        ret2 = row["RET2"].Cast<double>(),
                        TemperaturaAmbiente = row["AMB"].Cast<double>(),
                        TemperaturaTerreno = row["TERR"].Cast<double>(),
                        Presion = row["PRES"].Cast<double>(),
                        Caudal = row["CAUDAL"].Cast<double>(),
                        ACC_Control = row["PERC_CONTROL"].Cast<double>()
                        }
select item).ToList();
documentoExcel.Dispose();
```

3.1.2.2 *MongoDB Driver*

Esta librería realiza una conexión con la base de datos que vamos a utilizar, y nos da la posibilidad de generar, borrar, insertar, etc... tanto las distintas bases de datos que contiene, como las tablas que están dentro de las bases de datos [20]. Como hemos explicado anteriormente, la forma de utilizar esta librería es muy parecida a la de Microsoft ADO.NET.

```
this.StrConexionMongoDB =
ConfigurationManager.AppSettings["conexionMongoDb"];
_clienteMongo = new MongoClient(this.StrConexionMongoDB);
this.dbModel = new DBModelDataContext();
```

En nuestro caso la configuración de la base de datos, la tendremos almacenada en el fichero de configuración App.config de la aplicación “conexionMongoDb”.

3.1.3 Formulario de la aplicación

El formulario implementado dentro de la aplicación de la Pasarela es muy sencillo de utilizar y no contiene ninguna complicación añadida al usuario.

Primero el usuario debe seleccionar un Fichero Excel, con los datos importados del CSV, una vez cargados estos datos y si no hay ningún error en la importación, nos aparecerán en la tabla central del formulario los datos anteriormente descritos en el análisis. También nos aparecerán activos los dos botones de los diferentes SGBD, a los cuales podemos hacer la carga.

Si pulsamos el botón de MongoDB, la aplicación nos avisara que los datos cargados actualmente en la Base de datos se perderán y se almacenaran los que tenemos en ese momento en pantalla.

En el caso de que pulsemos el botón de SQL Server, el funcionamiento es el mismo que en el caso de MongoDB.

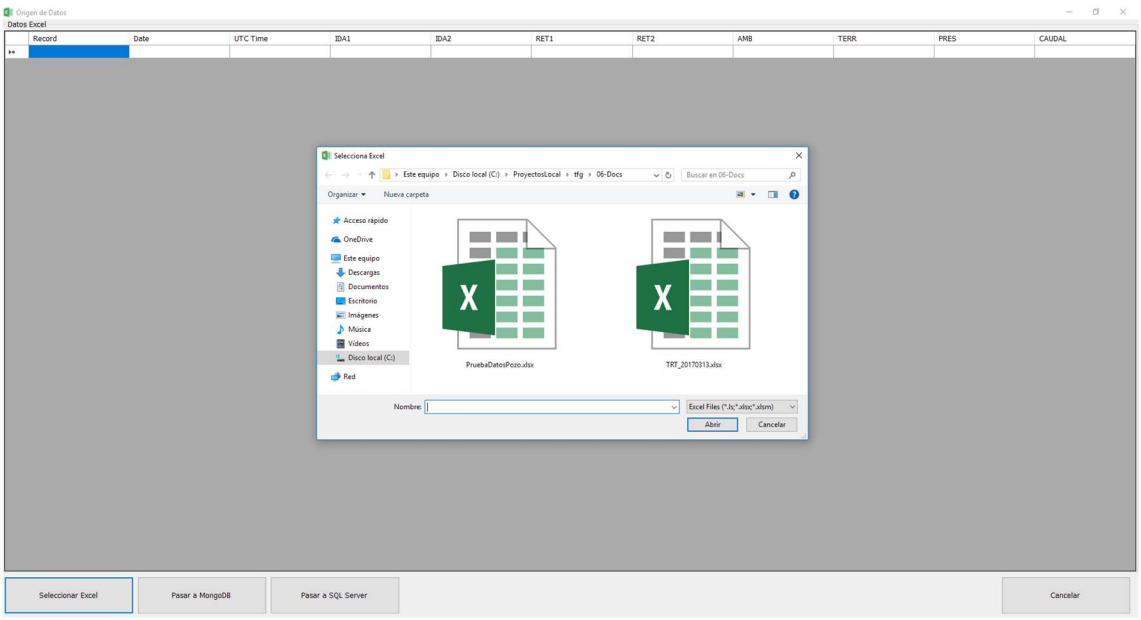

Fig. 16 Formulario Pasarela.

3.2 Web Api MongoDB - NodeJS

En esta primera Web Api, se ha elegido la dupla MongoDB – NodeJS, como explicamos en el análisis, la dupla MongoDB con NodeJS en la actualidad existe mucha documentación para implementar una Web Api, así como varias librerías de acceso a MongoDB desde NodeJS que nos facilitan mucho a la hora de implementarla.

Como ya comentamos anteriormente damos esta opción de Código Abierto y gratuita para dar el máximo abanico de posibilidades a la hora de llevarla a producción.

En esta versión de la Web Api, cabe reseñar que es mucho más ligera y rápida a la hora de la instalación tanto del SGBD como del motor NodeJS, no cabe mencionar también,

que nos da la posibilidad de instalación tanto en un sistema Microsoft Windows [21] como Unix [22] [23].

También cabría destacar, que para esta primera Web Api, en su momento se implementó una primera versión con la librería Mongoose [24]. Con esta versión de la Web Api, solo accedíamos a una colección, la única que teníamos en ese momento en la base de datos. Implementamos esta primera versión con Mongoose, ya que nos daba la posibilidad de dejar una estructura del proyecto simple y ordenada.

Para la segunda versión de la Web Api, utilizamos la librería nativa de MongoDB para NodeJS. En esta ocasión accedemos a un número mayor de colecciones, así como varias funcionalidades más. La estructura en este caso esta implementada en un único fichero "app.js".

3.2.1 Estructura de la Web Api con Mongoose

La estructura en este caso la tendremos desarrollada en un proyecto de JavaScript. Dentro de la carpeta diferenciaremos los controladores, modelos, etc... en diferentes subcarpetas. El fichero principal en este caso será "app.js", es el fichero que arrancaremos cuando lancemos la Web Api.

En la carpeta controllers, implementaremos el controlador de acceso a las Lecturas almacenadas en MongoDB. En el controlador implementaremos todos aquellos métodos de consulta, inserción, eliminación de datos como nos fuera necesario. En nuestro caso solo nos hizo falta implementar un método, el GET Universal, que nos devuelve todas las Lecturas almacenadas en la base de datos.

Dentro de la carpeta modelos, almacenaremos el modelo de datos que le será devuelto al usuario cuando solicite información a la Web Api.

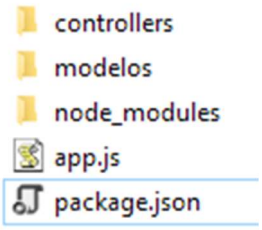

Fig. 17 Estructura WebApi Node.js

3.2.2 Estructura de la Web Api con MongoDB

La estructura en este caso la tendremos desarrollada en un único fichero, como hemos comentado anteriormente. El fichero principal en este caso será "app.js", es el fichero

que arrancaremos cuando lancemos la Web Api, así como contendrá toda la configuración de dirección, puerto, métodos aceptados, formato de los datos, etc..

3.2.3 Librerías utilizadas.

Se han utilizado varias librerías para implementar las diferentes Web Apis con NodeJS sobre MongoDB. Como hemos comentado anteriormente hemos implementado varias versiones de esta Web Api, y en lo referente a las librerías la diferencia entre la primera versión y la segunda, es la librería de acceso a datos. La primera utiliza la librería Mongoose y la segunda utiliza la librería nativa de MongoDB.

3.2.3.1 *Mongodb*

Driver de conexión, consulta, etc... para JavaScript. Con esta librería podemos consultar de forma directa las distintas colecciones almacenadas en la base de datos. De esta forma podemos incluir en un único fichero, al ser una aplicación pequeña, toda la lógica de la misma.

```
mongodb.MongoClient.connect('mongodb://localhost:27017/pozoTermico',
function (err, database) {
    if (err) {
        console.log('Error Conexión MongoDB');
        console.log(err);
        throw err;
    }
    db = database;
    var server = app.listen(3000, function () {
        var port = server.address().port;
        console.log("Servidor Web Api en el puerto: ", port);
    });
})
```

Con el driver nativo de MongoDB, podemos configurar, consultar una colección, etc... en una misma petición a la Web Api.

```
app.get("/api/Lecturas",function(req, res){
    db.collection(col_lecturas).find({}).toArray(function(err,docs){
        if(err){
            console.log("Error    al    recuperar    las    lecturas.",
err.message);
        }
        else{
            res.status(200).json(docs);
```

```
        }
    });
});
```

3.2.3.2 *Mongoose*

Con Mongoose realizaremos la conexión con MongoDB, de una forma más estructurada diferenciando los modelos, controladores y enrutando las consultas. Anteriormente hemos citado la estructura que dotaríamos a la Web Api.

Como podemos observar la conexión realizada con el driver nativo de MongoDB, como con Mongoose son muy similares.

```
//Conexion con MongoDB
mongoose.connect('mongodb://localhost:27017/pozoTermico', function(err,
res) {
    if(err)
    {
        console.log('Error Conexión MongoDB');
        console.log(err);
         throw err;
    }
});
```

Con mongoose podemos modelar los datos que queremos que reciba la aplicación.

```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;

var lecturaSchema = new Schema({
     ID: Number,
     FechaLectura: Date,
     HoraLectura: Date,
     PrimeraIda: Number,
     SegundaIda: Number,
     PrimeraVuelta: Number,
     SegundaVuelta: Number,
     MediaPrimera: Number,
     MediaSegunda: Number,
     TemperaturaAmbiente: Number,
     TemperaturaTerreno: Number,
     Presion: Number,
     Caudal: Number,
     Potencia: Number ,
       ModeloA: Number,
       ModeloB: Number,
     ACC_Control: Number
});
```

```
module.exports = mongoose.model('Lectura', lecturaSchema);
```

Podemos configurar un controlador utilizando el Modelo que acabamos de configurar. En nuestro caso solo nos hacía falta hacer una consulta con todos los datos contenidos en la colección de Lecturas.

```
var mongoose = require('mongoose');
var Lectura = mongoose.model('Lectura');

//GET - Return all registers
exports.findAll = function(req, res) {
    Lectura.find(function(err, lecturas) {
    if(err) res.send(500, err.message);
        console.log('GET /Lecturas')
    res.status(200).jsonp(lecturas);
    });
};
```

3.2.3.3 *Express*

Con la librería de NodeJS, Express [25], podemos generar una infraestructura Web rápida, minimalista y flexible. Simplemente realizando una simple llamada a la librería, configurando la respuesta de la petición y diciéndole la dirección y puerto donde debe atender, tendremos montado nuestro servidor, Web Api para nosotros, que atenderá las peticiones que se realicen.

```
var app = express();
...
api.route('/Lecturas').get(LecturaCtrl.findAll);
app.use('/api', api);

// Start server
app.listen(3000, function() {
 console.log("Servidor Node Pozo termico running on
http://localhost:3000");});
```

3.2.3.4 *Body-Parser*

Con la librería Body-Parser [26], *middleware* encargado de formatear tanto la información de salida como de entrada a la Web Api, para que esta la reciba en formato JSON.

3.2.4 Peticiones Web Api

Ahora detallaremos las diferentes peticiones implementadas para la Web Api de NodeJS – MongoDB. Como podremos observar en este caso, no están implementados los diferentes modelos de simulación utilizados en el estudio.

3.2.4.1 Petición Lecturas.

En esta petición obtenemos las lecturas almacenadas en la base de datos, en el caso de la Web Api con NodeJS, solo se ha implementado una opción.

3.2.4.1.1 Get Universal

No le pasaremos ningún valor como parámetro y nos devolverá todos los valores que hay almacenados en la base de datos.

```
get(http://localhost:3000/api/Lecturas)
```

3.2.4.1.2 Get Fecha Mínima y Máxima.

Con esta petición recogeremos la fecha mínima y máxima, de las lecturas que actualmente tenemos almacenadas en MongoDB.

```
get(http://localhost:3000/api/FechaMinMax?id=1)
get(http://localhost:3000/api/FechaMinMax?id=2)
```

3.2.4.2 Petición Configuración Básica

Con esta petición recuperaremos la configuración básica de los modelos, esto es el diámetro, profundidad, resistividad, alfa, etc…

```
get(http://localhost:3000/api/ConfiguracionBase)
```

3.2.4.3 Petición Modelos

Con este controlador recuperamos los distintos modelos calculados en la aplicación, y los cuales podremos añadir al grafico de modelos.

```
get(http://localhost:3000/api/ModelosComparacion)
```

3.3 Web Api SQL Server - .NET

En esta segunda Web Api, hemos elegido la dupla SQL Server - .NET. Se ha elegido esta segunda dupla porque es la que más conocemos, por experiencia laboral, y con la que nos encontramos más cómodos.

3.3.1 Estructura de la Web Api

La estructura de la Web Api está basada en 4 proyectos de .Net. Tres de los proyectos son bibliotecas de clases y el cuarto es una Web Api [27]. Los tres proyectos de clases son:

- TFG.PozoTermico.Datos en el cual tendremos el fichero de acceso a datos (Linq To SQL [28]). Este fichero es un .dbml, que actúa de una forma similar a una base de datos. [29]

- TFG.PozoTermico.Services en el cual tendremos las clases de acceso a los datos. Tendremos una clase por cada uno de los Controladores que configuremos en la Web Api. Cabe reseñar en este proyecto la necesidad de implementar interfaces para cada una de las clases implementadas. El porqué de esta necesidad se basa en que después nos hará falta para la implementación de la inyección de dependencias [30] dentro del proyecto de la Web Api.

- TFG.PozoTermico.Entity en el cual tendremos las entidades utilizadas en la aplicación.

- TFG.PozoTermico.WebApi en este proyecto estará contenida la Web Api en sí. Cabe destacar que dentro de este tendremos los Modelos y Controladores del mismo. Así como la configuración de la inyección de dependencias, el mapeo de las entidades utilizadas en la Web Api con sus correspondientes Modelos.

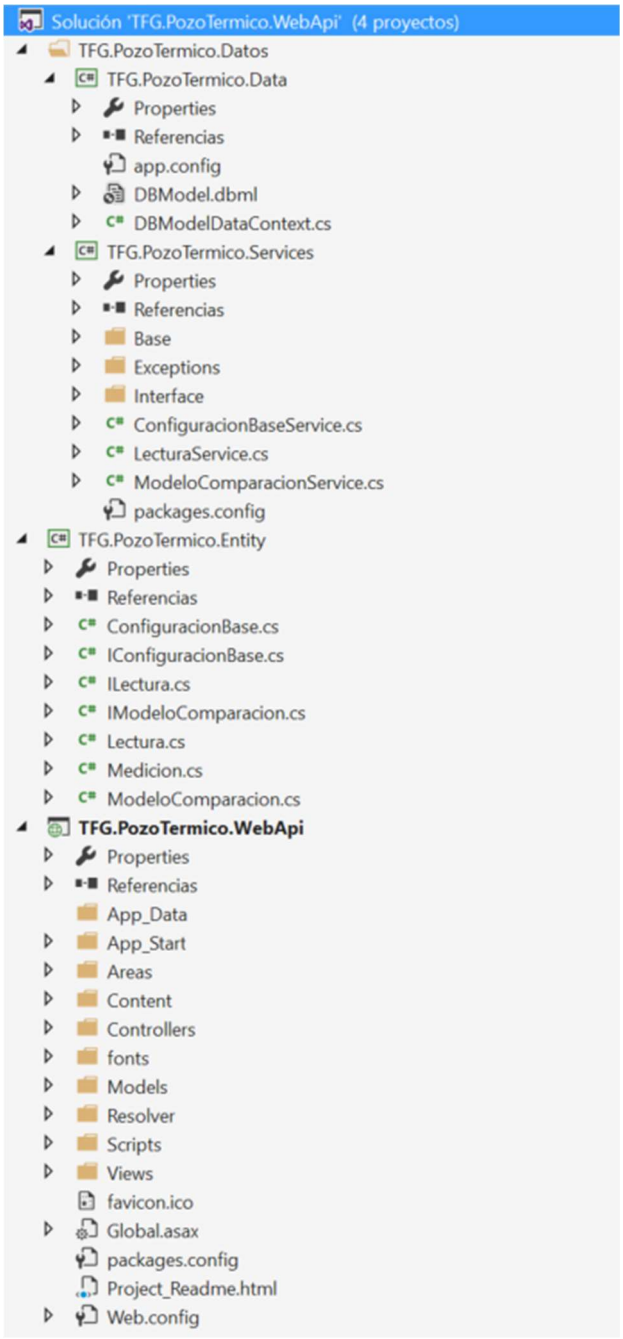

Fig. 18 Estructura WebApi SQLServer – NET

3.3.2 Librerías Utilizadas.

En este caso no nos ha sido necesario instalar ninguna librería de acceso a datos, como en los otros casos. El acceso a los datos se realiza de forma nativa con las librerías que lleva el framework .NET de Microsoft. Las librerías instaladas son utilizadas para el mapeo de los objetos, y para la inyección de dependencias.

De la misma forma que en la aplicación de la pasarela, hemos utilizado NuGet para la instalación de las librerías.

3.3.2.1 *AutoMapper*

AutoMapper [31] es una librería encargada de hacer un mapeo de los objetos. En nuestro caso de las entidades generadas en el proyecto TFG.PozoTermico.Entity, con los modelos que configuraremos en TFG.PozoTermico.WebApi encargado de servir las peticiones recibidas por la Web Api.

Para poder hacer el mapeo de forma automática, es necesario que configuremos en la Web Api, la correspondencia entre la Entidad y el Modelo. Esta configuración se encuentra dentro del fichero `~/App_Starat/WebApiConfig.cs`.

```
Mapper.Initialize(cfg =>
{
cfg.CreateMap<Lectura, LecturaModel>();
       cfg.CreateMap<ModeloComparacion, ModeloComparacionModel>();
       cfg.CreateMap<ConfiguracionBase, ConfiguracionBaseModel>();
});
```

3.3.2.2 *Microsoft Practices Unity*

Microsoft Practices Unity [32] es una librería encargada de la inyección de dependencias a través de un contenedor global. Igual que la librería anterior, tendremos que configurarlo en el fichero `~/App_Start/ WebApiConfig.cs`.

Como hemos comentado anteriormente en el proyecto TFG.PozoTermico.Services, deberemos tener implementado las interfaces de las mismas. El porqué de esta implementación es muy sencillo, en el caso de que en un futuro decidiéramos cambiar la base de datos, simplemente implementando las nuevas clases de Servicios hijas de las interfaces, y modificando el fichero de configuración de la Web Api, ya tendríamos en funcionamiento nuestra Web Api sin grandes cambios y de una forma bastante transparente.

```
var container = new UnityContainer();
container.RegisterType<ILecturaService,LecturaService>(new
ierarchicalLifetimeManager());
container.RegisterType<IModeloComparacionService,
ModeloComparacionService>(new HierarchicalLifetimeManager());
container.RegisterType<IConfiguracionBaseService,
ConfiguracionBaseService>(new HierarchicalLifetimeManager());
config.DependencyResolver = new UnityResolver(container);
```

3.3.3 Controladores Web Api

Vamos a describir con más detalle los controladores implementados en la Web Api SQL Server - .NET. La gran diferencia con la Web Api NodeJS – MongoDB, es que en esta versión están implementados los modelos de simulación utilizados en el estudio. Estos modelos los explicaremos más adelante.

Cabe reseñar que en este caso también podríamos filtrar la información de las lecturas por rango de tiempo, así como recalcular los modelos con los valores que nosotros le introducimos a la aplicación Web.

3.3.3.1 *Controlador Lecturas*.

En este controlador devolveremos toda la información referente a las lecturas que estén almacenadas en la base de datos. Pudiendo filtrar la información y cambiar los valores de cálculo de los modelos si nos hiciera falta.

3.3.3.1.1 Get Universal

No le pasaremos ningún valor como parámetro y nos devolverá todos los valores que hay almacenados en la base de datos, calculando los modelos con la configuración base que hay almacenada en la base de datos.

```
get(http://localhost:35398/api/Lecturas)
```

3.3.3.1.2 Get Calculado

Le pasaremos como parámetros el diámetro, profundidad, temperatura inicial, alfa, lambda y la resistividad. A partir de estos valores calculara los diferentes modelos, con todos los datos almacenados en la base de datos.

```
get(http://localhost:35398/api/Lecturas?diametro=0.08&profundidad=39&t
Inicial=19&alfa=6e-7&lambda=2&resistividad=0.2)
```

3.3.3.1.3 Get Calculado Rango Fechas

Igual que la anterior llamada, también le podemos pasar el rango de fechas que queremos que utilice para el cálculo del modelo, así como la fecha en la que queremos que comience a tener en cuenta la fecha del modelo.

```
get(http://localhost:35398/api/Lecturas?diametro=0.08&profundidad=39&t
Inicial=19&alfa=6e-7&lambda=2&resistividad=0.2&fIni=2017-04-
20&fFin=2017-05-03&fMod=2017-04-27)
```

3.3.3.2 *Controlador Configuración Base*.

En este controlador, devolvemos la configuración que hay por defecto en la base de datos, de los valores utilizados para el cálculo de los modelos.

En este controlador solo esta implementado el GET Universal, por lo que nos devolverá toda la información que hay almacenada en la base de datos.

```
get(http://localhost:35398/api/ConfiguracionBase)
```

3.3.3.3 *Controlador Fechas*.

En este controlador, devolveremos la fecha de la lectura mínima almacenada en la base de datos, así como la fecha máxima.

En este caso solo tendremos un método al cual le pasaremos un valor, que será el que nos indicara si el valor que queremos es el máximo o el mínimo.

```
get(http://localhost:35398/api/FechaMinMax?id=1)
get(http://localhost:35398/api/FechaMinMax?id=2)
```

3.3.3.4 *Controlador Modelos Simulación*.

En este controlador, devolvemos un listado con aquellos modelos calculados que actualmente hay implementados en el sistema.

```
get(http://localhost:35398/api/ModeloComparacion)
```

3.3.4 Modelos de simulación.

Dentro de la Web Api de SQL Server - .NET, se calculan unos modelos de simulación utilizados en el estudio para comparar los valores obtenidos. Estos modelos de simulación son modelo de fuente de línea infinita (*Infinite Line Source* [33] o *ILS* en inglés) y el modelo de fuente de línea finita (*Finite Line Source* [34] o *FLS* en inglés).

Para el cálculo de estos modelos se han utilizado en el caso del GET universal los valores por defecto que hay en la base de datos almacenados. En el caso de que el usuario quisiera modificar uno de esos valores y que se recalculara, se podría hacer con una de las otras llamadas existentes en el controlador de Lecturas.

Para estos dos modelos intervienen la profundidad (*H),* el diámetro (r_b*),* temperatura inicial (T_0), alfa (α), lambda (λ) y resistividad (R_b).

En las dos fórmulas intervendrá una variable calculada denominada (q_z *),* este valor será una constante calculada en los dos modelos. Este q_z *,* se calculará a partir de la siguiente formula.

$$qz = \frac{1}{n}\frac{\sum_{i=1}^{n}(T_{in_i} - T_{out_i})\, G_i C_{f_i}}{H}$$

Ecuación 1 Calculo de la qz

Donde intervendrán el caudal inyectado en ese momento (G_i), asi como un coeficiente calculado (C_{f_i}).

3.3.4.1 Infinite Line Source

La fórmula correspondiente a este modelo matemático de Eskilson es la siguiente.

$$T_{f,ILS}(t) = \frac{q_z}{4\pi\lambda}\left(\ln\left(\frac{4\alpha t}{r_b^2}\right) - \gamma\right) + q_z R_b + T_0 \quad \text{for } t \geq \frac{5r_0^2}{\alpha}$$

Ecuación 2 Formula ILS

Donde (t), es el instante de tiempo desde que es mayor al indicador que nos dice que los datos ya son buenos para tomarlos para el modelo.

El valor de (γ) *eulergama* es un valor constante predefinido por “0.577216” o bien en valor hexadecimal “3fe2788cfc6fb619”.

3.3.4.2 Finite Line Source

La diferencia de este modelo con el anterior es el factor de corrección que se le aplica a la formula.

$$T_{f,ILS}(t) = \frac{q_z}{4\pi\lambda}\left(\ln\left(\frac{4\alpha t}{r_b^2}\right) - \gamma - 3\sqrt{\frac{4}{\pi}}\left(\sqrt{\frac{t}{t_z}} + \frac{1}{4}\sqrt{\frac{t_z}{t}}\beta^2\right)\right) + q_z R_b + T_0 \quad \text{for } t \geq \frac{5r_0^2}{\alpha}$$

Ecuación 3 Formula de FLS

Como se puede observar a partir del valor del eulergamma, se aplica el valor de corrección calculado al instante de tiempo actual.

3.4 Aplicación Web

Para la aplicación Web, hemos escogido hacer la parte del FrontEnd [35] en Angular 2 [36] (Angular 4 en el momento de escribir esta memoria). Hemos optado por esta opción junto con Angular-cli, porque deja la estructura del proyecto mucha más ordenada y el código mucho más limpio y fácil de entender. Otro de los motivos por los que elegir este framework es que podemos instalarlo tanto en una plataforma Windows o Unix.

El usuario con la aplicación podrá llevar a cabo varias acciones.

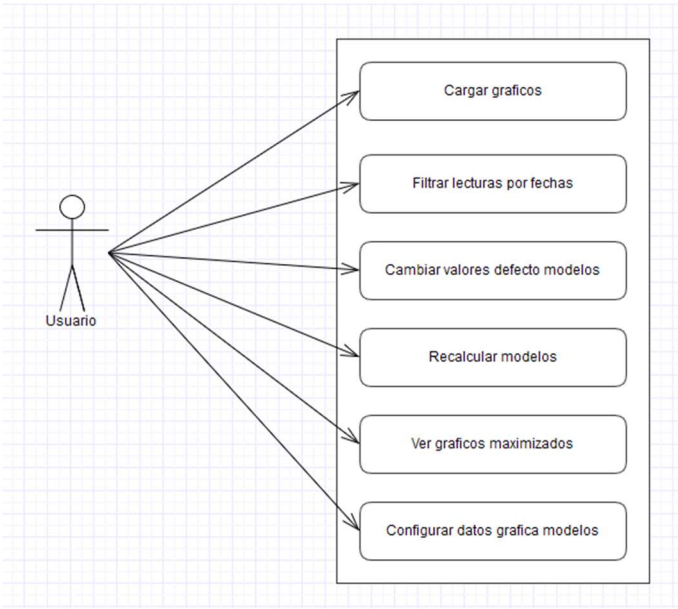

Fig. 19 Caso de Uso Aplicación Web

3.4.1 Estructura de la Aplicación.

Angular-cli proporciona una estructura de inicio sencilla y fácil de mantener. Para la aplicación hemos generado dentro de la carpeta “app” una carpeta “principal”. Esta carpeta contendrá los distintos componentes de nuestra aplicación, así como los modelos y servicios utilizados por la misma.

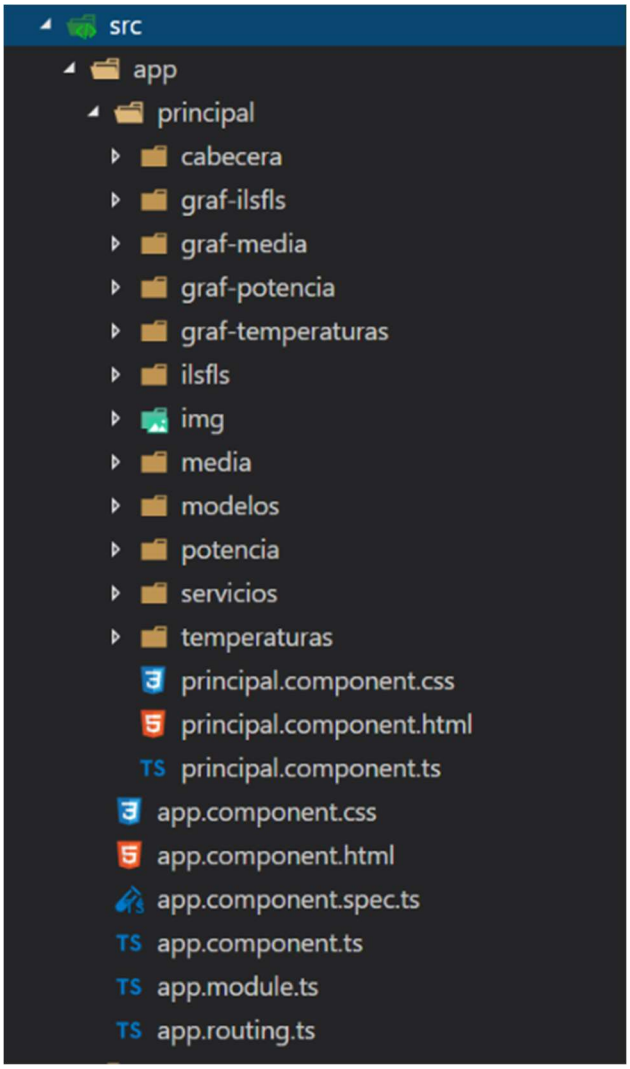

Fig. 20 Estructura Aplicación Angular.

Dentro de la carpeta principal podemos dividir la aplicación en cuatro partes, las cuales explicaremos con detalle a continuación.

3.4.1.1 *Componente principal*.

En este componente es de donde arranca nuestra aplicación. Este contendrá el resto de componentes implementados en la aplicación, cabecera, gráficos, etc…

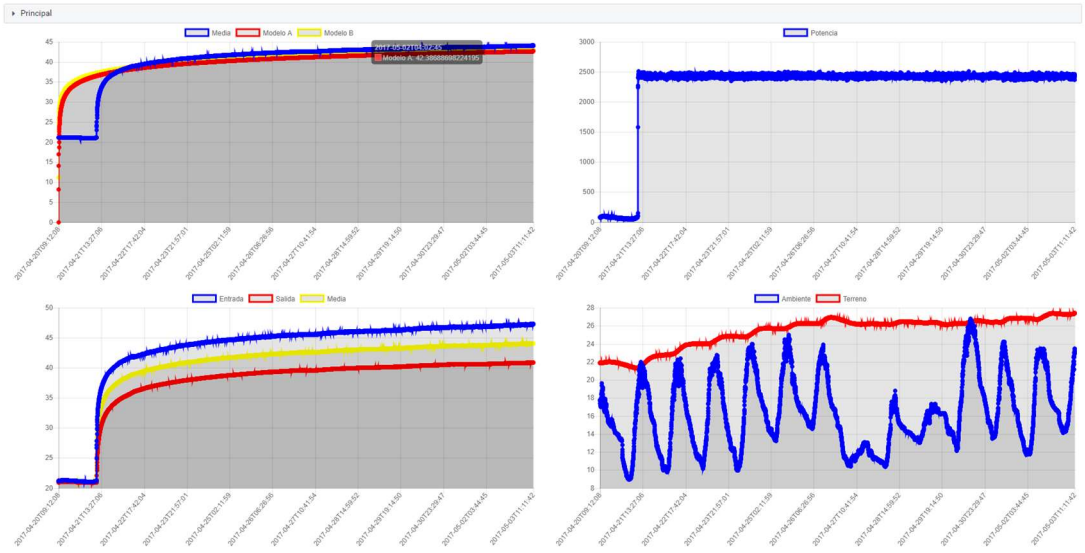

Fig. 21 Componente Principal

3.4.1.2 *Componentes gráficos*.

En cada uno de estos componentes estarán los distintos gráficos de la aplicación. Los componentes son:

- GrafIlsFlsComponent. (Media y Modelos de estudio).
- GrafMediaComponent. (Media, Temperatura entrada y salida).
- GrafPotenciaComponent. (Potencia inyectada).
- GrafTemperaturasComponent. (Temperatura Terreno y Ambiente).

En el archivo *html* contendrá el código del gráfico, y en cada uno de los *ts´s* la configuración de cada uno de los gráficos.

```
<div class="classDvTamanyoGrafica">
    <div class='row' id="dvGrafica">
        <canvas baseChart
                [datasets]="lineChartDataModelos"
                [labels]="lineChartLabelsModelos"
                [options]="lineChartOptions"
                [colors]="lineChartPresionColorsModelos"
                [legend]="lineChartLegend"
                [chartType]="lineChartType">
        </canvas>
```

```
    </div>
</div>
```

Definiciones dentro del archivo *ts,* con la configuración de la gráfica.

```
public arrayTemperaturaMedia: number[];
    public arrayTemperaturaModeloA: number[];
    public arrayTemperaturaModeloB: number[];
    //Arrays de labels
    public lineChartLabelsModelos: Date[];

    public lineChartDataModelos: Array<any> = [];
    public lineChartPresionColorsModelos: Array<any> = [
        { // blue
            borderColor: '#00f',
            pointBackgroundColor: '#00f',
            pointBorderColor: '#00f',
            pointHoverBackgroundColor: '#00f'
        },
        { // red
            borderColor: '#f00',
            pointBackgroundColor: '#f00',
            pointBorderColor: '#f00',
            pointHoverBackgroundColor: '#f00'
        },
        { // yellow
            borderColor: '#ff0',
            pointBackgroundColor: '#ff0',
            pointBorderColor: '#ff0',
            pointHoverBackgroundColor: '#ff0'
        }
    ];
    public lineChartLegend: Boolean = true;
    public lineChartType: String = 'line';
    public lineChartOptions: any = {
        responsive: true,
        beginAtZero: true
    };
```

3.4.1.3 *Componente Cabecera*.

En este componente tenemos el menú de navegabilidad por la aplicación, los valores con los que hemos calculado los modelos y el rango de fechas de los datos que tenemos cargados.

Los valores con los que se ha calculado los modelos inicialmente se pueden modificar, y volver a cargar las gráficas con los datos actualizados.

También podemos filtrar la información contenida en los gráficos por rango de fechas, nunca pudiendo poner una fecha menor o mayor a los datos que hay actualmente cargados en los distintos SGBD.

Fig. 22 Componente Cabecera.

3.4.1.4 *Componentes Contenedores*.

En estos componentes podemos ver los gráficos que se han cargado en el principal, de una forma individual y maximizada. Cada uno de estos componentes hará referencia al componente cabecera y al grafico que le corresponda. Lo explicaremos con más detalle más adelante.

Los componentes son:

- IlsFlsComponent. (Media y Modelos de estudio).
- MediaComponent. (Media, Temperatura entrada y salida).
- PotenciaComponent. (Potencia inyectada).
- TemperaturasComponent. (Temperatura Terreno y Ambiente).

3.4.2 Librerías Utilizadas

Las librerías utilizadas en este caso no son para el acceso a los datos, sino que utilizamos unas librerías externas para la generación de los gráficos, así como para darle una parte más agradable a la aplicación.

3.4.2.1 *Ng2-Charts*

Ng2-Charts [37] esta librería de angular está basada en la librería de JavaScript "Chart.js" [38]. Con esta librería podremos hacer gráficos de varios tipos como líneas, barras, tarta, etc. Solamente tendremos que pasarle al grafico la colección de datos, etiquetas y las opciones para el grafico.

Definiremos en el fichero html el código que hace referencia al gráfico, y en el fichero de TypeScript [39] definiremos tanto la colección con los datos, opciones de configuración,etc….

```
<canvas baseChart
      [datasets]="lineChartDataMedia"
      [labels]="lineChartLabelsMedia"
```

```
      [options]="lineChartOptions"
      [colors]="lineChartPresionColorsMedia"
      [legend]="lineChartLegend"
      [chartType]="lineChartType">
</canvas>
```

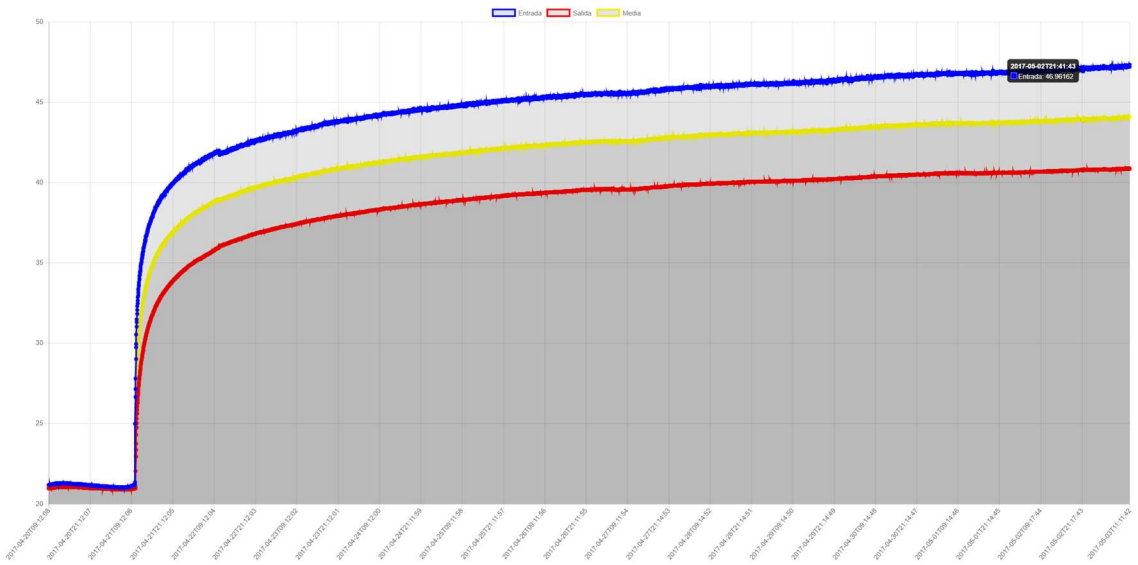

Fig. 23 Grafica Temperatura con Ng2-charts

3.4.2.2 *PrimeNg*

PrimeNg [40] es una librería gráfica que tiene una gran variedad de componentes para Angular, así como para otros frameworks de JavaScript y es de código abierto.

La variedad de componentes de la librería va desde un simple botón, hasta un popup para subir ficheros al servidor, pasando por un componente calendario.

Para poder utilizar cada uno de los componentes incluidos en la librería, tendremos que hacer referencia dentro de nuestro fichero "`app.module.ts`", al componente que vayamos a utilizar.

Fig. 24 Componentes PrimeNG

3.4.3 Páginas y Gráficos.

Para el usuario que vaya a utilizar la aplicación, está, está compuesta por 5 paginas. Una primera que es la principal e inicial, y 4 páginas más, una por cada uno de los gráficos.

3.4.3.1 *Página Principal*

Como hemos explicado anteriormente, desde esta página podemos ver de inicio los 4 gráficos de la aplicación, pudiendo desde esta ir a los distintos gráficos, modificar los datos para el cálculo de los modelos, así como filtrar la información que queremos visualizar en los gráficos.

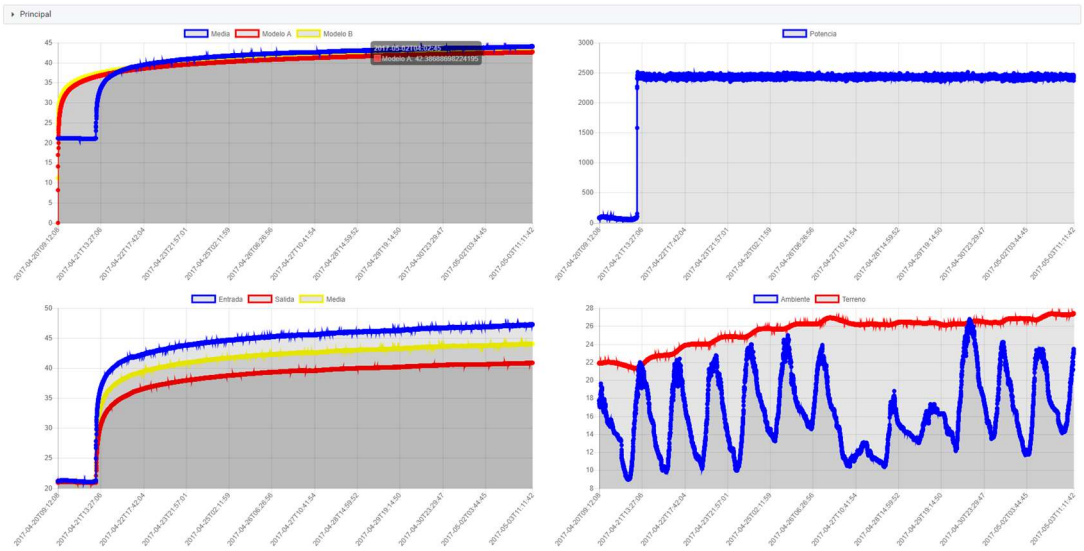

Fig. 25 Componente Principal

3.4.3.2 *Página Modelos*

En esta página podemos ver la media entre la temperatura de entrada y salida, más aquellos modelos que queramos añadirle al gráfico.

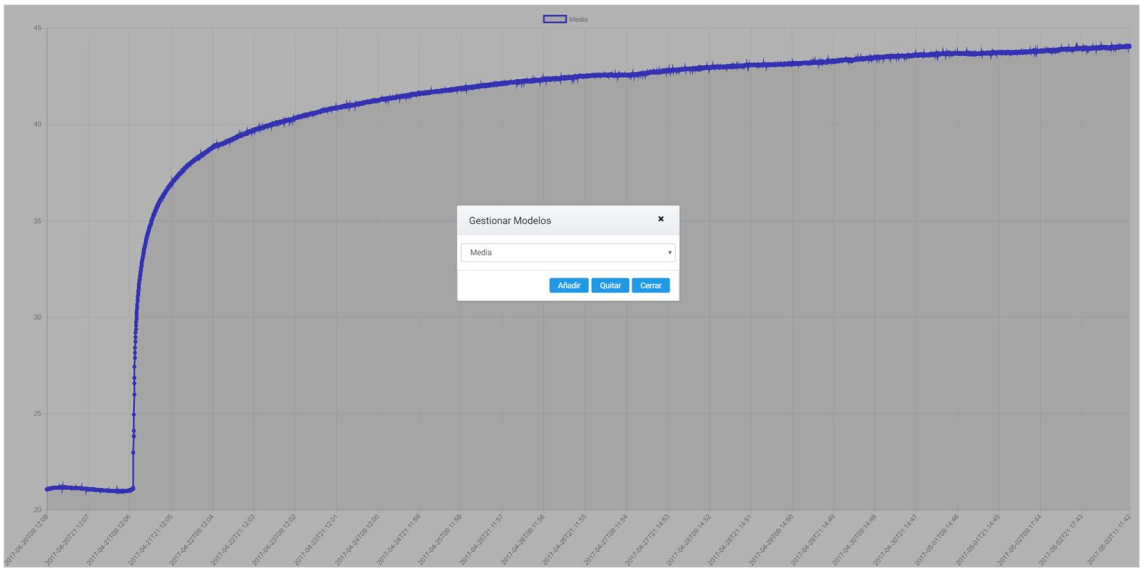

Fig. 26 Pagina Modelos

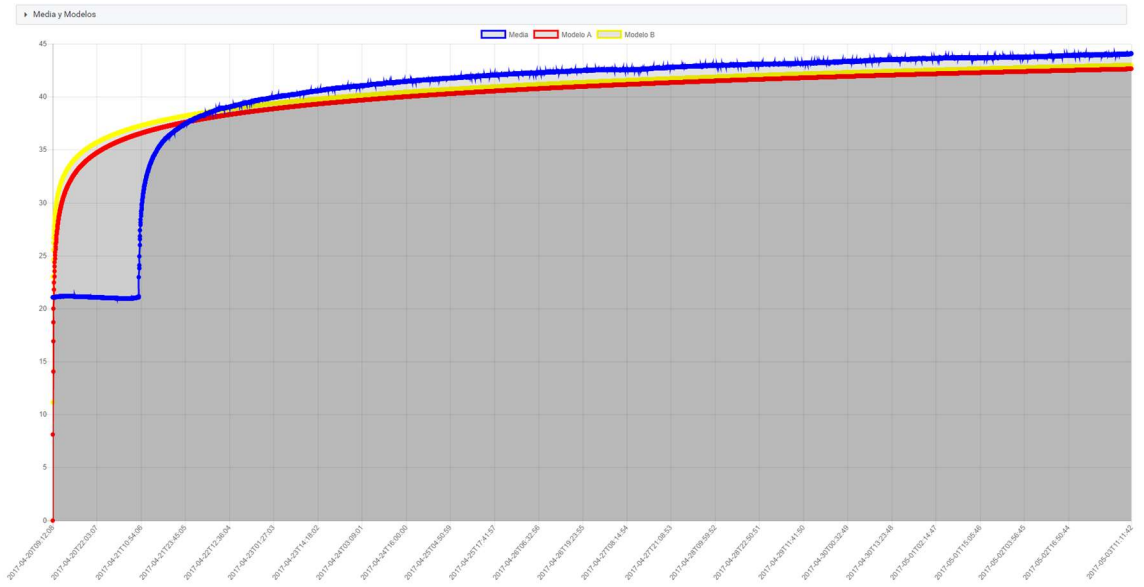

Fig. 27 Pagina modelos incluidos en el gráfico.

3.4.3.3 *Página Potencia*

En esta página podemos ver el grafico de la potencia inyectada en el estudio. Para poder calcular la potencia necesitamos la siguiente formula.

$$Pot = (\ _{in} - T_{out})\ G_i C_{f_i}$$

Ecuación 4 Formula de la potencia.

Si nos fijamos bien en la fórmula de la potencia podremos observar, que es parte de la fórmula para el cálculo de la (q_z). Es el cálculo que se realiza en el sumatorio del numerador.

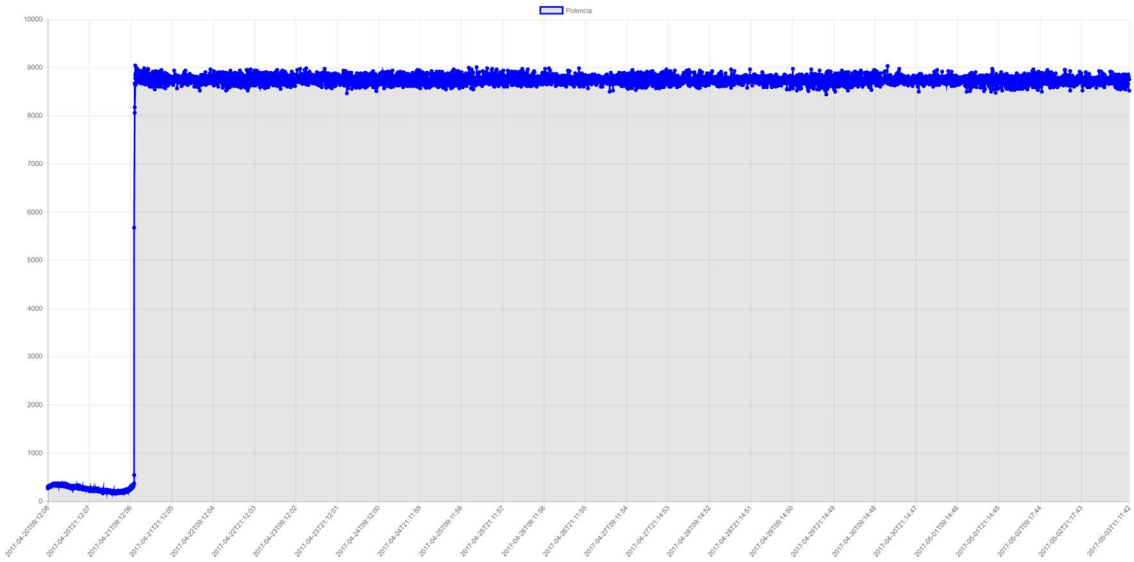

Fig. 28 Página Potencia

3.4.3.4 *Página Media*

En esta página podemos ver la temperatura de entrada, la temperatura de salida y una media de las dos. Los dos primeros valores vienen ya predefinidos de la base de datos no será necesario ningún calculo, no así el tercero que será una media aritmética de los dos valores.

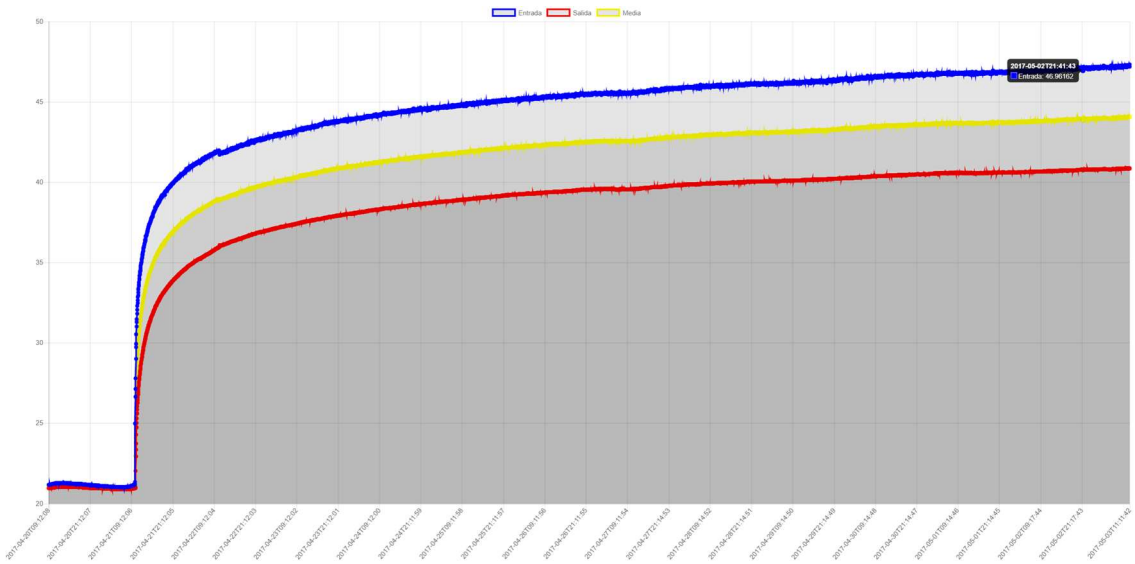

Fig. 29 Grafica con Ng2-charts

3.4.3.5 *Página Temperaturas.*

En esta página podemos ver la temperatura del terreno y la temperatura ambiente. Estos dos valores vienen de la base de datos y no se realiza ningún cálculo sobre ellos.

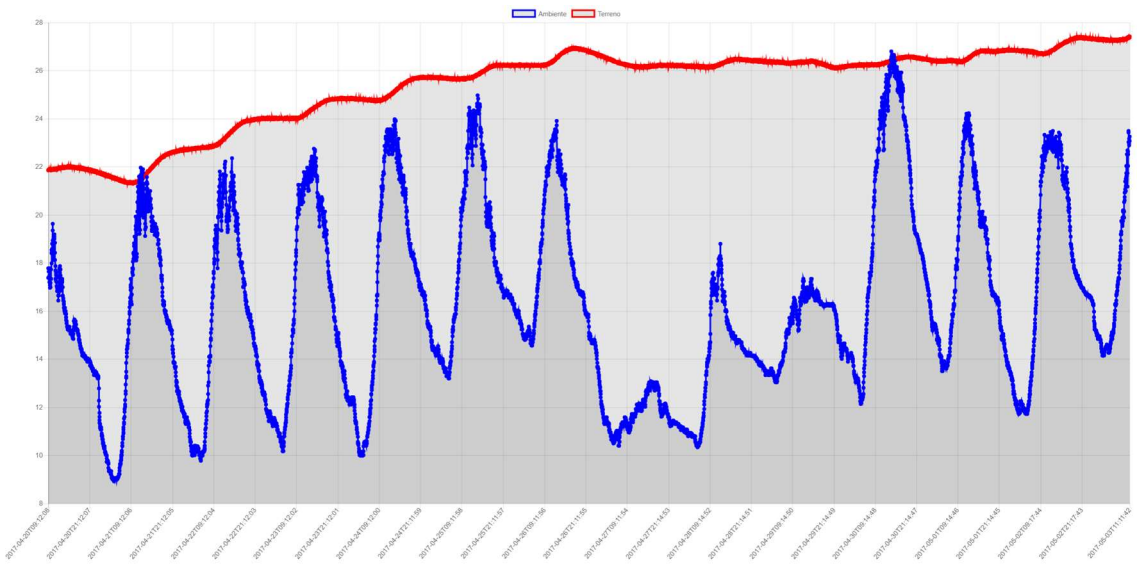

Fig. 30 Página Temperatura Terreno y Ambiente

4 Pruebas

Como se puede observar en las imágenes anteriormente mostradas, se ha cogido un fichero con 6279 lecturas. Estas lecturas van desde la fecha 20/04/2017 a las 9:12:08, hasta la fecha 03/05/2017 a las 11:11:42.

Estas lecturas están tomadas cada 3 minutos, y contienen la fecha, hora, dos valores de entrada, dos valores de salida, temperatura ambiente, temperatura del terreno, presión aplicada, caudal y un valor de control.

	Record	Date	UTC Time	IDA1	IDA2	RET1	RET2	AMB	TERR	PRES	CAUDAL	ACC_CONTROL
1	1	2017-04-20 09:12:08.000	09:12:08	21,15596	21,10532	20,9724	21,01671	17,37341	21,86918	3,277271	0,3428819	0
2	2	2017-04-20 09:15:08.000	09:15:08	21,16862	21,09899	20,95975	21,01671	17,77127	21,84658	3,280527	0,3404948	0
3	3	2017-04-20 09:18:08.000	09:18:08	21,17495	21,09899	20,96607	21,02937	17,74233	21,86353	3,271846	0,3404948	0
4	4	2017-04-20 09:21:08.000	09:21:08	21,16862	21,09899	20,96607	21,01671	17,63744	21,89742	3,274016	0,3404948	0
5	5	2017-04-20 09:24:08.000	09:24:08	21,17495	21,11798	20,95975	21,00405	17,63744	21,86353	3,27691	0,3404948	0
6	6	2017-04-20 09:27:08.000	09:27:08	21,16862	21,12431	20,9724	21,00405	17,7785	21,90307	3,276186	0,3404948	0
7	7	2017-04-20 09:30:08.000	09:30:08	21,18761	21,10532	20,96607	21,01671	17,4783	21,89742	3,28342	0,3404948	0
8	8	2017-04-20 09:33:08.000	09:33:08	21,17495	21,11166	20,9724	21,01671	17,33001	21,90307	3,28342	0,3404948	0
9	9	2017-04-20 09:36:08.000	09:36:08	21,18128	21,13697	20,97873	21,02304	17,19256	21,89742	3,277271	0,3404948	0
10	10	2017-04-20 09:39:08.000	09:39:08	21,18128	21,12431	20,9724	21,02304	17,09129	21,86353	3,280527	0,3404948	0

Fig. 31 Datos Pruebas SQL Server

Los datos de prueba en las gráficas anteriormente comentadas serian estos de aquí definidos, tanto para modelos, media, etc… Las gráficas superiores muestran el resultado de pruebas con un estudio real sobre el pozo de la UPV.

Conclusiones

Los pozos de intercambio de calor (*borehole heat exchanger* o BHE en inglés) son instalaciones geotérmicas, que persiguen mejorar la eficiencia energética de los sistemas de climatización. Para el correcto diseño de estos sistemas es necesario conocer las características del terreno.

El primer objetivo del proyecto a nivel conceptual, era entender el funcionamiento de un ensayo de respuesta térmica, así como el entender el funcionamiento del ERT instalado en el Campus de Vera.

El primer objetivo del proyecto más técnico, era recoger y tratar la información del PLC, este objetivo se ha cumplido junto con el segundo, almacenando la información obtenida en un SGBD.

Se han implementado dos Base de datos en MongoDB y otra en SQL Server. De esta forma en desarrollos futuros se puede optar por una solución de Código Abierto o por una comercial.

Otros de los objetivos era realizar una aplicación Web, y esta tenía que dibujar en unos gráficos la información obtenida de los datos almacenados en las bases de datos. Se ha implementado una solución con Angular 2 y Chart.js. Hemos optado por esta tecnología, ya que en el momento de seleccionar una era la que más fama y demanda profesional tenia.

Para que la aplicación pueda mostrar los datos en las gráficas, hemos implementado dos Web Apis una de código abierto con NodeJS y otra con MVC.NET, este era otro de los objetivos del proyecto.

Hemos decidido dar esta doble solución al problema, para que el usuario pueda optar o bien por una solución de código abierto o por una solución más comercial.

Otro objetivo era implementar los diferentes modelos, para su posterior análisis. En este caso en la versión de .NET, sí que tendríamos implementado el cálculo de dichos modelos. Por el contrario, en el caso de la versión de NodeJS, no estaría implementada por la falta de tiempo de llevarlo a cabo.

En trabajos futuros se podría implementar un driver para la recogida de la información de una forma más directa y sencilla. Con este driver, realizar un servicio encargado de ir almacenando la información leída en las distintas bases de datos.

Darle más interactividad al usuario con la aplicación, con nuevos gráficos o nuevas funcionalidades como la impresión de los mismos con los datos seleccionados para el estudio. Ver los datos cargados en cada grafica para su análisis.

También se podría escoger para los gráficos una versión comercial y más potente como DevExtreme [41], ya que daría muchísimo más juego a la hora del análisis de los resultados obtenidos en un TRT.

En el caso de que se vayan a incluir varios pozos, se podría modificar la base de datos, para que lleve una relación de los datos a que pozo pertenecen.

Siguiendo la consigna anterior, también se podría hacer en la parte de la aplicación a nivel gráfico, una comparativa tanto de datos como de graficas en un rango de tiempo, muestras analizadas, etc…

Para realizar este proyecto se han utilizado habilidades obtenidas a lo largo de todo el grado, así como la experiencia profesional en el desarrollo de aplicaciones.

De las asignaturas que he realizado en el grado, las dos que más he aplicado en el proyecto serian.

- Interfaces Persona Computador.
- Desarrollo centrado en el Usuario.

A título personal este proyecto me ha servido, para ver el funcionamiento de una Base de datos NoSQL como es MongoDB, y como desarrollar una aplicación con Angular2.

Bibliografía

[1] P. Eskilson, «Thermal Analysis of Heat Extraction Boreholes,» [En línea].

[2] B. a. M. P. M. A. a. M. T. a. L. L. a. U. J. F. Badenes, Proceedings of the European Geothermal Congress 2016, 2016.

[3] T. V. B. a. Á. M. a. E. F. a. J. L. G. S. a. J. M. I. a. J. P. a. P. J. F. d. C. a. J. F. Urchueguía, «Finite line-source model for borehole heat exchangers:,» Geothermics, 2009. [En línea]. Available: http://www.sciencedirect.com/science/article/pii/S0375650509000054.

[4] WikiPedia, «https://es.wikipedia.org/wiki/Valores_separados_por_comas,» [En línea].

[5] WikiPedia, «Stored Procedure,» [En línea]. Available: https://en.wikipedia.org/wiki/Stored_procedure.

[6] Wikipedia, «Web Api,» [En línea]. Available: https://en.wikipedia.org/wiki/Web_API.

[7] WikiPedia, «NoSQL,» [En línea]. Available: https://es.wikipedia.org/wiki/NoSQL.

[8] MongoDB, «MongoDb,» [En línea].

[9] S. SERVER, «SQL SERVER,» [En línea].

[10] NodeJS, «NodeJS,» [En línea].

[11] .. Framework, «.NET Framework,» [En línea].

[12] Angular, «Angular,» [En línea].

[13] Wikipedia, «Json,» [En línea]. Available: https://es.wikipedia.org/wiki/JSON.

[14] Microsoft, «Microsoft Developer Network,» [En línea].

[15] M. MSDN, «Proyecto de clases Visual Studio,» [En línea]. Available: https://msdn.microsoft.com/es-es/library/cc668164(v=vs.100).aspx.

[16] M. ADO.NET, «ADO.NET,» [En línea]. Available: https://msdn.microsoft.com/es-es/library/e80y5yhx(v=vs.110).aspx.

[17] Nuget, «Nuget,» [En línea]. Available: https://www.nuget.org/.

[18] M. Linq, «Microsoft Linq,» [En línea]. Available: https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/concepts/linq/introduction-to-linq-queries.

[19] Genbetadev, «LINQTOEXCEL,» [En línea].

[20] M. Driver, «MongoDB Driver,» [En línea].

[21] Microsoft, «Microsoft Windows,» [En línea]. Available: https://www.microsoft.com/es-es/windows/.

[22] OpenGroup, «Unix,» [En línea]. Available: http://www.opengroup.org/unix.

[23] Wikipedia, «Unix,» [En línea]. Available: https://es.wikipedia.org/wiki/Unix.

[24] Mongoose, «http://mongoosejs.com/,» [En línea].

[25] Express, «Express NodeJS,» [En línea].

[26] Body-Parser, «Body-Parser,» [En línea].

[27] Microsoft, «ASP.NET Web Api,» [En línea]. Available: https://www.asp.net/web-api.

[28] Microsoft, «Linq to SQL,» [En línea]. Available: https://msdn.microsoft.com/es-es/library/bb386976(v=vs.110).aspx.

[29] Microsoft, «DBML,» [En línea]. Available: https://msdn.microsoft.com/es-es/library/bb384428.aspx.

[30] Microsoft, «IOT,» [En línea]. Available: https://msdn.microsoft.com/es-es/communitydocs/net-dev/csharp/inyeccion-de-dependencias.

[31] AutoMapper, «AutoMapper,» [En línea].

[32] Unity, «Unity,» [En línea].

[33] P. Eskilson, «Thermal Analysis of Heat Extraction BoreHoles,» 1987.

[34] T. Bandos, A. Montero, P. Fernández de Córdoba y J. Urchueguía, «Improving parameter estimates obtained from thermal response tests: Effect pf ambient air temperature variations.,» *Geothermics,* 2011.

[35] Wikipedia, «FrontEnd,» [En línea]. Available: https://es.wikipedia.org/wiki/Front-end_y_back-end.

[36] A. CLI, «Angular CLI,» [En línea].

[37] NG2-Charts, «NG2-Charts,» [En línea].

[38] Chartjs.org, «Chart js,» [En línea]. Available: http://www.chartjs.org/.

[39] TypeScript, «TypeScript,» [En línea]. Available: https://www.typescriptlang.org/.

[40] PrimeNG, «PrimeNG,» [En línea].

[41] DevExpress, «DevExtreme,» [En línea]. Available: https://js.devexpress.com/.

[42] T. V. B. a. Á. M. a. E. F. a. J. L. G. S. a. J. M. I. a. J. P. a. P. J. F. d. C. a. J. F. Urchueguía, «Finite line-source model for borehole heat exchangers: effect of vertical temperature variations,» 2009. [En línea].

[43] Badenes, Borja; Mateo Pla, Miguel A; Magraner, Teresa; Lemus, Lenin; UrchueguÍa, Javier F, «Experimental facility to perform Thermal Response Tests and study the thermal behaviour of the ground,» 2016.