Apuntes de prácticas

18/02/20

- Buscar info sobre Angular9 (aunque actualmente se usa la v.8 y es la que se usará para el proyecto)
- Hacer pruebas con una copia del proyecto iotsens-janz-telemetry, el cual tendrá una base similar. Ojo con las variables de entorno.
- Ver la configuración del proyecto
- Intentar definir requisitos para Propuesta técnica (aunque me los tienen que dar y confirmar)
- Tutorial de Angular8 (instalar CLI Angular) para ir probándolo, web: https://www.freecodecamp.org/news/angular-8-tutorial-in-easy-steps/
- Tutorial de aplicación simple con Angular 7: https://www.youtube.com/watch?v=O13X14TGtm8&t=1804s
- visitar (si fuera necesario): https://v8.angular.io/start

Proyecto:

- Definir los requisitos en función de lo que está explicado en el folio
    - Añadir por ej. consumos futuros en función del cliente
    - Agrupar por zonas y/o consumo a los clientes
    - Que el cliente quiera ponerse una alarma cuando sobrepase el umbral que haya establecido
    - Buscar más ideas
    - Extra: añadir formulario para especificar las características de ese cliente y que sirva de ejemplo para agrupar a clientes con consumos similares
    - Visualización de datos “sencilla” (como en el ejemplo)

21/02/20

- Para que un puerto deje de escuchar y no haya problemas a la hora de compilar el proyecto, se debe ejecutar el comando:

  `netstat -lnp`

  Si se quiere ser más concreto:

  `netstat -lnp -t tcp`

  Acto seguido se debe encontrar el proceso que nos interesa terminar y poner:

  `kill <nombre>`

- Consultar web: http://arquitecturajava.com para poder entender y practicar el enrutado de los componentes y servicios de Angular con Java

BBDD:

Para la BBDD, entrar en:
http://dev-mysql01.grupogimeno.com/phpmyadmin/db_structure.php?server=1&db=waterClients

user: iotsens
contraseña: adc-iotsens

También se puede entrar:

user: desarrollo
contraseña: developer.2017

10/03/20

Traductor:
Documentación orginal de la instalación y uso:
https://github.com/ngx-translate/core#installation

Apuntes sobre cómo instalar la librería ngx-translate y usarla (está mejor explicado):
https://www.codeandweb.com/babeledit/tutorials/how-to-translate-your-angular8-app-with-ngx-translate

- Traducción con y sin parámetros
- Diferentes formas de crear la tubería

12/03/20

Git Stash:
Herramienta para el control de versiones (sirve por si se quiere probar alguna tontería en un momento dado y la lías): https://git-scm.com/docs/git-stash

Rest:
Para la correcta creación de urls: https://asiermarques.com/2013/conceptos-sobre-apis-rest/

13/03/20

Wiki de IoTSens:

- Para las variables de entorno:
    - http://wiki.grupogimeno.com/confluence/pages/viewpage.action?spaceKey=MAYA&title=Variables+de+entorno+por+proyecto

    - http://wiki.grupogimeno.com/confluence/display/MAYA/Entornos

- Página principal:
    - http://wiki.grupogimeno.com/confluence/#all-updates

Variables de entorno del proyecto (por ahora)**:**

```
SPRING_DATASOURCE_URL=jdbc:mariadb://dev-mysql01.grupogimeno.com:3306/waterCli
ents
SPRING_DATASOURCE_USERNAME=desarrollo
SPRING_DATASOURCE_PASSWORD=developer.2017
```

GitLab:
Repositorio del proyecto: http://git.grupogimeno.com/iotsens/iotsens-water-clients

Gestión:
Gestión del trabajo realizado (completar todos los días en los que se trabaje):
https://partestrabajo.grupogimeno.com/

23/03/20

Se está configurando el entorno de trabajo de manera local, ya que por cuestiones sanitarias y debido a la cuarentena, se está haciendo teletrabajo. También es posible conectarse remotamente al escritorio, pero no es seguro porque puede que se apague.
El pom.xml está por ahora en el gitnore, puesto que es recomendable no cambiar la configuración original.

Computaciones:
Para hacer las computaciones y llevar un seguimiento del trabajo, se deben ir rellenando los informes diarios:
https://partestrabajo.grupogimeno.com/

- Imputación web clientes (WaterClients): GT/2040/1/DE/SW/03

24/03/20

Se están arreglando los fallos que tenía en el código de la implementación de la funcionalidad “Listar Contadores”. Se debe tener en cuenta que no hace falta poner ON DELETE ni ON UPDATE a las tablas (archivos de la BBDD, *.sql), automáticamente ya lo gestiona. No es lo mismo que RESTRICT pero funciona prácticamente igual. También han habido otros fallos, tales como la comprobación del usuario a quién se le debe sacar la

información de sus contadores. No es necesario comprobar constantemente lo mismo, es conveniente añadir un Validate.
IMPORTANTE: Tener en cuenta que a la hora de poner routerLink, se deben poner ../{path} si se invoca desde páginas que no sean el main (ya que éstas son hijas de main).

Jira IOTSens:
Backlog de Jira:
http://jira.grupogimeno.com/jira/secure/RapidBoard.jspa?rapidView=150&view=planning&selectedIssue=SSW0008-19&quickFilter=944

RxJs:
Para aprender a cómo hacer llamadas al Servidor y la aplicación del patrón Observer en una API: https://rxjs-dev.firebaseapp.com/

25/03/20

En las inyecciones de dependencias en los beans de Spring, siempre se debe usar el constructor y no usar el @Autowired (en el caso de mi proyecto, los Servicios se deben inicializar en el constructor del controlador).

IMPORTANTE: el modelo del backend y el frontend debe ser igual, esto quiere decir que hasta los atributos se deben llamar igual.

Ejemplo de paso de parámetros:

```
//WatermeterService.ts

getWatermeter(meterId: number){
    let params = new HttpParams()
        .set("meterId", String(meterId));
    return this.http.get<Watermeter>("services/watermeters/", {params: params});
 }

//params son @ParamRequest en el backend (consultar el proyecto telemetry)
```

Se debe tener en cuenta que si se pasa un @PathVariable, está implícita en la ruta, simplemente hay que añadirla a la cadena que indique la dirección.

Enlace de interés:
https://en.wikipedia.org/wiki/Percent-encoding

Cambio de Traductor:
Se debe cambiar el traductor que se está empleando y buscar información sobre éste:
https://github.com/ngneat/transloco

26/03/20

No se ha podido avanzar mucho en el proyecto. Está dando problemas el diseño del frontend, en concreto la barra de navegación. También se ha investigado más sobre el nuevo traductor a usar.
Ya están preparadas las dos variables de entorno a usar para acceder, a través de los micros servicios, a el consumo diario, promedio, etc. de un contador.

27/03/20

Se ha realizado la migración de ngx-translate a transloco, el nuevo traductor que se usará a partir de ahora. Se han estado corrigiendo los fallos que se debían corregir.

RECOMENDACIÓN: Es mejor usar *directive* en vez de *pipe* con Transloco, puesto que es lo que se recomienda. Además, no hace falta poner <... translate>, con añadir correctamente la traducción es suficiente.

RECOMENDACIÓN: Es mejor usar NamedParameterJdbcTemplate, puesto que evita más errores, ya que tú nombras al parámetro y luego pones dicho nombre en la *query*, en vez de tener que estar pendiente de el orden en el que va cada parámetro. Está implementado ya en las consultas a la BBDD en el proyecto WaterClients. Se debe tener en cuenta que el orden de los parámetros de la función cambia.

Spring JDBC:
Tutorial para saber cuándo es más apropiado usar JdbcTemplate o NamedParameterJdbcTemplate: https://www.baeldung.com/spring-jdbc-jdbctemplate

---

30/03/20

He empezado con el uso de los servicios que llamarán a la api de IoTSens para que muestre los datos de un contador. Aún falta hacer métodos con los que recoger dicha información y sacarla por pantalla.

```
ID Sensor (real): Q18BC001962
```

IMPORTANTE: Los @Bean (están encima de algún método de una clase marcada como @Configure) se pueden usar haciendo @Autowired en la clase (ej. Pepito) que los vaya a usar. Sin embargo, es mejor que en el constructor de dicha clase (Pepito), sea quien pase como parámetros estos beans (en ese caso no hace falta poner ninguna anotación).

Variables de lectura de contadores:

- lectura: CURRENT_VALUE
- consumo horario: FORWARD_FLOW

La lectura es el total de m3 que han pasado por el contador, el consumo es los m3 que han pasado en un período de tiempo (la diferencia entre la lectura al final y al inicio del período).

31/03/2020

Comunicación entre *Components*:

- Documentación de AngularJS para saber cómo crear comunicaciones entre componentes (de padre a hijo y viceversa): https://angular.io/guide/component-interaction
- Tutorial para ver un ejemplo de la comunicación entre *components*: https://www.youtube.com/watch?v=I317BhehZKM

Han surgido varias dudas durante la implementación de MeasuresService y la parte del *frontend*.

- ¿Sería conveniente en el path del WatermeterController añadir meter? Es una forma de que no se tenga que volver a consultar la BBDD y sólo se ejecute la api de IoTSens.
- ¿Sería conveniente hacer un Enum de las diferentes variables que se pueden consultar sobre los contadores?
- ¿Está bien creado el componente hijo (watermeter-measures) del componente padre (watermeter)?

A pesar de estas dudas, ya he añadido una clase Enum TypeMeasure por si hiciera falta. En el *frontend* he creado un path hijo (watermeter-measures.component) de watermeter.component. Todavía se debe probar si funciona. He consultado varios tutoriales para entender la comunicación entre componentes. También he creado la clase Measure en el *frontend* para poder visualizar los datos por pantalla.
Por ahora no funciona sacar por pantalla las medidas del sensor, hay algo que hago mal a la hora de pasar información de un componente a otro. Por otro lado, he añadido el *spinner* para que aparezca mientras se hacen las consultas, aunque son tan rápidas que ni se llega a ver por pantalla.

01/04/20

Streams de Java8:
Muy útiles para la transformación de tipos o cambio de listas a mapas/listas con filtros. Son más rápidos y no gastan tanta memoria ni se debe escribir un código extenso.

- Guía de las diferentes opciones que hay: https://stackify.com/streams-guide-java-8/

- Ejemplo de transformación de tipos con filtrado: https://stackoverflow.com/questions/35650974/create-list-of-object-from-another-using-java-8-streams/35651357#35651357
- En el proyecto se utiliza en la clase MeasuresService.

DatePicker de AngularJS:
Para elegir una fecha a la hora de mostrar datos en la gráfica:
https://material.angular.io/components/datepicker/overview

PieChart de AngularJS:
Por ahora se ha instalado la librería @swimlane/ngx-charts, aunque la mayoría de los tutoriales están para chart.js (a mi me convence más ngx-charts).

- Documentación para instalar la librería @swimlane/ngx-charts: https://swimlane.gitbook.io/ngx-charts/
- Demo para probar esta librería: https://stackblitz.com/edit/swimlane-pie-chart-grid?embed=1&file=app/app.component.ts
- Tutorial para hacer un gráfico de barras con esta librería: https://medium.com/@swathisprasad/building-data-visualization-with-angular-and-ngx-charts-375942d49fd6
- Documentación de charts.js: https://www.chartjs.org/docs/latest/
- Muestras de *charts* con esta librería: https://www.chartjs.org/samples/latest/
- Tutorial de los diferentes tipos de *charts* que se pueden crear con la librería charts.js:
    - https://www.youtube.com/watch?v=HqFNZ1oz_m4
    - https://www.youtube.com/watch?v=ekp1Odf027s
- Tutorial de cómo recoger los datos de la api para luego añadirlos al *chart*: https://www.youtube.com/watch?v=RTzi5DS7On4

Kibana:
Documentación y qué es (parecido al Postman): https://www.elastic.co/es/kibana
Kibana de IoTSens: http://elk-dev01.iotsens.com/app/kibana#/dashboards?_g=()
Ejemplo de uso:
http://elk-dev01.iotsens.com/app/kibana#/discover?_g=(refreshInterval:(pause:!t,value:0),time:(from:'2020-04-01T14:39:16.080Z',to:'2020-04-01T14:39:16.081Z'))&_a=(columns:!(message),filters:!(('$state':(store:appState),meta:(alias:!n,disabled:!f,index:'1ca993f0-7226-11e9-bf2a-f7940ed351fe',key:environment,negate:!f,params:(query:iot-dev),type:phrase,value:iot-dev),query:(match:(environment:(query:iot-dev,type:phrase))))),index:'1ca993f0-7226-11e9-bf2a-f7940ed351fe',interval:auto,query:(language:kuery,query:'level:error%20and%20requester_user:%22waterclients.app%22'),sort:!('@timestamp',desc))

IMPORTANTE: La clase Date.java ya se ha quedado un poco desfasada, en el proyecto está la que actualmente se utiliza, DateFormat (sirve aunque el tipo de dato que te pidan sea Date).

- Ejemplo de uso de DateFormat: http://tutorials.jenkov.com/java-date-time/parsing-formatting-dates.html

Han habido problemas con la conexión al Microservicio de IoTSens, al principio estaba caído y no tengo los permisos necesarios para acceder (lo están reparando).

02/04/20

Intellij:
Atajos de teclado: http://raulavila.com/2015/02/atajos-teclado-ide/

IMPORTANTE: Se debe seguir teniendo muy en cuenta el estilo REST a la hora de crear las rutas.

Posiblemente se cree una ruta en el proyecto luego para indicar qué medida deseamos sacar, por ahora sólo se necesita saber que se puede acceder al contador y ver su información. La posible ruta sería: *"watermeters/{meter}/measures/{variable}*. Por otro lado, no se ha podido adelantar mucho, sigo sin poder compartir información de un componente a otro.

Posible fuente de texto para usar: Lato

03/04/20

Por fin se realizan las consultas a la api de IoTSens. El contador que me han facilitado sólo tiene medidas a partir del día 04/03/20. Se ha decidido que por defecto se saque las medidas del último mes. Por defecto hay medidas cada hora, debemos quedarnos con la media diaria.

IMPORTANTE: Buscar información de cómo usar ngOnChanges (fase de Angular), para que cada vez que haya un cambio de información venida desde el componente padre, se actualicen los datos en el hijo. Con ello puede que se solucione el problema de paso de información entre componentes. Se debe mirar también el tema de los eventos.

Fases de Angular
Ejemplo de cómo pasar la información de padre a hijo, teniendo en cuenta la fase ngOnChanges:
https://angular.io/guide/component-interaction#pass-data-from-parent-to-child-with-input-binding

Moment.js:
Se puede ver un ejemplo en el proyecto Telemetry

- Formatos de la función *moment()* de Angular: https://momentjs.com/

- Cosas chulas que se pueden hacer con la librería: https://programacion.net/articulo/manejar_fechas_y_horas_como_un_profesional_con_moment_js_1301

RECOMENDABLE: Se puede usar la anotación @DateTimeFormat para obtener fechas de las peticiones. Hay un ejemplo en los dos proyectos (Telemetry y WaterClients)

Falta por acabar el formateo y paso de las fechas para buscar en la api de IoTSens.

06/04/20

Se ha solucionado, más o menos, el problema del paso de información entre componentes. Convendría mirarlo y seguramente cambie la estructura debido a que en el listado de contadores se mostrarán algunas de las medidas.

07/04/20

Se ha continuado con el *chart* para poder imprimir los resultados de los sensores.

IMPORTANTE: El nombre de los *params* que se añaden en el *frontend* tienen que tener el mismo nombre que en *backend*. Por ejemplo: "*dateUntil"* en el *watermeter.service.ts* del *fromtend* tiene que ser igual a *"@RequestParam dateUntil"* de la clase *WatermeterController.java* del *backend*.

NOTA: *measureRequestBuilder.withLimit(3)* indica la cantidad máxima de medidas que se quiere mostrar por pantalla.

SensorId (real): Q18BC001962

IMPORTANTE: Para que no dé error cuando sacamos por pantalla una variable, se debe tener en cuenta una variable que indique que los datos han sido procesados. En el observable (*subscribe* en el componente), después de procesar los datos, cambiaremos la variable a true. Luego, en el formulario se puede hacer un **ngIf* de esa variable para ver si ya se pueden mostrar los datos recogidos.

08/04/20

IMPORTANTE: Si se quiere poner el **ngIf*, mejor que no se ponga incluyendo la parte en la que se pasa el dato del componente padre al hijo, no mostrará las medidas.

Hoy se ha creado el formulario básico para recoger las fechas y mostrarlas en la gráfica. Falta hacer un validador y que se pueda elegir también el tipo de dato y los intervalos de tiempo.

Git Stash en Intellij:
Cómo hacer una copia de los datos antes de cambiar de rama (Git Stash):
https://www.jetbrains.com/help/idea/work-on-several-features-simultaneously.html

Builder de la API IoTsens:
Cómo usar y hacer los *builders*:
http://git.grupogimeno.com/iotsens/iotsens-water-clients/merge_requests/7/diffs?commit_id=1ccdc249f50d0861978433ee0bfd06a8517be99f

---

09/04/20

Han habido una serie de problemas con la BBDD y la migración del proyecto en una aplicación multi-tenant, así que no se ha podido adelantar tanto como se esperaba.

Se han creado ya los dos *select* o *dropdown* para elegir tanto los intervalos de tiempo (días, meses, años…) como el tipo de datos a mostrar (media, máximo, mínimo…). También se han traducido el de intervalos de tiempo, aunque de manera “churrera”.

IMPORTANTE: Se debe tener en cuenta las limitaciones de los *enums* en Typescript. También tener en cuenta que los *selects* deben de formar parte del formulario para que cojan el dato, es decir, se debe añadir un campo en el *form* que referencie al dato que será elegido en el *select*.

14/04/20

Se ha tenido que cambiar lo de las operaciones, por ahora sólo se puede hacer una operación. Por tanto, se eliminará en el html el *select* para elegir dicha opción. Por contra, se ha añadido un *select* con las variables que podemos usar (que son sólo 2).

NOTA: Se debe tener en cuenta que el diseño sea parecido (en el aspecto “bonito”) de la app de SmartWater (tengo una captura guardada).

Operación WaterClients:
Única operación que se usará para la aplicación (conjuntamente con la variable CURRENT_VALUE): DIFFERENCE_PREVIOUS

Al final se ha cambiado toda lo referente al formulario usado para visualizar datos en la gráfica. Se ha cambiado por dos gráficas:

- La que indica el consumo promedio (CURRENT_VALUE + DIFFERENCE_PREVIOUS)
- La que indica la lectura (CURRENT_VALUE + LAST)

15/04/20

Se ha buscado un plugin o librería que se pudiera añadir a la librería charts.js, pero el que se ha encontrado no es muy bueno.

Se ha acabado de añadir la gráfica que faltaba y se ha hecho un cálculo a mano del consumo promedio.

16/04/20

Se ha añadido ya la visualización de la media diaria y la diferencia con la misma. Están habiendo problemas con la implementación de los componentes en la página, ya que no se quedan estructurados como debería ser.

17/04/20

NOTA: Para eliminar los *imports* que sobren y los reordena de una clase en el Intellij, se puede usar el comando: **ctr-alt-o**

Tareas a hacer:
- Personalizar el *subscribe*, que la función devuelva lo que tenga que devolver para cada uno de los datos. ok
- Cambiar *moment()* por *new Date()* cuando haga falta. ok
- Comprobar el *Login* para que esté mejor estructurado. ok
- Mejor estructuración de la información de un contador. ok
- De la clase *main*, se debe cambiar *nav* por *row*. ok

20/04/20

Todo el tiempo se ha dedicado a mejorar el front-end.

21/04/20

Se ha estado cambiando la visualización de la lista de contadores, corrigiendo errores y sacando por pantalla el consumo diario.

22/04/20

Se ha estado implementando la lista de alarmas y corrigiendo errores a la hora de mostrar el consumo diario promedio, la diferencia y el consumo de hoy, puesto que si no hay datos, tampoco te salía un mensaje de error.

mañana: hacer un formulario para sacar las alarmas, poner de qué a qué fechas se quieren ver

- activas: si, no, todas
- período fechas
- tipo de alarma, este lo dejaremos de momento, haz los dos anteriores

---

23/04/20

Se ha acabado de implementar la funcionalidad básica de listar alarmas. También, se ha estado implementando el formulario para buscar alarmas, el cual ya funciona correctamente.

Hay varios problemas que se deben preguntar:

- La librería de *materials* de Angular no está instalada
- Al intentar instalarla ha dado varios errores

23/04/20

Se ha mejorado la visualización de la lista de alarmas y el formulario. Han habido problemas a la hora de implementar el mat-paginator, así que se ha optado por usa el psginator de bootstrap (aunque no es tan visualmente agradable y personalizable).

Se han mejorado algunos aspectos del funcionamiento interno a la hora de conseguir datos que eran un poco “chapuceros”.

24/04/20

27/04/20

Se ha empezado con la mejora de la lista de contadores, añadiendo los tipos de alarma que actualmente tienen activas los contadores y su consumo de hoy.

28/04/20

Están habiendo problemas a la hora de mostrar el consumo de hoy para cada contador, ya que pueden existir problemas de concurrencia a la hora de añadir un resultado en el *Map*, es por ello que se ha estado probando con el Mutex de Angular, para ver si con ello se resuelve el problema de acceso concurrente.

29/04/20

Al final el Mutex no ha servido y se ha vuelto a dejar como estaba. Se ha customizado la visualización de alarmas en la lista de contadores. Falta hacer la gráfica, que casi está, hay que cambiar parte de la toma de datos y añadir un método para crear la gráfica.

30/04/20

Se han estado corrigiendo errores y continuando la mejora de la lista de contadores, añadiendo la gráfica respectiva para cada contador.
También se ha empezado con la HU11, que consiste en añadir información adicional sobre los contadores. Está creado todo el *backend* y añadida la tabla necesaria con algunos datos en la BBDD.

IMPORTANTE: En MySQL, los valores booleanos (o tinyint) son un poco especialitos, realmente son enteros que pueden adopar el valor de 1 o 0, aunque realmente pueden ser 2, 3, 4, etc hasta lo que ocupe 1 *byte*.

Boolean en MySQL:
Tutorial en el que te explica cómo tratar los *boolean (tinyint)* en MySQL:
https://www.mysqltutorial.org/mysql-boolean/
Añadir toggles: Botoncito con on/off

04/05/20

Se arreglaron algunos errores del *backend* para que se pudiera recoger correctamente la información addicional de un contador. También se empezó a hacer el *frontend*, aunque hubo algunos problemas con el enlace para ir a la página del formulario con los datos adicionales. (4h)

05/05/20

Se ha seguido con el *frontend*. Hay problemas a la hora de mostrar los *swich*, puesto que van con retardo a la hora de actualizar su estado y no se puede solventar poniendo un **nglf*.

También se ha creado un formulario básico con botones tipo *toggle/switch* y varios campos para indicar cantidad de personas, niños y baños. También se ha traducido toda la página. Han habido algunos problemas a la hora de crear el formulario, ya que primero se debían recoger los datos y rellenarlo por defecto con esos datos (en caso de tener esa información en la BBDD).

Falta añadir un validador de números y traducir el botón para generar un nuevo formulario.

(7h)

06/05/20

Se ha estado arreglando la muestra de algunos datos para que queden bonitos. Se ha cambiado la BBDD y la migración para que cada vez que se inserte un contador se genere su correspondiente fila en la tabla AdditionalWatermeterData con valores por defecto.

También se ha cambiado en el *frontend* la vista correspondiente al formulario de información adicional para que siga la nueva estructura.

Además, se ha cambiado el *backend* en función de la nueva estructura.

(9h)

07/05/20

Han seguido habiendo problemas con el *frontend*, pero por fin se han solucionado.

En el *frontend*, con respecto a la vista de la información adicional de un contador, se ha añadido el título y un botón para volver a atrás/cancelar. También se han estado arreglando y cambiando algunas cosas con respecto a la obtención y tratamiento de datos.

08/05/20

Se han estado corrigiendo errores la mayoría del tiempo. También se ha contemplado las diferentes posibilidades de aviso al usuario en caso de hacer una consulta *put/post/get* y que, por algún motivo (sin conexión con la BBDD, fallo de la api, etc), no se pueda realizar. Al final, por ahora se ha dejado de lado y ya se hará cuando todo esté integrado para que sea homogéneo.

IMPORTANTE: las anotaciones @ControllerAdvice y @ExceptionHandler son útiles para hacer excepciones específicas si se quiere notificar al usuario. Por ahora no será necesario, pero es mejor tenerlo en cuenta.

11/05/20

Se ha hecho una reunión para ver qué faltaba por hacer del proyecto y qué me daría tiempo a acabar. En principio, se me ha asignado hacer la visualización de un mapa con los contadores. En caso de acabar antes de estas dos semanas, empezaré con la parte de *Machine Learning* con casos ficticios, ayudándome de la BBDD y de la api de IoTsens para que me proporcione las medidas a lo largo de un año.

Han habido dificultades con la instalación del *plugin* que tiene IoTsens para la visualización de mapas (que a su vez implementa la librería LeafLet). Finalmente se ha podido instalar pero aún hay problemas a la hora de intentar usarla.

12/05/20

Se ha continuado con la instalación de la librería que tiene IoTsens para la visualización de mapas. Finalmente se ha podido usar más o menos gracias a la ayuda de Jose, aún así todavía no muestra el mapa correctamente.

13/05/20

Se ha continuado probando la librería de IoTsens, pero han habido problemas y, puesto que solo queremos el mapa para mostrar los contadores, al final se ha optado por probar la librería Leaflet. Por ahora aún no funciona del todo, pero es más fácil de manejar que la otra librería.

_____

18/05/20

Han habido problemas con la licencia del Intellij y con el visionado del mapa. (4h)
Se ha continuado con la parte de *machine learning*, viendo tutoriales para saber usar la librería sklearn de Python. (5h)

Falta también saber si los marcadores del mapa se deberían guardar en la BBDD o consultar directamente a la api de IoTsens.

_____

19/05/20

Se ha podido visualizar el mapa y se han hecho las configuraciones iniciales.

Finalmente se ha podido hacer el *backend* y el *frontend* de toda la funcionalidad respectiva a la HU13, visualizar las coordenadas de los contadores en el mapa. (8h)

_____

20/05/20

He estado corrigiendo todos los errores que me ha estado comentado Roberto y haciendo algunas comprobaciones con respecto a unos resultados inusuales sobre las lecturas de los contadores que ha devuelto la api de IoTsens. También he probado todas las funcionalidades implementadas hasta el momento y he detectado un pequeño problema a la hora de actualizar los datos adicionales de un contador, pero ha quedado solucionado.

_____

21/05/20

He añadido pop-ups con información al mapa, para que cuando el usuario pinche en el marcador pueda ver la información del contador. Para ello, he tenido que cambiar parte del backend y del frontend relacionadas con esa funcionalidad. También, mientras Roberto revisaba el proyecto, he estado preparando algunos datasets de prueba para poder proseguir con la parte de machine learning.

_____

22/05/20

Puesto que es el último día y no me iba a dar tiempo a probar ni implementar nada con respecto a la parte de previsiones y todo lo relacionado con el machine learning, me he

dedicado a probar enteramente toda la aplicación de nuevo y asegurarme de que funcione correctamente. También he grabado una demo mostrando todo lo que he ido implementando hasta ayer.

Computaciones:
Para hacer las computaciones y llevar un seguimiento del trabajo, se deben ir rellenando los informes diarios:
https://partestrabajo.grupogimeno.com/

- Imputación web clientes (WaterClients): GT/2040/1/DE/SW/03

*ver qué es tensor flow (google)